ANAIS

SBSeg 2017

XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

6 A 9 DE NOVEMBRO - BRASÍLIA - DF

ANAIS

Editora
Sociedade Brasileira de Computação - SBC

Organizadores
Edna Dias Canedo, UnB
Rafael Timóteo de Sousa Júnior, UnB

Realização
FGA/UnB – Faculdade UnB Gama, Universidade de Brasília
ENE/UnB – Departamento de Engenharia Elétrica, Universidade de Brasília

Promoção
SBC – Sociedade Brasileira de Computação

Copyright © 2017 Sociedade Brasileira de Computação
Todos os direitos reservados

Capa: Diego Oliveira, INCT em Segurança Cibernética, ENE/UnB

Editoração: Daniel Alves Tavares, INCT em Segurança Cibernética, ENE/UnB
Diego Martins de Oliveira, INCT em Segurança Cibernética, ENE/UnB
Edna Dias Canedo, INCT em Segurança Cibernética, ENE/UnB
Rafael Timóteo de Sousa Junior, INCT em Segurança Cibernética, ENE/UnB
Robson de Oliveira Albuquerque, INCT em Segurança Cibernética, ENE/UnB

Dados Internacionais de Catalogação na Publicação (CIP)

SI602 Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (17. : 2017 : Brasília, DF)

Anais [do] XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 6 a 9 de novembro de 2017; Organizadores, Edna Dias Canedo; Rafael Timóteo de Sousa Júnior. – Brasília, DF: Sociedade Brasileira de Computação, 2017.

804 p.

Título: XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais: SBSEG 2017: Anais

ISBN: 978-85-7669-422-96

1. Ciência da computação. 2. Informática. 3. Segurança da Informação. 4. Segurança de sistemas. 5. Criptografia. I. Canedo, Edna Dias. II. Sousa Júnior, Rafael Timóteo. III. Título.

CDU 004(082)

Ficha catalográfica elaborada por Elen Sandra Rocha de Oliveira – CRB 1/1875
Laboratório LATITUDE/UnB – Projeto DIPLA/FUB (TED MP/DIPLA 005/2016)

Sociedade Brasileira de Computação – SBC

Presidência
Lisandro Zambenedetti Granville (UFRGS), Presidente
Thais Vasconcelos Batista (UFRN), Vice-Presidente

Diretorias
Renata de Matos Galante (UFRGS), Diretora Administrativa
Carlos André Guimarães Ferraz (UFPE), Diretor de Finanças
Antônio Jorge Gomes Abelém (UFPA), Diretor de Eventos e Comissões Especiais
Avelino Francisco Zorzo (PUC-RS), Diretor de Educação
José Viterbo Filho (UFF), Diretor de Publicações
Claudia Lage da Motta (UFRJ), Diretora de Planejamento e Programas Especiais
Marcelo Duduchi Feitosa (CEETEPS), Diretor de Secretarias Regionais
Eliana Silva de Almeida (UFAL), Diretora de Divulgação e Marketing

Diretorias Extraordinárias
Ricardo de Oliveira Anido (UNICAMP), Diretor de Relações Profissionais
Esther Colombini (UNICAMP), Diretora de Competições Científicas
Raimundo José de Araújo Macêdo (UFBA), Diretor de Cooperação com Sociedades Científicas
Cláudia Cappelli (UNIRIO), Diretora de Articulação com Empresas

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219 - Bairro Agronomia
91.509-900 – Porto Alegre RS

CNPJ: 29.532.264/0001-78
http://www.sbc.org.br

© 2017 Sociedade Brasileira de Computação

Mensagens das Coordenações do SBSeg 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação Geral do SBSeg 2017

Boas-vindas aos participantes do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2017), que pela segunda vez é realizado em Brasília. Sem dúvida é uma grande honra para nossas instituições, a Universidade de Brasília – UnB com o apoio do Instituto Federal de Brasília – IFB, acolher o evento científico brasileiro que se tornou o centro da discussão sobre avanços realizados e desafios a serem superados na área de segurança da informação e de sistemas computacionais. A realização de um evento desse porte e de tal importância para a comunidade científica representou um desafio maior em um período marcado por inúmeras dificuldades vividas pelo país e pelas instituições federais de ensino. Cabe assim destacar o papel da Área de Engenharia de Redes de Comunicação do Departamento de Engenharia Elétrica (ENE) da Faculdade de Tecnologia (FT), bem como da Área de Engenharia de Software da Faculdade do Gama, ambas da UnB, que apoiaram desde o início a realização do evento.

A programação técnica do SBSeg 2017 conta com uma rica variedade de atividades de grande qualidade científica e tecnológica. São 13 sessões técnicas nas quais serão apresentados 34 artigos completos e 12 resumos estendidos, selecionados por meio de um rigoroso trabalho de revisão. A programação inclui quatro palestras proferidas por renomados pesquisadores internacionais e duas palestras por pesquisadores nacionais, abordando temas extremamente atuais nos tópicos do simpósio. São oferecidos quatro minicursos, voltados à formação e atualização dos participantes sobre tópicos selecionados através de chamada pública. O SBSeg 2017 apresenta também seis workshops focados em temas específicos e emergentes, relacionados à temática do Simpósio, um Salão de Ferramentas e um Fórum de Segurança Corporativa.

Um aspecto fundamental do SBSeg é o comprometimento com a qualidade, especialmente graças à composição de Comitês de Programa por pesquisadores brasileiros e estrangeiros com grande renome e inserção na área. Agradecemos por esse dedicado trabalho aos coordenadores de cada comitê: Paulo Lício de Geus (IC-Unicamp) e Rossana Maria de Castro Andrade (DC-UFC) – Comitê de Programa do SBSeg 2017, Márjory Da Costa Abreu (DIMAp-UFRN) – XI Workshop de Trabalhos de Iniciação Científica e de Graduação – WTICG, Raphael Carlos Santos Machado (INMETRO) – III Workshop de Regulação, Avaliação da Conformidade e Certificação de Segurança – WRAC+, Diego F. Aranha (Unicamp) e Jeroen van de Graaf (UFMG) – III Workshop de Tecnologia Eleitoral – WTE, Marco Aurélio Amaral Henriques (Unicamp) – VII Workshop de Gestão de Identidades Digitais – WGID, Díbio Leandro Borges (CIC-UnB) e Vinicius Ruela Pereira Borges (CIC-UnB) – VI Workshop de Forense Computacional – WFC, Raul Ceretta Nunes (UFSM) – Minicursos – MC, Ricardo Felipe Custódio (UFSC) e Routo Terada (IME-USP) – Palestras e Tutoriais, Daniel Macêdo Batista (IME-USP) – Salão de Ferramentas – SF, e Laerte Peotta de Melo (Banco do Brasil) – Fórum de Segurança Corporativa – FSC.

Para a realização do SBSeg 2017, foram fundamentais o envolvimento e a colaboração de várias pessoas e entidades. Em especial, gostaríamos de agradecer aos membros do comitê de organização local, por conta de seu trabalho voluntário e incansável. Gostaríamos de agradecer à diretoria da Sociedade Brasileira de Computação, à Coordenação da Comissão Especial em Segurança da Informação e de Sistemas Computacionais da SBC (CESeg) e ao Comitê Consultivo do SBSeg, pela confiança depositada e pelo permanente suporte e incentivo nos momentos de dificuldade. Agradecemos ainda aos patrocinadores

© 2017 Sociedade Brasileira de Computação

do simpósio: Comitê Gestor da Internet no Brasil (CGI.br), CNPq, CAPES, FAPDF, a Associação Nacional dos Peritos Criminais Federais – APCF e a empresa Kryptus. Agradecemos à equipe administrativa da SBC pelo competente apoio organizacional prestado. Por fim, nossos agradecimentos aos alunos, técnicos e professores do Laboratório de Engenharia de Redes (LabRedes), Laboratório de Tecnologias da Tomada de Decisão (Latitude) e Programa de Pós-Graduação em Engenharia Elétrica (PPGEE), da UnB, bem como os membros do IEEE ComSoc Centro-norte Brasil Chapter, por viabilizarem a realização de um evento do porte do SBSeg.

No período de 6 a 9 de novembro 2017 estão reunidos em Brasília estudantes, professores, pesquisadores, governo e profissionais da área, todos com o objetivo de trocar ideias, compartilhar experiências e estabelecer laços pessoais, objetivando a discussão sobre avanços realizados e desafios a serem superados na área de segurança da informação e de sistemas computacionais. Sejam bem-vindos ao Planalto Central e desfrutem de uma estada agradável e um evento proveitoso!

Edna Dias Canedo (UnB)
Rafael Timóteo de Sousa Júnior (UnB)
Coordenadores Gerais do SBSeg 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação do Comitê de Programa

O Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) chega a sua décima sétima edição como um dos importantes eventos científicos do país, servindo como ponto de convergência para a extensa comunidade da área. Prova disso foi o alto número de submissões de artigos completos, 94 no total, dos quais 34 foram aceitos. De modo semelhante, na trilha de artigos curtos tivemos um número elevado de submissões, 34 artigos, dos quais 12 foram aceitos.

O alto números de submissões em ambas as trilhas tornou o processo de seleção bastante laborioso, mas mesmo assim garantindo um mínimo de 3 e um máximo de 5 avaliações por artigo para os artigos completos. A mesma comunidade que submeteu seus resultados de pesquisa participou do processo de seleção em avaliação cruzada às cegas, à qual agradecemos pela dedicação e envolvimento, tanto em confiar tais resultados a este evento quanto em dispender considerável quantidade de tempo na avaliação dos artigos de seus pares.

Dentre os 394 autores que submeteram artigos (incluindo repetições de múltiplas autorias), 379 são filiados a instituições brasileiras e 9 são de instituições de 6 países estrangeiros (outros 6 preferiram não identificar sua filiação). O Comitê de Programa contou com 57 pesquisadores da comunidade, filiados a universidades do país inteiro, além de algumas organizações públicas e empresas. Os membros do comitê se juntam a outros pesquisadores para compor um total de 126 avaliadores de trabalhos submetidos e gerar 493 pareceres, os quais foram submetidos a uma fase de discussão para aumentar o grau de concordância entre opiniões diferenciadas, no caso dos artigos completos. O trabalho do comitê ainda continua, com várias atividades complementares sendo realizadas até durante o evento em si, incluindo coordenação de sessões técnicas, organização de workshops, seleção de melhores, entre outros.

Agradecemos também à Coordenação Geral do SBSeg 2017, nas pessoas dos professores Edna Dias Canedo e Rafael Timóteo de Sousa Junior, por nos confiar a tarefa de coordenadores do Comitê de Programa, sempre pacientes com nossas demandas. Vários outros colaboradores da Coordenação Geral também merecem nossos agradecimentos por ajudar na execução das trilhas de artigos técnicos. Finalmente, agradecemos a presença de todos os participantes, sem a qual o objetivo principal do evento, que é a integração da comunidade, não aconteceria. Sejam bem-vindos e tenham um excelente SBSeg!

Paulo Lício de Geus (IC-Unicamp)
Rossana Maria de Castro Andrade (DC-UFC)
Coordenadores do Comitê de Programa do SBSeg 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação do Salão de Ferramentas 2017

Esta é a primeira edição do Salão de Ferramentas do SBSeg e me sinto honrado em ter sido convidado para ser o coordenador do evento. O objetivo do Salão é servir como um fórum de encontro de desenvolvedores(as) e pesquisadores(as) interessados(as) na implementação e avaliação de desempenho de sistemas e ferramentas que contribuem com o avanço do conhecimento da área de segurança da informação e de sistemas computacionais.

Nesta primeira edição seis (6) ferramentas foram registradas, três (3) submetidas e duas (2) aceitas para serem apresentadas no evento. Cada submissão recebeu três revisões. Aproveito para agradecer aos membros do comitê de programa que auxiliaram tanto nas revisões quanto em todas as decisões acerca do formato do evento. Agradeço também aos autores e às autoras que reconheceram a relevância do Salão nesta primeira edição.

As apresentações das ferramentas serão realizadas em dois momentos. Inicialmente haverá demonstrações em stands onde os presentes poderão ver as ferramentas em funcionamento e tirar as mais diversas dúvidas sobre seus usos e implementações diretamente com os(as) autores(as). Num segundo momento as ferramentas serão apresentadas no formato tradicional de apresentações de artigos em seções técnicas. Espera-se que esses dois formatos de apresentação sejam úteis para a comunidade de Segurança conhecer as ferramentas desenvolvidas e passar a colaborar com seus desenvolvimentos daqui para frente, já que os códigos de ambas estão disponíveis.

As duas ferramentas aceitas nesta edição do Salão de Ferramentas estão relacionadas com Segurança em Redes de Computadores e com Computação em Nuvem. Uma delas tem por objetivo detectar e prevenir intrusões em tempo real com base em processamento por fluxo e a outra tem por objetivo automatizar o provisionamento de servidores para competições de segurança de Capture-the-Flag (CTF).

Espero que toda a comunidade presente no SBSeg possa aproveitar ao máximo o Salão de Ferramentas em Brasília.

Gostaria de finalizar esta mensagem agradecendo aos coordenadores gerais do SBSeg 2017, Edna Dias Canedo e Rafael Timóteo de Sousa Junior pelo auxílio durante toda a organização do Salão, desde a primeira versão do website até todo o trabalho em busca de patrocínio.

Daniel Macêdo Batista (IME-USP)
Coordenador do Salão de Ferramentas do SBSeg 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação do WTICG

O Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG) é um evento itinerante, organizado em conjunto com o Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg). Neste ano de 2017, o evento será realizado na cidade de Brasília. O objetivo do workshop é promover a divulgação de trabalhos científicos nas áreas de segurança da informação e de sistemas computacionais em todos os seus âmbitos. O WTICG é destinado aos potencias futuros pesquisadores das áreas citadas anteriormente, que estejam na condição de estudante de graduação ou recém-graduados no período da submissão.

Nesta edição o WTICG contou com a valiosa colaboração de 25 revisores de alta competência técnica, que após o processo de avaliação de 21 trabalhos submetidos na categoria de trabalho completo selecionou 9 desses trabalhos para apresentação no Workshop.

Após as apresentações orais do WTICG serão anunciados os três melhores trabalhos completos.

Neste ano, consolidamos a implementação do CESeg Student Grants, um programa da CESeg (Comissão Especial em Segurança da Informação e de Sistemas Computacionais da Sociedade Brasileira de Computação) que tem por objetivo estimular a participação de estudantes no WTICG. O estímulo foi realizado por meio do subsídio da inscrição dos estudantes de graduação ou recém-graduados que tiveram trabalhos aceitos no WTICG.

Eu gostaria de agradecer aos coordenadores gerais, Profa. Edna Dias Canedo e Prof. Rafael Timóteo de Sousa Junior, pela confiança depositada em mim para coordenar este workshop, aos autores pela escolha do WTICG para submeter seus trabalhos, aos revisores e comitê de programa pelo trabalho ótimo de revisão.

Gostaria de dizer também que foi muito interessante e prazeroso para mim conduzir os trabalhos do WTICG e espero que todos aproveitem o evento ao máximo.

Márjory Da Costa Abreu (DIMAp-UFRN)
Coordenadora do WTICG 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação do WGID

Já na sétima edição, o Workshop de Gestão de Identidades Digitais (WGID) é um evento do SBSeg que objetiva ser um fórum para discussões sobre o estado da arte em gestão de identidades digitais, identificar os principais desafios de pesquisa e aproximar os grupos de pesquisadores atuantes na área.

Nesta edição, o programa do workshop continuará contando com apresentações de artigos científicos completos e resumos estendidos cuidadosamente selecionados com base em pelo menos três revisões por trabalho feitas pelos membros do Comitê de Programa específico do workshop. Os artigos tratarão de temas atuais e relevantes para a área de gestão de identidades como auto-adaptação de infraestruturas de autenticação, sistemas de gestão de identidades baseados em blockchain, segurança em assinaturas digitais, uso da infraestrutura eduroam como testbed e análise de custo de autenticação em redes veiculares, dentre outros. As ideias apresentadas nos artigos são bastante diversificadas e inovadoras e convidamos todos a prestigiarem as apresentações.

O WGID 2017 contará ainda com palestras e um painel de discussões de representantes do governo federal sobre as iniciativas e os desafios enfrentados em nível nacional para tratar das diversas questões relacionadas a gestão de identidades. Da parte do Tribunal Superior Eleitoral, teremos uma apresentação sobre o novo documento nacional de identidade, que foi regulamentado por lei recentemente e deverá ser o registro de identidade único do país. Técnicos do Ministério do Planejamento, Desenvolvimento e Gestão descreverão um projeto piloto desenvolvido recentemente em parceria com a iniciativa privada no sentido de avaliar o potencial de uso da tecnologia de blockchains em gestão de identidade e de cadastro de cidadãos. O Ministério da Saúde também estará presente e abordará os desafios que são enfrentados nas ações de registro, identificação e cadastro de cidadãos. Após as palestras, os técnicos formarão uma mesa de discussão sobre os principais pontos apresentados e responderão às perguntas da plateia.

O encontro de pesquisadores no WGID será aproveitado ainda para se divulgar as atividades do Comitê Técnico de Gestão de Identidades e do Laboratório de Gestão de Identidades, ambos da RNP, com o objetivo de oferecer novos serviços e fóruns de discussão para a comunidade de ensino e pesquisa.

Gostaríamos de registrar nossos agradecimentos aos membros do Comitê de Programa pelo trabalho de excelente qualidade realizado nas revisões dos artigos e a todos os autores que prestigiaram o WGID, submetendo seus trabalhos e divulgando suas pesquisas.

Agradecemos também todos que, de uma forma ou de outra, colaboraram na organização do WGID 2017 e, em especial, os Coordenadores Gerais do SBSeg 2017, Profs. Edna Dias Canedo e Rafael Timóteo de Sousa Junior, pelo enorme apoio dado à realização deste workshop.

Em nome do Comitê de Programa do WGID, cumprimentamos e damos as boas vindas a todos os participantes do workshop, na expectativa de que tirem o máximo proveito do evento.

Prof. Marco Aurélio Amaral Henriques (Unicamp)
Coordenador do WGID 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação do WFC

O Workshop de Forense Computacional (WFC), já na sua sexta edição, é um fórum para discussões e apresentações técnicas de trabalhos em torno do estado da arte de tecnologias relacionadas à forense computacional. De forma geral trata do estudo de evidências digitais que possam ser usadas para efeito investigativo e auditoria.

Gostaríamos de agradecer à comunidade de pesquisadores que submeteram trabalhos, e em particular aos membros do Comitê de Programa e revisores extras que se esforçaram para que em curto espaço de tempo conseguíssemos avaliações críticas e ponderadas em benefício do avanço nessa temática. A taxa de aceitação do WFC 2017 foi de 33,3%, onde 4 artigos foram selecionados para apresentação final, sendo que 83% dos artigos receberam no mínimo 3 avaliações.

Além da apresentação dos trabalhos selecionados teremos uma palestra convidada, a ser proferida pelo Chefe do Serviço de Perícias em Informática, do Instituto Nacional de Criminalística da Polícia Federal, perito Luciano Lima Kuppens, sobre Criptoanálise em Perícias Digitais.

Aguardamos a todos com entusiasmo em Brasília para as atividades do SBSeg e do WFC em 2017.

Díbio Leandro Borges (CIC – UnB)
Vinicius Ruela Pereira Borges (CIC – UnB)
Coordenadores do WFC 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação do WTE

A terceira edição do Workshop de Tecnologia Eleitoral (WTE), evento satélite do XIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2017), tem como objetivo principal fomentar a pesquisa em tecnologia eleitoral, na tentativa de reaproximar a academia das questões ligadas ao voto eletrônico. Mesmo o Brasil realizando eleições puramente eletrônicas desde 1996 (e integralmente desde 2000), a participação do meio acadêmico na iniciativa ainda tem sido pouco expressiva. Desta forma, fornecer um fórum para discussão técnica, apresentação de artigos e contato com trabalhos de excelência internacional pode contribuir para o treinamento de estudantes, pesquisadores e profissionais na área.

Para esta edição, decidiu-se por ampliar o evento para ocupar um dia completo, com programa formado pela apresentação de artigos científicos e palestra convidadas. Foram recebidas e revisadas 5 submissões em caráter anônimo, sendo 2 aceitas para publicação e outras 2 aceitas condicionalmente. As palestras convidadas são ministradas pelo Tribunal Superior Eleitoral e pelos pesquisadores Prof. Dr. Feng Hao (Universidade de Newcastle) e Matthew Bernhard (Universidade de Michigan) e tratam, respectivamente, da evolução do sistema de votação eletrônica brasileiro, descentralização de sistemas de votação verificáveis e um esforço de auditoria após as eleições americanas de 2016.

A coordenação do evento agradece em especial ao Comitê de Programa, composto pelos professores Mario Gazziro (UFABC), Ruy J. Guerra B. de Queiroz (UFPE) e Roberto Samarone Araújo (UFPA); que auxiliou na avaliação dos artigos submetidos. Espera-se que o workshop venha desempenhar papel importante do ponto de vista científico, na forma de apresentação de trabalhos acadêmicos; mas também contribua para um amadurecimento na discussão em torno de tecnologia eleitoral utilizada no país, estabelecendo bases mais sólidas para sua avaliação.

Diego F. Aranha (Unicamp)
Jeroen van de Graaf (UFMG)
Coordenadores do WTE 2017

© 2017 Sociedade Brasileira de Computação

Mensagem da Coordenação do Workshop sobre Regulação, Avaliação da Conformidade, Testes e Padrões de Segurança (WRAC+)

O Workshop sobre Regulação, Avaliação da Conformidade, Testes e Padrões de Segurança (WRAC+) chega à sua terceira edição mantendo a sua missão de proporcionar um fórum onde Governo, Laboratórios, Academia, Indústria e Sociedade possam discutir os potenciais e os limites das ferramentas da regulação e da avaliação da conformidade no campo da Segurança da Informação. Neste terceiro WRAC+, contaremos com três palestras técnicas e com um painel que proporcionará uma discussão ampla a respeito dos temas do Workshop.

Agradecemos aos autores que escolheram o WRAC+ como um veículo para a publicação de suas pesquisas, ao Comitê de Programa que realizou uma avaliação cuidadosa dos artigos submetidos, e ao Comitê Organizador do SBSeg, que sempre ofereceu apoio incondicional aos organizadores deste WRAC+.

Raphael Carlos Santos Machado (Inmetro)
Coordenador do WRAC+ 2017

© 2017 Sociedade Brasileira de Computação

Mensagem do Coordenador de Minicursos

O Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) é um evento científico promovido anualmente pela Sociedade Brasileira de Computação (SBC) e representa o principal fórum no país para a apresentação de pesquisas e atividades relevantes ligadas à segurança da informação e de sistemas. O evento promove a integração da comunidade brasileira de pesquisadores e profissionais atuantes nessa área. Entre as principais atividades técnicas do SBSeg encontram-se a trilha de Minicursos, que têm como objetivo a atualização em temas normalmente não cobertos nas grades curriculares e que despertem grande interesse entre acadêmicos e profissionais.

Nesta edição do SBSeg (2017), 20 propostas de minicursos foram submetidas, um número expressivo que demonstra a importância deste evento no panorama nacional de pesquisa. Destas, 4 foram selecionadas para publicação e apresentação, representando assim uma taxa de aceitação de 20%. O Comitê de Avaliação dos Minicursos foi composto por 14 renomados pesquisadores para a elaboração dos pareceres. Cada proposta recebeu ao menos 3 pareceres, gerando ao todo mais 60 revisões. Além disto, inúmeras mensagens foram trocadas entre os membros do comitê durante a fase de discussão. Cada minicurso selecionado corresponde a um capítulo deste livro e sua apresentação é parte da programação do SBSeg 2017.

O livro de Minicursos reúne 4 capítulos produzidos pelos autores das propostas de minicursos aceitas. O Capítulo 1, alinhado à tendência crescente e inevitável de uso de pequenos dispositivos computacionais, discute questões contemporâneas na área de segurança e privacidade no contexto da computação ubíqua, bem como aponta para problemas de pesquisa ainda não solucionados. O Capítulo 2, de maneira similar, apresenta as principais tecnologias de segurança baseada em hardware, abordando funcionalidades e usos da arquitetura Intel Software Guard Extentions (SGX). Os Capítulos 3 e 4 discutem a tecnologia blockchain, uma tecnologia baseada em difusão e que tem impulsionado novas pesquisas, desenvolvimentos e inovações na área de segurança computacional. O Capítulo 3, apresenta os fundamentos da tecnologia e os aspectos de programação que envolvem o desenvolvimento de aplicações e segurança de sistemas. O Capítulo 4, apresenta como a tecnologia blockchain pode ser usada para prover segurança e privacidade no contexto da Internet das Coisas (IoT). Enfim, o resultado é um livro com assuntos atuais e que permitem uma atualização em temas de interesse da academia e indústria.

Raul Ceretta Nunes (UFSM)
Coordenador de Minicursos do SBSeg 2017

© 2017 Sociedade Brasileira de Computação

Comitê de Programa do SBSeg 2017

Paulo Lício de Geus, Unicamp - Coordenador
Rossana Maria de Castro Andrade, UFC - Coordenadora

Alberto Schaeffer Filho, UFRGS
Aldri dos Santos, UFPR
Altair Olivo Santin, PUCPR
André Grégio, UFPR
André Santos, UECE
Carla Merkle Westphall, UFSC
Carlos Alberto da Silva, UFMS
Carlos Eduardo da Silva, UFRN
Carlos Maziero, UTFPR/UFPR
Célio Vinicius Neves de Albuquerque, UFF
Davidson Boccardo, Green Hat Labs
Denise Goya, UFABC
Diego Aranha, UNICAMP
Dorgival Guedes, UFMG
Edna Canedo, UnB
Eduardo Alchieri, UnB
Eduardo da Silva, IFSC
Eduardo Feitosa, UFAM
Eduardo Góes, DPF
Eduardo Souto, UFAM
Emerson Ribeiro de Mello, IFSC
Hao Chi Wong, Intel
Jean Martina, UFSC
Jeroen Graaf, UFMG
Joaquim Celestino Júnior, UECE
Karina Mochetti, UFF
Leonardo Fagundes, UVRS
Leonardo Oliveira, UFMG
Lisandro Zambenedetti Granville, UFRGS
Luciano Paschoal Gaspary, UFRGS
Luiz Carlos Albini, UFPR
Luiz Fernando Rust da Costa Carmo, INMETRO
Marco Aurelio Amaral Henriques, UNICAMP
Marcos Simplicio Jr, Poli-USP
Marinho Barcellos, UFRGS
Marjory Abreu, UFRN
Mário Sérgio Alvim, UFMG
Michele Nogueira, UFPR
Michelle Wangham, UNIVALI
Miguel Castro, UFC
Miguel Elias Mitre Campista, UFRJ
Otto Carlos Muniz Bandeira Duarte, UFRJ
Paulo André da Silva Gonçalves, UFPE
Pedro Braconnot Velloso, UFRJ
Priscila Solis, UnB
Rafael Timóteo de Sousa Junior, UnB
Raphael Machado, INMETRO
Raul Ceretta Nunes, UFSM
Raul Weber, UFRGS
Ricardo Custódio, UFSC
Ricardo Dahab, UNICAMP
Roberto Gallo, KRYPTUS Segurança da Informação
Routo Terada, IME-USP
Sergio de Oliveira, UFSJ

© 2017 Sociedade Brasileira de Computação

Comitê de Programa do SBSeg 2017

Paulo Lício de Geus, Unicamp - Coordenador
Rossana Maria de Castro Andrade, UFC - Coordenadora

Revisores

Adi Marcondes, UFPR
Alan de Sá, Marinha do Brasil
Álvaro Robles, UFRJ
Amir Ahmad, USP
Anderson da Silva, UFRGS
Antonio Franco, UFMG
Arthur Callado, UFC
Arthur Jacobs, UFRGS
Artur Luis, UFMG
Bruno Medeiros de Barros, USP
Carlos André Batista de Carvalho, UFPI
Carlos Henrique O. M. André, UFRJ
Cezar Murilo de Toledo, USP
Charles Prado, Inmetro
Cleverton Vicentini, PUC-PR
Daniel Marcon, UFRGS
Danilo Vasconcelos, IFCE
Dianne Medeiros, UFRJ
Eder John Scheid, UFRGS
Eduardo Cominetti, USP
Eduardo Viegas, PUC-PR
Fabrício Ceschin, UFPR
Fernanda Gomes, UFSC
Fernando Garcia, IFCE
Fernando Quintão Pereira, UFMG
Fernando Augusto Teixeira, UFSJ
Flavia Agostini, InMetro
Frederico Sampaio, UFMG
Gerson Camillo, UFSC
Gilbert Martins, UFAM
Glederson Santos, IFSul
Hugo Sadok, UFRJ
Igor Alvarenga, UFRJ
Igor Jochem Sanz, UFRJ
Jackson Mallmann, PUCPR
Jéssica Carneiro, UFMG
Jhonatan Geremias, PUCPR
Jomaro Rodrigues, UFPR
Jorge Werner, UFSC-LRG
Leopoldo Alexandre Freitas Mauricio, UFRJ
Lucas Freire, UFRGS
Lucas Lins, Grupo Clavis Segurança da Informação
Lucila Bento, Grupo Clavis Segurança da Informação
Maicon Stihler, CEFET-MG
Marcelo Caggiani Luizelli, UNIPAMPA
Marcial Fernandez, UECE
Marcos Araujo, UFRJ
Marcos Silva, USP
Marcus Botacin, Unicamp
Maria Luiza Santos, UFMG
Martin Andreoni Lopez, UFRJ
Márcio Carvalho, UFRGS
Miguel Neves, UFRGS
Muriel Franco, UFRGS
Nicolas Kagami, UFRGS
Rafael Brinhosa, UFSC
Rafael Lopes Gomes, UECE
Rafael Obelheiro, UDESC
Ricardo Pfitscher, UFRGS
Ricardo Luis dos Santos, UFRGS
Robson de Oliveira Albuquerque, UnB
Rone Silva, UFSJ
Rosembergue Souza, UFRJ
Sérgio de Medeiros Câmara, UFRJ
Sergio Oliveira, UFSJ
Silvio Costa Sampaio, UFRN
Taciane Martimiano, UFSC
Valderi Leithardt, UNIVALI
Vilmar Abreu, PUCPR
Wilson Melo Jr, InMetro
Yan Justino, UFRN

© 2017 Sociedade Brasileira de Computação

Comitê de Organização WFC

Coordenadores

Díbio Leandro Borges, (CIC – UnB)
Vinicius Ruela Pereira Borges, (CIC – UnB)

Membros do comitê de programa

Altair Santin, Pontifícia Universidade Católica do Paraná (PUCPR)
Dino Amaral, Banco do Brasil (BB)
Flávio de Deus, Universidade de Brasília (UnB)
Hugo Vieira Neto, Universidade Tecnológica Federal do Paraná (UTFPR)
Iwens Sene Jr, Universidade Federal de Goiás (UFG)
João Gondim, Universidade de Brasília (UnB)
João Paulo Sousa, Polícia Civil do Distrito Federal (PCDF)
Leandro Oliveira, Universidade Federal de Goiás (UFG)
Rafael Obelheiro, Universidade do Estado de Santa Catarina (UDESC)
Ricardo Dahab, Universidade Estadual de Campinas (Unicamp)
William Schwartz, Universidade Federal de Minas Gerais (UFMG)

Comitê de Organização WGID

Coordenador

Marco Aurélio Amaral Henriques, Unicamp

Membros do comitê de programa

Altair Santin, PUC-PR
Antonio Tadeu Gomes, LNCC
Carlos da Silva, UFRN
Carlos Ferraz, UFPE
Debora Muchaluat-Saade, UFF
Edelberto Franco, UFJF
Emerson Ribeiro de Mello, IFSC
Fábio Borges, LNCC
Flavio de Deus, UnB
Gustavo Motta, UFPB
Jean Martina, UFSC
Leonardo Oliveira, UFMG
Marco Aurélio Amaral Henriques, Unicamp
Michelle Wangham, UNIVALI
Natália Fernandes, UFF
Rafael Timóteo de Sousa Junior, UnB
Ricardo Custódio, UFSC
Ricardo Dahab, Unicamp
Roberto Samarone Araujo, UFPA

© 2017 Sociedade Brasileira de Computação

Comitê de Organização WRAC+

Coordenador

Raphael Carlos Santos Machado, Inmetro

Membros do comitê de programa

Alan de Sá, Marinha do Brasil
Alex Garcia, IME – Instituto Militar de Engenharia
Altair Santin, PUC-PR – Pontifícia Universidade Católica do Paraná
Claudio de Farias, UFRJ – Universidade Federal do Rio de Janeiro
Daniel Menasche, UFRJ – Universidade Federal do Rio de Janeiro
Davidson Boccardo, Green Hat Segurança da Informação
Fábio Borges, LNCC – Laboratório Nacional de Computação Científica
Luiz Fernando Rust da Costa Carmo, Inmetro
Marcio Miranda, UFRRJ – Universidade Federal Rural do Rio de Janeiro
Mário Benevides, UFRJ – Universidade Federal do Rio de Janeiro
Paulo Bueno, CTI Renato Archer
Rodolfo Souza, Inmetro
Sanderson Barbalho, UnB – Universidade de Brasília
William de Souza, CASNAV – Centro de Análises de Sistemas Navais

Comitê de Organização WTE

Diego de Freitas Aranha, Unicamp
Jeroen van de Graaf, UFMG
Mario Gazziro, UFABC
Roberto Samarone Araújo, UFPA
Ruy J. Guerra B. de Queiroz, UFPE

Comitê de Organização WTICG

Coordenador

Márjory da Costa Abreu, UFRN

Membros do comitê de programa

Altair Santin, PUC-PR
Anderson Nascimento, UnB
Arlindo L. Marcon Jr., IFPR
Carla Merkle Westphall, UFSC
Carlos Maziero, UFPR
Carlos Eduardo da Silva, UFRN
Daniel G. Costa, UEFS
Diego Aranha, Unicamp
Edna Dias Canedo, FGA-UnB
Eduardo Feitosa, UFAM
Eduardo Souto, UFAM
Emerson Ribeiro de Mello, IFSC
João Gondim, UnB
Jose Antonio Xexeo, IME
Leonardo B. Oliveira, UFMG
Lisandro Zambenedetti Granville, UFRGS
Luiz Fernando Rust da Costa Carmo, Inmetro
Michelle Wangham, Univali
Paulo André da Silva Gonçalves, UFPE
Rafael Obelheiro, UDESC
Ricardo Dahab, Unicamp
Rossana Andrade, UFC
William de Souza, University of London

© 2017 Sociedade Brasileira de Computação

Comitê de Organização Local

Aletéia Patrícia Favacho de Araújo, CIC – UnB
Alex Sandro Roschildt Pinto, Campus Blumenau – UFSC
César Augusto Borges de Andrade, CDS – Exército Brasileiro
Diego Martins de Oliveira, IFB/UnB
Edison Pignaton de Freitas, Inf – UFRGS
Eduardo Adilio Pelinson Alchieri, CIC – UnB
Fabiano Cavalcanti Fernandes, IFB
Fábio Lúcio Lopes de Mendonça, ENE – UnB
Georges Daniel Amvame Nze, ENE – UnB
Giovanni Almeida Santos, FGA – UnB
João Paulo Carvalho Lustosa da Costa, ENE – UnB
Priscila América Solís Mendez Barreto, CIC – UnB
Robson de Oliveira Albuquerque, ENE – UnB
Sérgio Andrade de Freitas, FGA – UnB
Ugo Silva Dias, ENE – UnB
William Ferreira Giozza, ENE – UnB

© 2017 Sociedade Brasileira de Computação

Sumário

Mensagens das Coordenações do SBSeg 2017 vi

Comitê de Programa do SBSeg 2017 xviii

Artigos Completos 1

Sessão técnica 1 (ST1): Criptografia e Criptoanálise 2

1. *Pipeline Oriented Implementation of NORX for ARM Processors*
   Luan Cardoso dos Santos, Julio López 2
2. *A Zero-Knowledge Proof for the Hidden Subset Sum Problem*
   Charles F. de Barros 16
3. *A Longitudinal and Retrospective Study on How Developers Misuse Cryptography in Online Communities*
   Alexandre Braga, Ricardo Dahab 28

Sessão técnica 2 (ST2): Aplicações 42

1. *Processador de Uso Específico para o Algoritmo SHA-1*
   Carlos E. B. Santos Junior, Marcelo A. C. Fernandes 42
2. *Eliminação Segura de Arquivos em Memória Não-Volátil*
   Julia S. Weber, Avelino F. Zorzo 56
3. *Um método de identificação de navegadores Web baseado na Web Audio API*
   Jordan S. Queiroz, Eduardo L. Feitosa 70

Sessão técnica 3 (ST3): Internet das Coisas & Computação em Nuvem 84

1. *ConCReCT: Um Mecanismo para Mitigação dos Efeitos de Ataques de Negação de Serviço em Internet das Coisas Industriais*
   Vladimir Borgiani, Patrick Moratori, Juliano Kazienko, Emilio Tubino 84
2. *Uma Avaliação de Desempenho de Segurança Definida por Software através de Cadeias de Funções de Rede*
   Igor J. Sanz, Igor D. Alvarenga, Martin Andreoni, Leopoldo A. F. Mauricio, Diogo M. F. Mattos, Marcelo G. Rubinstein e Otto Carlos M. B. Duarte 98
3. *Análise de Séries Temporais Multivariadas para Detecção de Intrusão em Nuvens Computacionais*
   Mayk Fernando Choji, Paulo Lício de Geus 112

Sessão técnica 4 (ST4): Criptografia e Criptoanálise & Malware & Aplicações 126

1. *Esquema de Acordo de Chaves de Conferência Baseado em um Problema de Funções Quadráticas de Duas Variáveis*
   Luis Antonio B. Kowada1, Raphael C. S. Machado 126
2. *Aplicação Dinâmica de Políticas Estáticas de Fluxo*
   Carina Capelão de Oliveira, Glauco Gonçalves Cardoso, Fernando Magno Quintão Pereira 140
3. *An operational costs analysis of similarity digest search strategies using approximate matching tools*
   Vitor Hugo Galhardo Moia, Marco Aurélio Amaral Henriques 154

© 2017 Sociedade Brasileira de Computação

Sessão técnica 5 (ST5): Intrusão - detecção, prevenção e resposta.....168
1. *Flow-Based Intrusion Detection for SCADA networks using Supervised Learning*
   Gabriel Vasquez, Rodrigo S. Miani, Bruno B. Zarpelão 168
2. *Análise de Alertas de Sistemas de Detecção de Intrusão: Uso de Aprendizado Supervisionado na Redução de Alertas Falsos Positivos*
   Eduardo Alves Moraes, Carlos Alexandre Carvalho Tojeir, Rodrigo Sanches Miani, Bruno Bogaz Zarpelão..... 182
3. *Uma arquitetura para detecção de botnets baseada na análise do tráfego DNS descartado*
   Luiz Gonzaga Mota Barbosa, Joaquim Celestino Júnior, André Luiz Moura dos Santos..... 196

Sessão técnica 6 (ST6): Intrusão - detecção, prevenção, e resposta.....210
1. *Detecção de Intrusão Através de Aprendizagem de Fluxo no Ambiente do Adversário*
   Eduardo K. Viegas, Altair O. Santin, Vilmar Abreu, Luiz E. S. Oliveira..... 210
2. *Atraindo, conhecendo e repudiando atacantes VoIP*
   Daniel Bauermann, Giovani Luís Emmert 224
3. *Santos': Algoritmo para Detecção de Ataques do Tipo Stepping-stones*
   Amanda S. Santos, Marcus A.R Tenorio, Adriano A. Santos, Andrey Brito 236

Sessão técnica 7 (ST7): Malware.....250
1. *Analysis, Anti-Analysis, Anti-Anti-Analysis: An Overview of the EvasiveMalware Scenario*
   Marcus Botacin, Vitor Falcão da Rocha, Paulo Lício de Geus, André Grégio..... 250
2. *Detecção de malware metamórfico baseada na indexação de grafos de dependência de dados*
   Luis Rojas Aguilera, Eduardo Souto, Gilbert Breves Martins 264
3. *Using Huffman Trees in Features Selection to Enhance Performance in Spam Detection*
   Cleber K. Olivo, Altair O. Santin, Luiz E. S. Oliveira 278

Sessão técnica 8 (ST8): Redes 292
1. *Experiências com um Honeypot DNS: Caracterização e Evolução do Tráfego Malicioso*
   Tiago Heinrich, Felipe de Souza Longo, Rafael R. Obelheiro 292
2. *Combatendo vulnerabilidades em programas P4 com verificação baseada em asserções*
   Lucas Freire, Miguel Neves, Alberto Schaeffer-Filho, Marinho Barcellos..... 306
3. *Cache nFace: uma Contramedida Simples para o Ataque em Conluio Produtor-Consumidor em Redes Centradas em Conteúdo*
   André Nasserala, Igor Monteiro Moraes..... 320

Sessão técnica 9 (ST9): Controle de Acesso, Autenticação, Biometria, Confiança e Gerenciamento de Identidade
1. *Privacy-aware web authentication protocol with recovery and revocation*
   Lucas Boppre Niehues, Ricardo Custódio 334
2. *Análise de Vulnerabilidade de Esquemas de Segredo Compartilhado Considerando um novo Modelo de Ameaça*
   Rick Lopes de Souza, Martin Vigil, Ricardo Felipe Custódio 348
3. *Authentication of Identity Documents Using DNSSEC, Digital Signatures and QR Codes*
   Luiz Fernando Ribeiro Amaral, Jorge Guilherme Silva dos Santos, Mateus Almeida Rocha, Joseph Gersch, Georges Daniel Amvame-Nze, Robson de Oliveira Albuquerque, Rafael Timóteo de Sousa Júnior 362

© 2017 Sociedade Brasileira de Computação

4. Uma Avaliação das Prevenções de Phishing em Navegadores Web
Carlo M. R. da Silva, Eduardo L. Feitosa, Vinícius C. Garcia 374

Sessão técnica 10 (ST10): Controle de Acesso, Autenticação e Gerenciamento de Identidade & Intrusão - detecção, prevenção e resposta 388

1. *Marca d'água estruturada*
Lucila Maria de Souza Bento , Davidson Rodrigo Boccardo, Raphael Carlos Santos Machado, Vinícius Gusmão Pereira de Sá e Jayme Luiz Szwarcfiter..... 388
2. *Um Sistema Adaptativo de Detecção e Reação a Ameaças*
Antonio Gonzalez Pastana Lobato, Martin Andreoni Lopez, Gabriel Antonio F. Rebello, e Otto Carlos Muniz Bandeira Duarte..... 400
3. *Uso de Controle Chaveado para Mitigação de Ataque Ativo de Identificação de Sistemas com Malha Fechada*
Alan Oliveira de Sá, Luiz F. Rust da Costa Carmo, Raphael C. S. Machado 414

Sessão técnica 11 (ST11): Controle de Acesso, Autenticação, Biometria, Confiança e Gerenciamento de Identidade 428

1. *Eigenface vs Random Forest: Um Estudo Comparativo em Reconhecimento Facial*
Jardel Ribeiro de Lima e Rosalvo Fereira Oliveira Neto 428
2. *The performance of error correcting codes in discrete and continuous fuzzy pairing protocols*
Henrique de Rocha Deus, Vladimir Portela Parente, Jeroen van de Graaf..... 442
3. *Detecção de Canvas Fingerprinting em PáginasWeb baseada no Modelo Vetorial*
Pablo A. da P. Elleres, Adria M. de Oliveira, Eduardo L. Feitosa..... 456

Resumos Estendidos - Seção Técnica 1 - Malware 470

1. *Static Analysis on Disassembled Files: A Deep Learning Approach to Malware Classification*
Dhiego Ramos Pinto, Julio Cesar Duarte 471
2. *Mitigando a Evasão de Instrumentadores Binários Dinâmicos*
Ailton Santos Filho, Arthur Binda Alves, Isaque Vieira, Eduardo L. Feitosa..... 479
3. *Detecção de Desfiguração de Sites por Similaridade em Imagens*
Henry Mendes de Jesus, José Eduardo Malta de Sá Brandão..... 487
4. *Um Estudo Prático sobre o Potencial do Ataque Slowloris a partir de Dispositivos Móveis*
Lucas O. C. Aversari, Raoni Kulesza, Josilene A. Moreira..... 494
5. *Um Método para Detecção de Bots Sociais Baseado em Redes Neurais Convolucionais Aplicadas em Mensagens Textuais*
Paulo A. Braz, Ronaldo R. Goldschmidt..... 501
6. *A era dos Crypto Ransomwares: um estudo de caso sobre o WannaCry*
Guilherme Baesso Moreira, Vanusa Menditi Calegario, M.Sc., Julio Cesar Duarte, D.Sc., Anderson F. Pereira dos Santos, D.Sc. 509

Resumos Estendidos - Seção Técnica 2 - Intrusão - detecção, prevenção e resposta & Criptografia e Criptoanálise & Aplicações & Hardware.....517

1. *Keyblock: a software architecture to prevent keystroke injection attacks*
C. David B. Borges, J. Rafael B. de Araujo, Robson L. de Couto, A. Márcio A. Almeida 518
2. *A publicly verifiable protocol for random number generation*
João Penna, Jeroen van de Graaf..... 525

© 2017 Sociedade Brasileira de Computação

3. *Robustness Testing of CoAP Server-side Implementations through Black-box Fuzzing Techniques*
Bruno da S. Melo, Paulo Lício de Geus 533
4. *Detecção e autorreparo de anomalias em redes definidas por software*
Fernando Luiz Moro, Alexandre Amaral, Ana Paula Amaral, Rodrigo Nogueira.....541
5. *Análise de segurança da distribuição de raízes na ICP-Brasil*
Bruno C. Dias Ribeiro, Edson Floriano S. Junior, Diego F. Aranha 549
6. *Evaluating the Randomness of the RNG in a Commercial Smart Card*
Wellinton Costa Ribeiro, Marcus Tadeu Pinheiro Silva..... 557

Artigos dos Workshops do SBSeg 2017 564

WFC – VI Workshop de Forense Computacional 565
1. *Fast Similarity Digest Search: A new strategy for performing queries efficiently with approximate matching*
Vitor Hugo Galhardo Moia, Marco Aurélio Amaral Henriques..... 566
2. *Avaliação da Eficácia de Classificadores de Malware ao Longo do Tempo*
Fabrício Ceschin, André Grégio, David Menotti, Marcos Castilho 576
3. *Dizang: Uma solução para coleta de evidências forenses de ataques de injeção na nuvem*
Hamilton J. S. Fonte II, Marcos A. Simplicio Jr. 586
4. *ForensicWork: Desenvolvimento de um Framework para Perícia Forense em Computação na Nuvem*
Cátia Oliveira de Jesus, Mailson Sousa Couto 596

WGID - VII Workshop de Gestão de Identidades Digitais - Artigos completos 602
1. *Supporting the Self-adaptation of Authorization Infrastructures*
Welkson Renny de Medeiros, José Diego Saraiva, Carlos Eduardo da Silva..... 603
2. *Análise de sistemas de gestão de identidade baseados em blockchain*
Antônio Unias de Lucena, Marco Aurélio Amaral Henriques..... 615

WGID - VII Workshop de Gestão de Identidades Digitais - Resumos Estendidos ... 627
1. *Case Study: Augmenting Security of Cloud-Based Digital Signatures*
Roberto Gallo, Conrado P. L. Gouvêa 628
2. *Eduroam Testbed: Um Ambiente Real para Experimentação em AAA*
Edelberto F. Silva, Luciano F. da Rocha, Javier R. Q. Ancieta, André Forigato, Anderson Almeida, Jonathan Welley 632
3. *Avaliação do Custo de Autenticação em VANETs*
Ivo Carlson R. Barros, Artur Souza, Guilherme Maia, Leonardo B. Oliveira..... 636

WRAC+ – III Workshop sobre Regulação, Avaliação da Conformidade, Testes e Padrões de Segurança 640
1. *Caminhos para a Segurança: Normalização ou Adequação de Padrões*
Carlos Roberto Viana, Raphael Machado..... 641
2. *Testes de sobrecarga: uma avaliação sobre requisitos de Disponibilidade e Desempenho*
Carlos A. M. S. Teles, Raphael Carlos Santos Machado..... 647

WTE – III Workshop de Tecnologia Eleitoral 653
1. *Do back-end para o front-end: Uma Avaliação de Segurança de Aplicações Móveis da Justiça Eleitoral*
Wellington Silva de Souza, Daniel G. de Azevedo, Samuel Xavier de Souza 654

 © 2017 Sociedade Brasileira de Computação

2. *Registro impresso do voto, autenticado e com garantia de anonimato*
Rodrigo Carneiro Munhoz Coimbra, José Roberto Menezes Monteiro e Gladiston da Silva Costa..... 666
3. *Urna eletrônica de terceira geração: Um protótipo para eleições auditáveis*
Fernando Teodoro de Lima, Mario A. Gazziro, Antonio de Abreu Batista Jr, Paulo Matias, João V. C. Costa 677
4. *Variations on Wombat: end-to-end voting on a shoestring*
Camilo Gómez, João Moreno, Jeroen van de Graaf, Alejandro Hevia 686

WTICG – XI Workshop de Trabalhos de Iniciação Científica e de Graduação 696

Sessão técnica 1 (ST1): Criptografia e criptoanálise.....697
1. *A integração do Criptossistema El Gamal Limiar ao Sistema de Votação via Internet CIVIS*
André A. Silva Neto, Roberto Araújo..... 697
2. *Avaliação de assinaturas baseadas em hash para a Internet das Coisas*
Jéssica C. Carneiro, Leonardo B. Oliveira..... 707
3. *Benchmarking tools for verification of constant-time execution*
Arthur Costa Lopes, Diego de Freitas Aranha..... 716

Sessão técnica 2 (ST2): Forensics 727
4. *Estudo do mecanismo de descoberta de vizinhançaa do IPv6 para a realização do ataque man in the middle*
Matheus Dias Queiroz, André Leon S. Gradvoh 727
5. *Investigando as Percepções de Privacidade do Usuário em Redes Sociais Online*
Ingrid L. dos Santos, Karla S. Pereira, Eduardo L. Feitosa..... 737
6. *Uma Análise Comparativa de Ferramentas de Esteganálise em Estego-Imagens com Texto Embutido*
Lucas C. Possatti, Leandro C. Resendo, Gilberto Sudré Filho, Jefferson O. Andrade, Karin S. Komati..... 747

Sessão técnica 3 (ST3): Autenticação / Autorização 757
7. *Uma Proposta de Arquitetura para Autorização Federada com Internet das Coisas*
Carlos Eduardo da Silva, Gabriela Cavalcante da Silva..... 757
8. *Estudo Sobre o Impacto das Vulnerabilidades de um Cartão Inteligente Sem Contato*
Luiz Eduardo M. Matheus, Eduardo P. Julio, Alex B. Vieira, Edelberto Franco.....767
9. *Desenvolvendo Aplicações de Cidades Inteligentes com Autenticação e Autorização baseada na Plataforma FIWARE*
Gabriela Cavalcante da Silva, Carlos Eduardo da Silva..... 777

SF - Salão de Ferramentas 787
1. *CATRACA: uma Ferramenta para Classificação e Análise de Tráfego Escalável Baseada em Processamento por Fluxo*
Martin Andreoni Lopez, Igor J. Sanz, Diogo M. F. Mattos, Otto Carlos M. B. Duarte e Guy Pujolle..... 788
2. *Provisionamento automatizado de servidores para competições de segurança da informação*
Lucas Magalhães, Antônio Carlos F. Petri, Gabriel de S. Alves, Cesar Augusto C. Marcondes, Paulo Matias..... 796

© 2017 Sociedade Brasileira de Computação

Artigos Completos

© 2017 Sociedade Brasileira de Computação

Pipeline Oriented Implementation of NORX for ARM Processors

Luan Cardoso dos Santos[1], **Julio López**[1]

[1]Laboratory of Security and Cryptography - LASCA
Institute of Computation - UNICAMP
Av. Albert Einstein, 1251, Room 84
Cidade Universitária Zeferino Vaz
13083-852 Campinas SP Brazil

luan@lasca.ic.unicamp.br, jlopez@ic.unicamp.br

Abstract. *NORX is a family of authenticated encryption algorithms that advanced to the third-round of the ongoing CAESAR competition for authenticated encryption schemes. In this work, we investigate the use of pipeline optimizations on ARM platforms to accelerate the execution of NORX. We also provide benchmarks of our implementation using NEON instructions. The results of our implementation show a speed improvement up to 48% compared to the state-of-art implementation on Cortex-A ARMv8 and ARMv7 processors.*

Resumo. *NORX é uma família de algoritmos de cifração autenticada que participa da terceira fase do CAESAR, competição para esquemas de cifração autenticada. Nesse trabalho, investigamos o uso de optimizações de* pipeline *em plataformas ARM de forma a acelerar a execução do NORX. Também mostramos tempos da nossa implementação usando instruções NEON. Nossos resultados mostram melhoria de até 48% na velocidade de execução comparado com implementações estado-da-arte em processadores Cortex A ARMv8 e em processadores ARMv7.*

1. Introduction

Authenticated Encryption algorithms (AE) are symmetric-key cryptographic schemes where the main objective is to provide simultaneously confidentiality, integrity, and authentication. In an intuitive form, confidentiality means that an adversary with access to the ciphertext and nonce cannot recover any information of the plaintext beyond its length, and the ciphertext itself is indistinguishable from random bits. Similarly, authenticity guarantees that a ciphertext cannot be manipulated to generate a valid authentication for any given message.

NORX [1] is an authenticated encryption scheme currently participating in the CAESAR [2] competition, which has the objective of choosing an authenticated encryption algorithms that offer advantages over the AES-GCM algorithm [3], as candidates for a future standard in authenticated encryption.

We show software optimization techniques for the NORX family of algorithms, targeting ARM processors. We choose ARM processors due to their widespread usage in consumer electronics, such as IOT devices, smartphones, embedded devices, and gadgets.

© 2017 Sociedade Brasileira de Computação

In this section, we will provide some background on the concepts used throughout the paper regarding AE, cryptographic competitions, and sponge functions. In Section 2, we will give a brief description of the NORX algorithm, and in Section 3 we will show the main characteristics of the target architecture. Section 4 will describe our implementation techniques and in Section 5, we will present benchmarks and discuss the results.A final conclusion is given in Section 6.

1.1. AEAD Algorithms

An authenticated encryption scheme is an algorithm that uses a secret key and a public nonce to process a plaintext and generate a ciphertext and an authentication tag. Furthermore, an AE scheme can also receive extra data that is authenticated together with the plaintext. In that mode of operation, this scheme is called Authenticated Encryption with Additional Data (AEAD). Such a scheme is useful, for example, to encrypt the body of a message, while keeping the receiving address in plain form, and authenticating the whole. This way, the recipient of a message can guarantee that public data was not modified by a third party. A basic block diagram of an authenticated encryption algorithm is shown in Figure 1.

Formally an AEAD scheme is defined by the tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and the associated sets $\text{Nonce} = \{0,1\}^n$, $\text{Header} \subset \{0,1\}^*$ and $\text{Message} \subseteq \{0,1\}^*$. The Message set must satisfy the membership test $M \in \text{Message} \Rightarrow M' \in \text{Message}$ for any M' with the same length of M. The keyspace $\mathcal{K}$ is a non-empty finite set of strings. The encryption algorithm $\mathcal{E}$ is a deterministic algorithm that receives as input strings $K \in \mathcal{K}$, $N \in \text{Nonce}$, $H \in \text{Header}$ and $M \in \text{Message}$. The encryption algorithm returns a string $\mathcal{C} = \mathcal{E}_K^{N,H}(M) = \mathcal{E}_K(N, H, M)$. The decryption algorithm $\mathcal{D}$ is a deterministic algorithm that receives as input the strings $K \in \mathcal{K}$, $N \in \text{Nonce}$, $H \in \text{Header}$ and $\mathcal{C} \in \{0,1\}^*$ and returns $\mathcal{D}_K^{N,H}(\mathcal{C}) = \mathcal{D}_K(N, H, C)$, that is either a string from the set of possible messages, or a symbol $\perp$ meaning that the set of ciphertext, nounce and key is invalid. Beyond that, it is required that $\mathcal{D}_K^N(\mathcal{E}_K^N(M)) = M$ for all $K \in \mathcal{K}$, $N \in \text{Nounce}$ and $M \in \text{Message}$, and that $|\mathcal{E}_K^{N,H}| = l(|M|)$ for some linear-time length function l [4].

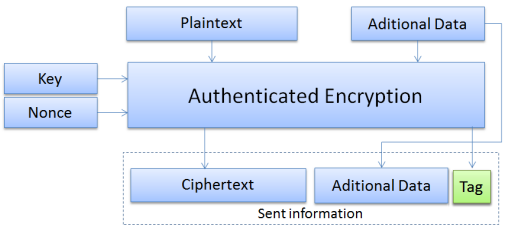


Figure 1. Basic block design of an AEAD, where ciphertext and authentication tag are produced by processing plaintext, additional data, key, and nonce.

1.2. The CAESAR competition

The algorithm considered in this paper is a competitor of the third round of CAESAR –Competition for Authenticated Encryption: Security, Applicability, and Robustness. Following the footsteps of previous competitions, such as the AES, eSTREAM, and SHA, CAESAR aims to select a portfolio of authenticated ciphers that offer advantages over

 © 2017 Sociedade Brasileira de Computação

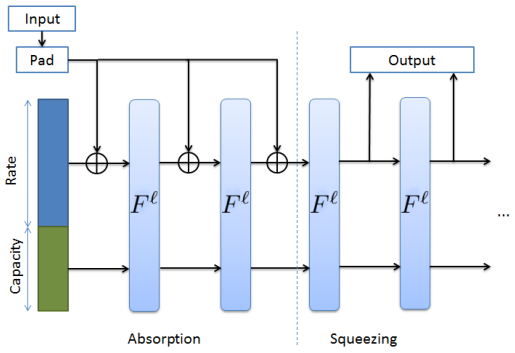


Figure 2. The basic design of a sponge function, showing the absorption and squeeze processes [7].

NIST's AES-GCM and that are suitable for widespread adoption. The AES competition is regarded as one of the responsible for promoting an improvement in the scientific knowledge about block ciphers. Similarly, eSTREAM [5] and SHA-3 [6] promoted research in the areas of stream ciphers and hash functions, and it is expected that the CAESAR competition brings the same impact in the research area of authenticated ciphers [2].

1.3. Sponge function

A cryptographic sponge function, introduced as a primitive for authenticated encryption in [7] and as a general cryptographic function in [8][9], is an algorithm with a finite internal state that receives as input a string of any length and produces as output a string of any desired length. Sponge functions can be used to create various cryptographic primitives, such as hash functions, MACs, stream ciphers, pseudorandom number generators and authenticated encryption schemes. A sponge function can be imagined as a real-world sponge, where data is absorbed and then squeezed from it.

A sponge is based on three main components: A state S of b bits, subdivided into *rate* and *capacity* sections of respectively r and c bits; a round permutation function F^l of b bits with a round number l defined in terms of a permutation F of b bits as the l-fold iteration $F^l(S) = F(F(...F(S)))$ which is used to transform the state in each round; and a padding rule P for the input. A sponge works by initializing the state value and "absorbing" r bits from the padded input and transforming the state with $F^l(S)$. After that, the sponge is ready to be "squeezed", removing up to r bits before needing to evaluate $F^l(S)$ again. Figure 2 illustrates the operation of a sponge [7]. An example of a practical use of sponge functions in cryptographic primitives is the SHA-3[6] hash algorithm, that uses a 1600-bit sponge.

2. The NORX AEAD family of algorithms

NORX is an AEAD scheme created by Jean-Philippe Aumasson, Philipp Jovanovic and Samuel Neves [10], supporting associated data in the form of both headers and trailers. NORX also supports arbitrary parallelism and is optimized for efficient hardware and software implementations, with a SIMD-friendly construction, no secret array indexing, and only bitwise operations. ARX primitives are thoroughly used, without modular additions, and is based on the monkey-duplex construction. NORX's core permutation

© 2017 Sociedade Brasileira de Computação

Table 1. The five instances of NORX

Instance name	w	l	p	t	k	n
NORX64-4-1	64	4	1	256	256	128
NORX32-4-1	32	4	1	128	128	64
NORX64-6-1	64	6	1	256	256	128
NORX32-6-1	32	6	1	128	128	64
NORX64-4-4	64	4	4	256	256	128

function is based on ChaCha's permutation [11], with the integer addition $(a + b)$ replaced by the approximation[1] $a \oplus b \oplus (a \wedge b) \ll 1$, which in turn –according to the design team of NORX– simplifies cryptanalysis and improves hardware efficiency [10].

The NORX family of algorithms is parametrized by the word size in bits w; a round number ℓ with $1 \leq \ell \leq 63$; a parallelism degree p with $0 \leq p \leq 255$ (where $p = 0$ defines arbitrary parallelism) and a tag t. Regarding the key length, NORX32 uses a 128-bit key, NORX64 a 256-bit key, while NORX16 and NORX08 use a 96-bit and an 80-bit key respectively. The 32 and 64-bit versions of NORX also use an $n = 2w$ bits nonce; the 8-bit and 16-bit variants have a nonce of $n = 32$ bits. On the CAESAR submission, Aumasson et al. [10] propose five instances of NORX for different uses cases. They are listed in Table 1, from the highest recommendation at the top to the lowest. The naming convention for a specific instance of the algorithm is NORXw-l-p-t, with w, l, p and t being the instance parameters. When the tag length is the default $t = 4w$, then the notation is shortened as NORXw-l-p.

NORX parametrized with $w = 32$ bits is adequate for lightweight applications and resource-constrained environments, requiring small hardware area and small ROM size for software implementations. On the other hand, the instances with $w = 64$ bits are adequate for high-performance and high-security applications, being efficient in both 64-bit and 32-bit CPUs[2]. Requirements for ASIC implementations are about 64 kGE, and at most 64 bytes of ROM for the initialization constants. It is also possible to implement NORX using only one byte plus the sponge size of data in RAM [10].

In the following sections, we will show in more details the specification of NORX, as given in [10].

2.1. NORX's mode of operation

NORX follows a duplexed sponge layout, as shown in Figure 3. NORX's construction allows parallel processing of the payload, defined by p. For serial processing, with $p = 1$, the layout of NORX is that of a standard duplexed sponge. For a value $p > 1$, the number of parallel processing lanes is given by the value of p; for example, Figure 3 illustrates the case of $p = 2$. For $p = 0$, the number of processing lanes is bounded by the size of the payload itself, making the layout of NORX similar to that of the PPAE construction [12].

[1] This approximation is derived from the identity $a + b = (a \oplus b) + (a \wedge b) \ll 1$.

[2] A draft of these use-cases can be found in the CAESAR mailing list at the address `https://groups.google.com/forum/#!topic/crypto-competitions/DLv193SPSDc`.

 © 2017 Sociedade Brasileira de Computação

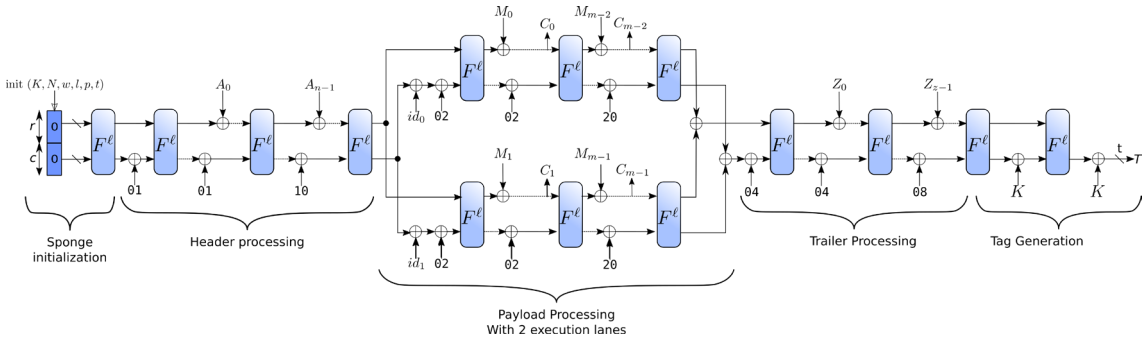


Figure 3. The layout of NORX with parallelism degree $p = 2$. Notice that the sponge is divided into multiple execution lanes in the payload processing step. Those lanes can be computed in parallel, as there is no data dependency amongst them. When $p = 1$ a single lane is executed, making the payload processing similar to header and trailer processing. Based on a figure from [10].

2.2. NORX permutation function

NORX's core is the permutation function $F^{\ell}()$, applied to the NORX internal state S, with ℓ being the number of rounds. The state is a concatenation of 16 w-bit words in the form $S = s_0 \parallel \cdots \parallel s_{15}$, where the words $s_0, \cdots, s_{11}$ are called the *rate words*, where data is injected and extracted from, and the remaining words $s_{12}, \cdots, s_{15}$ are called *capacity words*. Conceptually, the state can be viewed as a 4×4 matrix:

$$S = \begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix}$$

A single permutation $F()$ processes the state S by applying the G function to the matrix's columns and then diagonals. The G function is described in Algorithm 1, and the permutation F is specified in Algorithm 2.

Algorithm 1 NORX G permutation function

Function G
  input: a, b, c, d ▷ Four words of the State
  $a \leftarrow (a \oplus b) \oplus ((a \wedge b) \ll 1)$
  $d \leftarrow (a \oplus d) \ggg r_0$
  $c \leftarrow (c \oplus d) \oplus ((c \wedge d) \ll 1)$
  $b \leftarrow (c \oplus b) \ggg r_1$
  $a \leftarrow (a \oplus b) \oplus ((a \wedge b) \ll 1)$
  $d \leftarrow (a \oplus d) \ggg r_2$
  $c \leftarrow (c \oplus d) \oplus ((c \wedge d) \ll 1)$
  $b \leftarrow (c \oplus b) \ggg r_3$
  output: a, b, c, d
end Function

NORX's encryption and decryption primitives can be described by Algorithm 3 and 4, where `header`, `branch`, `payload`, `merge`, `trailer` and `tag` are domain

Algorithm 2 NORX F round function

Function F
  input: $S, G()$ ▷ Norx State $s_0 \cdots s_{15}$ and $G()$ function
  / Processing the columns */*
  $s_0, s_4, s_8, s_{12} \leftarrow G(s_0, s_4, s_8, s_{12})$
  $s_1, s_5, s_9, s_{13} \leftarrow G(s_1, s_5, s_9, s_{13})$
  $s_2, s_6, s_{10}, s_{14} \leftarrow G(s_2, s_6, s_{10}, s_{14})$
  $s_3, s_7, s_{11}, s_{15} \leftarrow G(s_3, s_7, s_{11}, s_{15})$
  / Processing the diagonals */*
  $s_0, s_5, s_{10}, s_{15} \leftarrow G(s_0, s_5, s_{10}, s_{15})$
  $s_1, s_6, s_{11}, s_{12} \leftarrow G(s_1, s_6, s_{11}, s_{12})$
  $s_2, s_7, s_8, s_{13} \leftarrow G(s_2, s_7, s_8, s_{13})$
  $s_3, s_4, s_9, s_{14} \leftarrow G(s_3, s_4, s_9, s_{14})$
  output: S
end Function

separation constants; K is the key, N is the nonce, A is the additional data to be processed before the plaintext, `Msg` is the plaintext, Z is the additional data to be processed after the plaintext, `Tag` is the authentication tag, and `Cipher` is the ciphertext. For more details on the algorithm's description, the reader is invited to read chapter 2 of [10].

Algorithm 3 NORX AEAD encryption

Function ENCRYPT(K, N, A, Msg, Z)
  $S \leftarrow \text{initialise}(K, N)$
  $S \leftarrow \text{absorb}(S, A, \texttt{header})$
  $\bar{S} \leftarrow \text{branch}(S, |Msg|, \texttt{branch})$
  $\bar{S}, Cipher \leftarrow \text{encrypt}(\bar{S}, M, \texttt{payload})$
  $S \leftarrow \text{merge}(\bar{S}, |M|, \texttt{merge})$
  $S \leftarrow \text{absorb}(S, Z, \texttt{trailer})$
  $S, Tag \leftarrow \text{finalise}(S, \texttt{tag})$
  return $Cipher, Tag$
end Function

3. Platforms - ARM processors

In this section, we will briefly describe the target architecture of this work. The ARM –Advanced RISC Machine– architecture is a mainly 32-bit architecture owned by the British company ARM Holdings. The ARM architecture was introduced in 1985, and with more than 86 billion chips produced up to 2016, it has a big share of the consumer and embedded processor market [13].

The 32-bit ARM processors feature a Load/Store architecture without support for unaligned memory accesses, uniform 16×32-bit registers, and mostly a single clock cycle execution. Beyond that, the processors also feature conditional execution for most instructions and a 32-bit barrel shifter that can be used without affecting the performance with most arithmetic instructions and addresses calculations. The architecture can be divided into three main family lines: Cortex-M, Cortex-A, and Cortex-R. The Cortex-M

 © 2017 Sociedade Brasileira de Computação

Algorithm 4 NORX AEAD Decryption

Function DECRYPT($K, N, A, Cipher, Z, T$)
  $S \leftarrow \text{initialise}(K, N)$
  $S \leftarrow \text{absorb}(S, A, \texttt{header})$
  $\bar{S} \leftarrow \text{branch}(S, |C|, \texttt{branch})$
  $\bar{S}, Msg \leftarrow \text{decrypt}(\bar{S}, C, \texttt{payload})$
  $S \leftarrow \text{merge}(\bar{S}, |C|, \texttt{merge})$
  $S \leftarrow \text{absorb}(S, Z, \texttt{trailer})$
  $S, Tag' \leftarrow \text{finalise}(S, \texttt{tag})$
  if $Tag' == T$ **then return** Msg, Tag
  else
    return $\perp$ ▷ Symbol for failed decryption
  end if
end Function

cores are the simplest ones, with a focus on embedded systems with a low footprint and low energy requirements. The Cortex-A cores are more powerful, with the focus on power efficiency and they are deployed in a wide range of products. Lastly, the Cortex-R cores are suitable for high-performance real-time systems, where a high reliability is needed [14].

In this paper, the software implementations focus on the Cortex-A cores, namely Cortex-A7, A15, and A53, mainly for their large use in consumer electronics such as smartphones and tablets. Benchmarks of the implementations were also carried on the embedded processors containing Cortex-M4, M3, and M0 cores, as a way to evaluate the optimization impact on simpler processors.

The main characteristics of the target cores of this work are as follows [14]:

- Cortex-A7: Currently the most power efficient ARMv7-A core, with over a billion shipped units in production. The processor is capable of 40-bit physical addressing and has an eight-stage in-order pipeline. The A7 core is compatible with higher performance cores such as the Cortex-A15 and A17 for use with the big.LITTLE technology, where high-performance cores are combined with highly efficient cores in a heterogeneous computation approach.
- Cortex-A15: A high-performance ARMv7-A core, well suited to consumer items such as smartphones and embedded applications. As with other processors of the same line, it is capable of 40-bit physical addressing. It also features a 15 stage pipeline for integer calculations.
- Cortex-A53: An ARMv8-A core capable of seamlessly running both 32-bit and 64-bit code, and is made as an efficient 64-bit core for a low area and power footprint. Like the Cortex-A7, it is capable of being deployed together with high-end CPUs for chips with heterogeneous cores. The Cortex-A53 uses an efficient eight-stage in-order pipeline.

4. Implementation and Techniques

In the next sections, we discuss the optimization techniques applied to NORX, in order to obtain better performance in comparison to the state-of-art implementation [15]. Profiling the code to identify hotspots of interest to optimize was the first step in this work. For that,

© 2017 Sociedade Brasileira de Computação

the Linux tool `perf` was used to analyze the code. Results are shown in Figure 4. Notice that the call for `sha256_compress`, responsible for 8.86% of overhead, is not a part of NORX. Instead, it is used to generate pseudorandom inputs for the algorithm benchmark, and should not be considered into optimization efforts.

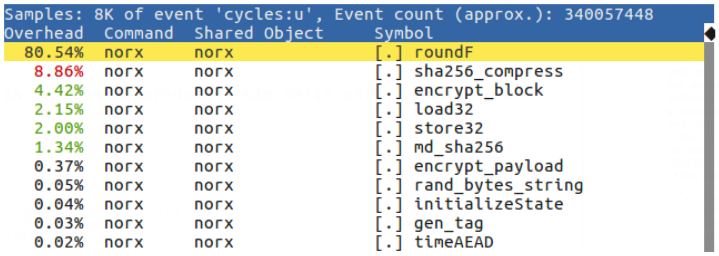


Figure 4. The result of the analysis of NORX, showing that the round function is responsible for about 70% of overhead. Tests carried out on an Odriod XU4 device, with a Cortex-A15 core.

4.1. Improving the use of the processor's pipeline

The function G is executed on the columns and then on the diagonals of the 4×4 matrix representation of NORX State, using Algorithm 5 for each column and diagonal. Since there is no dependency between each column and diagonal, the function G can be rewritten in a way that each step of G is executed right after the other, for each column or diagonal. This way, the execution can be arranged in groups of two columns or four columns. This approach allows a better use of the pipeline, improving the execution performance. Figure 5 illustrates the idea behind both approaches to optimize the function G.

Algorithm 5 NORX original implementation of G function

1: **Function** G
2: **input:** a, b, c, d ▷ Four words of the State
3: $a = (a \oplus b) \oplus ((a \wedge b) \ll 1)$
4: $d = d \oplus a$
5: $d = \mathtt{ROR}(d, \mathtt{r0})$
6: $c = (c \oplus d) \oplus ((c \wedge d) \ll 1)$
7: $b = b \oplus c$
8: $b = \mathtt{ROR}(b, \mathtt{r1})$
9: $a = (a \oplus b) \oplus ((a \wedge b) \ll 1)$
10: $d = d \oplus a$
11: $d = \mathtt{ROR}(d, \mathtt{r2})$
12: $c = (c \oplus d) \oplus ((c \wedge d) \ll 1)$
13: $b = b \oplus c$
14: $b = \mathtt{ROR}(b, \mathtt{r3})$
15: **output:** a, b, c, d
16: **end Function**
17: **Function** F
18: **input:** $S, \mathtt{G}()$ ▷ Whole Norx State and permutation function.
19: $(s_0, s_4, s_8, s_{12}) \leftarrow \mathtt{G}(s_0, s_4, s_8, s_{12})$ ▷ column step
20: $(s_1, s_5, s_9, s_{13}) \leftarrow \mathtt{G}(s_1, s_5, s_9, s_{13})$

 © 2017 Sociedade Brasileira de Computação

21: $(s_2, s_6, s_{10}, s_{14}) \leftarrow \mathtt{G}(s_2, s_6, s_{10}, s_{14})$
22: $(s_3, s_7, s_{11}, s_{15}) \leftarrow \mathtt{G}(s_3, s_7, s_{11}, s_{15})$
23: $(s_0, s_5, s_{10}, s_{15}) \leftarrow \mathtt{G}(s_0, s_5, s_{10}, s_{15})$ ▷ diagonal step
24: $(s_1, s_6, s_{11}, s_{12}) \leftarrow \mathtt{G}(s_1, s_6, s_{11}, s_{12})$
25: $(s_2, s_7, s_8, s_{13}) \leftarrow \mathtt{G}(s_2, s_7, s_8, s_{13})$
26: $(s_3, s_4, s_9, s_{14}) \leftarrow \mathtt{G}(s_3, s_4, s_9, s_{14})$
27: **output:** S
28: **end Function**

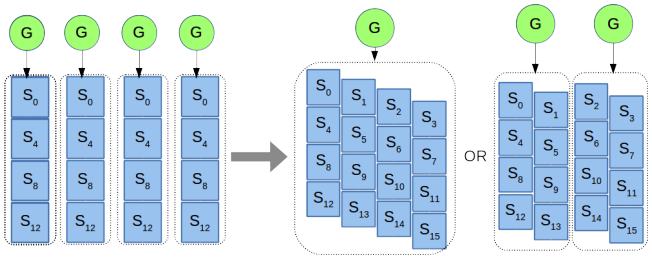


Figure 5. Illustration of two possible ways to reinterpret the G function; on the left, the original one. On the center, the 4-way pipeline, and on the right, the two-way pipeline. Notice that it is only shown the first column transformation.

In this way, the optimized code of function G for a 4-way pipeline is described in Algorithm 6 and the same function, optimized for a 2-way pipeline is described in Algorithm 7. In those algorithms `ROR(a, r)` is the bitwise right rotation of `a` by `r` bits; `ROR({a, b, c, d}, r)` is each word in the tuple `{a,b,c,d}` rotated by `r` bits and `r1`, `r2`, `r3`, `r4` are NORX rotation constants. Notice that, in order to execute a complete $F()$ function, $G4()$ must be called twice, with a different argument order in the second call to execute the diagonal step. Similarly, $G2()$ must be called a total of four times in order to execute the column and diagonal steps.

Algorithm 6 The function G with 4-way pipeline optimization

1: **Function** G4
2: **input:** S ▷ Whole Norx State
3: $s_0 = (s_0 \oplus s_4) \oplus ((s_0 \wedge s_4) \ll 1)$
4: $s_1 = (s_1 \oplus s_5) \oplus ((s_1 \wedge s_5) \ll 1)$
5: $s_2 = (s_2 \oplus s_6) \oplus ((s_2 \wedge s_6) \ll 1)$
6: $s_3 = (s_3 \oplus s_7) \oplus ((s_3 \wedge s_7) \ll 1)$
7: $s_{12} = s_{12} \oplus s_0$; $s_{13} = s_{13} \oplus s_1$
8: $s_{14} = s_{14} \oplus s_2$; $s_{15} = s_{15} \oplus s_3$
9: $\{s_{12}, s_{13}, s_{14}, s_{15}\} = \mathtt{ROR}(\{s_{12}, s_{13}, s_{14}, s_{15}\}, \mathtt{r0})$
10: $s_8 = (s_8 \oplus s_{12}) \oplus ((s_8 \wedge s_{12}) \ll 1)$
11: $s_9 = (s_9 \oplus s_{13}) \oplus ((s_9 \wedge s_{13}) \ll 1)$
12: $s_{10} = (s_{10} \oplus s_{14}) \oplus ((s_{10} \wedge s_{14}) \ll 1)$
13: $s_{11} = (s_{11} \oplus s_{15}) \oplus ((s_{11} \wedge s_{15}) \ll 1)$
14: $s_{12} = s_{12} \oplus s_0$; $s_{13} = s_{13} \oplus s_1$
15: $s_{14} = s_{14} \oplus s_2$; $s_{15} = s_{15} \oplus s_3$
16: $\{s_4, s_5, s_6, s_7\} = \mathtt{ROR}(\{s_4, s_5, s_6, s_7\}, \mathtt{r1})$
17: $s_0 = (s_0 \oplus s_4) \oplus ((s_0 \wedge s_4) \ll 1)$

© 2017 Sociedade Brasileira de Computação

18: $s_1 = (s_1 \oplus s_5) \oplus ((s_1 \wedge s_5) \ll 1)$
19: $s_2 = (s_2 \oplus s_6) \oplus ((s_2 \wedge s_6) \ll 1)$
20: $s_3 = (s_3 \oplus s_7) \oplus ((s_3 \wedge s_7) \ll 1)$
21: $s_{12} = s_{12} \oplus s_0$; $s_{13} = s_{13} \oplus s_1$
22: $s_{14} = s_{14} \oplus s_2$; $s_{15} = s_{15} \oplus s_3$
23: $\{s_{12}, s_{13}, s_{14}, s_{15}\} = \texttt{ROR}(\{s_{12}, s_{13}, s_{14}, s_{15}\}, \texttt{r2})$
24: $s_8 = (s_8 \oplus s_{12}) \oplus ((s_8 \wedge s_{12}) \ll 1)$
25: $s_9 = (s_9 \oplus s_{13}) \oplus ((s_9 \wedge s_{13}) \ll 1)$
26: $s_{10} = (s_{10} \oplus s_{14}) \oplus ((s_{10} \wedge s_{14}) \ll 1)$
27: $s_{11} = (s_{11} \oplus s_{15}) \oplus ((s_{11} \wedge s_{15}) \ll 1)$
28: $s_{12} = s_{12} \oplus s_0$; $s_{13} = s_{13} \oplus s_1$
29: $s_{14} = s_{14} \oplus s_2$; $s_{15} = s_{15} \oplus s_3$
30: $\{s_4, s_5, s_6, s_7\} = \texttt{ROR}(\{s_4, s_5, s_6, s_7\}, \texttt{r3})$
31: **output:** S
32: **end Function**

Algorithm 7 The function G with 2-way pipeline optimization

1: **Function** GH
2: **input:** $s_0, s_1, s_4, s_5, s_8, s_9, s_{12}, s_{13}$
3: ▷ Either two columns or diagonals of State.
4: $s_0 = (s_0 \oplus s_4) \oplus ((s_0 \wedge s_4) \ll 1)$
5: $s_1 = (s_1 \oplus s_5) \oplus ((s_1 \wedge s_5) \ll 1)$
6: $s_{12} = s_{12} \oplus s_0$
7: $s_{13} = s_{13} \oplus s_1$
8: $s_{12} = \texttt{ROR}(s_{12}, \texttt{r0})$
9: $s_{13} = \texttt{ROR}(s_{13}, \texttt{r0})$
10: $s_8 = (s_8 \oplus s_{12}) \oplus ((s_8 \wedge s_{12}) \ll 1)$
11: $s_9 = (s_9 \oplus s_{13}) \oplus ((s_9 \wedge s_{13}) \ll 1)$
12: $s_4 = s_4 \oplus s_8$
13: $s_5 = s_5 \oplus s_9$
14: $s_4 = \texttt{ROR}(s_4, \texttt{r1})$
15: $s_5 = \texttt{ROR}(s_5, \texttt{r1})$
16: $s_0 = (s_0 \oplus s_4) \oplus ((s_0 \wedge s_4) \ll 1)$
17: $s_1 = (s_1 \oplus s_5) \oplus ((s_1 \wedge s_5) \ll 1)$
18: $s_{12} = s_{12} \oplus s_0$
19: $s_{13} = s_{13} \oplus s_1$
20: $s_{12} = \texttt{ROR}(s_{12}, \texttt{r2})$
21: $s_{13} = \texttt{ROR}(s_{13}, \texttt{r2})$
22: $s_8 = (s_8 \oplus s_{12}) \oplus ((s_8 \wedge s_{12}) \ll 1)$
23: $s_9 = (s_9 \oplus s_{13}) \oplus ((s_9 \wedge s_{13}) \ll 1)$
24: $s_4 = s_4 \oplus s_8$
25: $s_5 = s_5 \oplus s_9$
26: $s_4 = \texttt{ROR}(s_4, \texttt{r3})$
27: $s_5 = \texttt{ROR}(s_5, \texttt{r3})$
28: **output:** $s_0, s_1, s_4, s_5, s_8, s_9, s_{12}, s_{13}$
29: **end Function**
30: **Function** G2

© 2017 Sociedade Brasileira de Computação

31: **input:** $S, GH()$ ▷ Whole Norx State and the 2-col permutation
32: $(s_0, s_1, s_4, s_5, s_8, s_9, s_{12}, s_{13}) \leftarrow GH(s_0, s_1, s_4, s_5, s_8, s_9, s_{12}, s_{13})$
33: $(s_2, s_3, s_6, s_7, s_{10}, s_{11}, s_{14}, s_{15}) \leftarrow GH(s_2, s_3, s_6, s_7, s_{10}, s_{11}, s_{14}, s_{15})$
34: **output:** S
35: **end Function**

In algorithm 7, the even numbered lines execute $G()$ on a line or column of the internal state, while the odd numbered lines execute the same instruction on a independent line or column of the state. This allows issue of independent `xor` instructions, such as in lines 6 and 7. Similarly, the same will happen with `and` and `lsh` instructions. The same idea is applied further on 6, but instead of only issuing instructions for two independent sets of state words, the whole state is operated at once. This allows the code to issue most of instructions in sets of four, without dependencies amongst them. This is specially useful for processors with a deep pipeline.

Regarding the security of those optimizations, we used Flowtracker to analyze the behavior of the algorithm, and the code runs in constant time. Beyond that, NORX is resistant against side-channel attacks by design, with cryptanalysis regarding differential, algebraic, fixed-point, slide, and rotational attacks, [10]. Furthermore, there are no table-lookups, no branching, or loops dependent on secret data. For implementation correctness, we compared our outputs with the reference algorithm, using the same set of input data. We verified internal consistency using encryption-decryption of random sets of plaintext, nonce, and keys.

4.2. Code improvements

A few minor improvements were also applied on the code. The ones with a positive impact on the code performance were:

- Extensive use of preprocessor macros and function inlining, which avoids overhead while still keeping code readability.
- Avoid the use of temporary variables whenever possible, doing most of the encryption, decryption, and additional data processing in place.
- Using a prefix operation instead of a postfix one on loop counters yields small improvements, more visible on Cortex-M based processors.
- Initialize the sponge using constants instead of calculating it as $F^2(0 \parallel 1 \parallel 2 \parallel \cdots \parallel 15)$, where each number j is represented as using w bits.
- Where possible, concatenate shift and rotate operations together with arithmetic operations, as to allow the use of the target processor's barrel shifter, making the shift operation free.

Other approaches were tested, such as replacing `memcpy()` calls with loops, manually unrolling loops and changing memory alignment. Those did not impact the performance in any significant way, resulting in negligible variations in cycle count.

5. Benchmarks and results

The benchmarks were carried out on an Odroid XU4 device running Arch Linux for the Cortex-A7 and Cortex-A15 cores, and on an Odroid-C2 device for the Cortex-A53, running the same OS. The code was compiled using GCC 6.3.1. Each test consists of the encryption of random data from `/dev/urandom` with lengths between 128 bytes

© 2017 Sociedade Brasileira de Computação

to 1 megabyte in powers of two increments. Tests were also carried on an Arduino Zero, with a Cortex-M0 core; an Arduino Due with a Cortex-M3 core and a Teensy 3.2 with a Cortex-M4 core. On the Cortex M4, M3, and M0 devices, the codes were compiled using `arm-none-eabi-gcc 4.8.3`. The compilation flags used were: `-O3 -Wall -Wextra -std=c99 -fno-schedule-insns -fomit-frame-pointer -Wno-old-style-declaration -funroll-loops -fpeel-loops` We consider that the cycle counter on the target processors show consistent values, and it is adequate to compare our implementations with the reference ones. For ARM-v7-A architecture, the following inline assembler code is used to enable the access to performance counter, and return the value of the performance registers which can be used to measure elapsed clock cycles between calls of the instruction `asm("mrc p15, 0, %0, c9, c13, 0" : "=r"(value));`. For AArch64 compatible processors, the following code is used: `asm ("mrs %0, pmccntr_el0" : "=r" (r));` .

Lastly, these calls are done before and after the call to the encryption or decryption function, and with the difference between the two measures, the average cycle per byte is measured, and the median of multiple measurements is reported in this work. This methodology is similar to the one used by SUPERCOP (System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives)[16].

Table 2. Cycles per byte for NORX encryption. Plaintext length of 256KiB on the 32bit processors. The best result for each plataform and cypher is in bold type.

		Ref. code	4x pipe	2x pipe	Speedup
NORX 3261	Cortex A7	29.45	29.70	**24.72**	16%
	Cortex A15	17.77	**14.23**	15.16	20%
NORX 6461	Cortex A7	48.52	50.09	**46.65**	4%
	Cortex A15	33.83	**26.76**	28.33	21%
NORX 3264	Cortex A7	28.46	33.74	**26.50**	7%
	Cortex A15	16.88	**15.26**	15.37	10%

Table 3. Cycles per byte for NORX encryption on the 64-bit platform. Plaint text length of 256KiB. The best result for each plataform and cypher is in bold type.

	Ref.	4x pipe	2x pipe	Speedup
NORX 3261	19.55	**10.94**	12.27	44%
NORX 6461	10.29	**5.84**	6.58	43%
NORX 3264	19.42	**12.08**	13.06	38%

In Table 2, we show the results for the 32-bit processors, namely Cortex A7 and A15; in Table 3 we show the results for the 64-bit Cortex A53 processor; lastly, in Table 4, we show the results on Cortex-M processors. We choose to show the average CBP –cycle per byte– with an input length of 256KiB since it better dilutes the overhead values, without risking overflow on the cycle counting registers, except in the Cortex-M based processors, due to memory constraints.

For the 32-bit variant of NORX, a 4× pipeline implementation is faster than the reference code in up to 20% on a 32-bit ARM and 44% on the 64-bit Cortex-A53. Our optimized implementation is faster than the reference NEON implementation: the 2× pipeline implementation is 12% faster than the reference code on the Cortex A7 core; the

 © 2017 Sociedade Brasileira de Computação

Table 4. Perfomance of NORX3261 (cycles per byte) on 32-bit Cortex-M architecture.

Cortex model	Size	No pipeline optimizations	Ref. code	4x pipe	2x pipe
M0	8KiB	99.52	100.12	111.84	99.96
M3	32KiB	49.96	50.49	67.21	66.26
M4	16KiB	49.96	50.49	47.28	66.26

4× pipeline implementation is 22% faster on the Cortex A15. While NORX has a SIMD friendly construction, with the internal state fitting in four 128-bit NEON registers, there are two extra transformations needed in each application of the function G in order to align the words between the column and diagonal steps. This transformation requires three extra pairs of SIMD load and store instructions, two `vext.8` instructions, and a `vwsp` instruction. We believe that this, together with the extra cost needed to transfer data from the NEON registers back to the ARM registers every round, coupled with the optimal usage of the pipeline makes our solution better than using SIMD instructions for these cores.

For the 64-bit variant of NORX, a 2× pipeline is better suited for the Cortex-A7 processor, and a 4× pipeline for the Cortex-A15 processor, due to the differences in pipeline length. With SIMD instructions being adequate for larger volume of data, NORX6461 on the 32-bit platform shows better performance using SIMD instructions, mainly due to the 64-bit word rotations being expensive using the 32-bit ARM registers, in comparison to the neon approach, where the rotations of two words can be done at the same time in the 128-bit register. For Cortex A53, both pipeline implementations show satisfactory results, being 43% faster than the reference code. In relation to a NEON implementation, the 4× pipeline implementation is 39% faster and the 2× pipeline implementation is 31% faster. Similar to NORX3261, the presence of a native 64-bit register and a deep pipeline with 8 stages makes a pipeline oriented approach superior to the SIMD alternative.

The multisponge approach, NORX3264, shows similar behavior to that of the 32-bit single-sponge algorithm. Table 2 and Table 3 show the execution times for a single thread implementation running on a single core. The multisponge algorithm is better suited for a multithread implementation, with each thread being responsible for a instance of the internal state. Our tests show that such an approach can result in up to 76% speedup, in relation to the single thread implementation.

6. Conclusions

This work shows how a pipeline oriented optimization can yield significant performance improvements on an authenticated encryption algorithm, specifically NORX, outperforming NEON vectorial code in some situations, while at the same time using only portable C code. These optimizations also result in little to no performance penalties on smaller Cortex-M cores. We believe that these techniques can also be applied to other algorithms that use similar constructions, resulting in a better performance with little drawbacks.

7. Aknowledgements

This work was supported by LG Electronics via the ”Crypto for IOT” project. The second author was partially supported by a research productivity grant from CNPq. We thank Diego Aranha for helping with the benchmark platforms.

© 2017 Sociedade Brasileira de Computação

References

[1] J. Aumasson, P. Jovanovic, and S. Neves, "NORX v3.0," norx.io/data/norx.pdf, Sep. 2016. [Online]. Available: norx.io/data/norx.pdf

[2] C. CAESAR, "Competition for authenticated encryption: Security, applicability, and robustness," http://competitions.cr.yp.to, Apr. 2013. [Online]. Available: http://competitions.cr.yp.to

[3] M. J. Dworkin, "Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac," *Special Publication (NIST SP)-800-38D*, 2007.

[4] P. Rogaway, "Authenticated-encryption with associated-data," in *ACM Conference on Computer and Communications Security*. ACM, 2002, pp. 98–107.

[5] M. Videau, "estream," in *Encyclopedia of Cryptography and Security (2nd Ed.)*. Springer, 2011, pp. 426–427.

[6] B. Preneel, "AHS competition/sha-3," in *Encyclopedia of Cryptography and Security (2nd Ed.)*. Springer, 2011, pp. 27–29.

[7] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Duplexing the sponge: single-pass authenticated encryption and other applications," *IACR Cryptology ePrint Archive*, vol. 2011, p. 499, 2011.

[8] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge functions," in *ECRYPT hash workshop*, vol. 2007. Citeseer, 2007.

[9] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "On the indifferentiability of the sponge construction," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 4965. Springer, 2008, pp. 181–197.

[10] J. Aumasson, P. Jovanovic, and S. Neves, "NORX: parallel and scalable AEAD," in *ESORICS (2)*, ser. Lecture Notes in Computer Science, vol. 8713. Springer, 2014, pp. 19–36.

[11] D. J. Bernstein, "Chacha, a variant of salsa20," in *Workshop Record of SASC*, vol. 8, 2008.

[12] A. Biryukov and D. Khovratovich, "PAEQ: parallelizable permutation-based authenticated encryption," in *ISC*, ser. Lecture Notes in Computer Science, vol. 8783. Springer, 2014, pp. 72–89.

[13] A. Holdings, "Arm: Media fact sheet," https://www.arm.com/-/media/arm-com/news/ARM-media-fact-sheet-2016.pdf?la=en, Sep. 2016. [Online]. Available: https://www.arm.com/-/media/arm-com/news/ARM-media-fact-sheet-2016.pdf?la=en

[14] ——, "Processors cortex-a," http://www.arm.com/products/processors/cortex-a, Mar. 2017. [Online]. Available: http://www.arm.com/products/processors/cortex-a

[15] J. Aumasson, P. Jovanovic, and S. Neves, "Norx reference implementations (software)," https://github.com/norx/norx, 2015.

[16] D. J. Bernstein, "Supercop: System for unified performance evaluation related to cryptographic operations and primitives," 2009.

 © 2017 Sociedade Brasileira de Computação

A Zero-Knowledge Proof for the Hidden Subset Sum Problem

Charles F. de Barros[1]

[1]Departamento de Ciência da Computação (DCOMP)
Universidade Federal de São João Del Rei (UFSJ)
São João Del Rei – MG – Brazil

charlesbarros@ufsj.edu.br

Abstract. *In this paper, we propose a zero-knowledge proof for a special case of the hidden subset sum problem. This problem was presented by [Boyko et al. 1998] as the underlying problem of methods for generating random pairs of the form* $(x, g^x \pmod p)$ *using precomputations. The proof we propose is an adaptation of a zero-knowledge protocol for the subset sum problem presented by [Blocki 2009].*

1. Introduction

Zero-knowledge proofs have interesting cryptographic applications, as they provide means for one party to prove knowledge of some secret, without revealing any information about that secret to the other party. This is a very desirable property in authentication scenarios, for example.

It is known that some conventional authentication protocols, such as password-based ones, may be vulnerable to offline attacks, based on the use of dictionaries or rainbow tables, because hashes of the passwords must be transmitted or stored somewhere. If an adversary has access to these hashes, he may compare them to a list of pre-computed hashes of known passwords. Whenever a match is found, he is able to authenticate to the system.

In this scenario, we say that the adversary *learns* something by intercepting the password hash, as long as he is able to compare it to other hashes. In a zero-knowledge protocol, the adversary learns nothing by eavesdropping on the communication between the two parties.

A simple way of applying this concept to the design of authentication protocols is to provide zero-knowledge proofs for hard mathematical problems. In this case, one party could prove that she knows the solution for a given instance of the problem, without revealing any information about that solution.

In this paper, we propose a zero-knowledge proof for the hidden subset sum problem. The proof we propose is based on a protocol by [Blocki 2009] for the classical subset sum problem. The hidden subset sum problem was presented by [Boyko et al. 1998] as the underlying problem of methods for generating randomly distributed pairs of the form $(x, g^x \pmod p)$ using precomputations. This new mathematical problem was considered potentially as hard, or even harder than the classical subset sum problem, since no algorithm was known to solve it, apart from exhaustive search.

 © 2017 Sociedade Brasileira de Computação

The hardness of the problem was thoroughly analysed by [Nguyen and Stern 1999], which established a security criterion for using the generators based on hidden subset sums. They proposed a lattice-based attack, which was likely to succeed whenever the parameters of the problem satisfied certain conditions.

Nonetheless, no further cryptographic applications for the hidden subset sum problem were proposed, maybe due to the lack of a more rigorous hardness result. In this paper, we shed some light on the subject by presenting a possible relation between the hardness of the hidden subset sum problem and the hardness of the classical subset sum problem.

1.1. Roadmap

This paper is organized as follows: in Section 2, we fix some notations and present some theoretical review on subset sums, hidden subset sums and interactive zero-knowledge proofs. In Section 3, we review Blocki's zero-knowledge proof for the subset sum problem, and in Section 4 we extend Blocki's protocol to the hidden subset sum problem. A brief performance analysis is presented in Section 5, and in Section 6 we present our final remarks and proposals for future work.

2. Theoretical Review

In this section, we provide a brief review on subset sums, hidden subset sums and zero-knowledge proofs. From now on, we denote by $\mathcal{I}_n$ the set of indices $\{1, \cdots, n\}$, and the cardinality of a set $\mathcal{A}$ shall be denoted by $|\mathcal{A}|$. We also use the symbol $\oplus$ to denote the xor operation (sum modulo 2).

2.1. The Subset Sum Problem

The subset sum problem is well known in complexity theory. Essentially, it consists of, given a set

$$\mathcal{A} = \{a_1, \cdots, a_n\} \tag{1}$$

of integers and a *target* integer t, deciding whether exists a subset of $\mathcal{A}$ whose sum is t. The problem is known to be NP-complete [Cormen et al. 2001].

We may think of the problem of deciding whether exists a subset of indices $\mathcal{S} \subset \mathcal{I}_n$ such that

$$\sum_{j \in \mathcal{S}} a_j = t. \tag{2}$$

An interesting variant of the subset sum problem consists of finding two subsets of the same size with the same sum. This variant is known as the *equal partition problem*.

Problem 2.1 (Equal Partition Problem) *Given a set $\mathcal{A} = \{a_1, \cdots, a_n\} \subset \mathbb{Z}$, find a partition $\mathcal{I}_n = \mathcal{S}_1 \cup \mathcal{S}_2$ such that*

1. $|\mathcal{S}_1| = |\mathcal{S}_2| = n/2$;
2. $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$; *and*
3. $\sum_{j \in \mathcal{S}_1} a_j = \sum_{j \in \mathcal{S}_2} a_j = t$ *for some target integer* t.

 © 2017 Sociedade Brasileira de Computação

Note that the problem can also be formulated over the ring of integers modulo M, for some integer M. In fact, given M, n positive integers, $t \in \mathbb{Z}_M$ and a set

$$\mathcal{A} = \{a_1, \cdots, a_n\} \subset Z_M, \tag{3}$$

decide whether exists $\mathcal{S} \subset \mathcal{I}_n$ such that

$$\sum_{j \in \mathcal{S}} a_j \equiv t \pmod{M}. \tag{4}$$

In its non-decisional version, the goal of the subset sum problem is to find the subset $\mathcal{S}$, if such exists. Note that both the equal partition and the modular variants are still NP-complete. In fact, we can reduce any instance of subset sum to an instance of equal partition by simply appending n zeros to the input set (considered here as having cardinality n). On the other hand, any instance $\mathcal{A} = \{a_1, \cdots, a_n\}$ of the subset sum problem over the integers is also an instance over $\mathbb{Z}_{M+1}$, where $M = \sum_{j=1}^{n} a_j$.

Despite the fact that the problem is NP-complete, there are methods to solve "easy" instances, such as superincreasing sequences and low-density subset sums. We focus our attention on the latter. We define the *density* in (1) as

$$d = \frac{n}{\log_2(\max\limits_{1 \leq i \leq n} a_i)}. \tag{5}$$

Similarly, in (3), the density is given by

$$d = \frac{n}{\log_2(M)}. \tag{6}$$

It was shown by [Lagarias and Odlyzko 1985] that subset sums with sufficiently low density, namely less than $0.6463\cdots$ can be efficiently solved by using lattice reduction methods, such as [Lenstra et al. 1982] and [Schnorr 1994]. Posteriorly, [Coster et al. 1992] improved the method to solve subset sums with density less than $0.9408\cdots$.

The subset sum problem is a special case of the knapsack problem, which underlies some well known public-key cryptosystems, such as [Merkle and Hellman 1978], [Chor and Rivest 1988] and [T. Okamoto and Uchiyama 2000]. The core idea of a knapsack-based cryptosystem is to use an "easy" knapsack as the secret key (a superincreasing one, for example), while the public key is a "hard" knapsack. However, these cryptosystems may be prone to low-density attacks, which exploit the fact that the public key is a knapsack with low density.

2.2. The Hidden Subset Sum Problem

The hidden subset sum problem, as the name suggests, was inspired by the classical subset sum problem. It was presented by [Boyko et al. 1998] as the underlying problem of methods for generating random pairs of the form $(x, g^x \pmod{p})$ using precomputation. This is useful for protocols based on the discrete-logarithm problem, such as El Gamal [El Gamal 1985] and DSS [NIST 1994] signatures, where generation of the aforementioned pairs is considerably expensive.

© 2017 Sociedade Brasileira de Computação

We present here a brief description of one of the generators based on the hidden subset sum problem. Keep in mind that the goal is to generate randomly distributed pairs of the form $(x, g^x \pmod p)$. Firstly, choose a prime number p and $g \in \mathbb{Z}_p^*$ of order M. Generate random integers $a_1, \cdots, a_n \in \mathbb{Z}_M$ and compute $\beta_i = g^{a_i}$, for $i = 1, \cdots, n$. Store the $a_i's$ and the $\beta_i's$ in a table.

Whenever a pair $(x, g^x \pmod p)$ is needed, choose a random subset $S \subset \mathcal{I}_n$ and compute

$$b = \sum_{i \in S} a_i \pmod M. \tag{7}$$

If $b = 0$, stop and start again with a new subset S. As soon as $b \neq 0$, compute

$$B = \prod_{i \in S} \beta_i \pmod p. \tag{8}$$

The value of x is given by b and g^x is given by B. Output the pair (b, B), where we have clearly that $B = g^b \pmod p$. In an active attack, an adversary may gather many values of b, say $b_1, \cdots, b_m$, and try to figure out the values of the a_i's. This is an instance of the hidden subset sum problem, which we state below.

Problem 2.2 (Hidden Subset Sum Problem) *Let n, m, M be positive integers. We are given $\mathcal{B} = \{b_1, \cdots, b_m\} \subset \mathbb{Z}_M$, and the goal is to find $\mathcal{A} = \{a_1, \cdots, a_n\} \subset \mathbb{Z}_M$ such that, for all $i = 1, \cdots, m$, there exists $\mathcal{S}_i \subset \mathcal{I}_n$ such that*

$$b_i \equiv \sum_{j \in \mathcal{S}_i} a_j \pmod M. \tag{9}$$

Until the publication of the work by [Nguyen and Stern 1999], no algorithm to solve this problem was known. In their paper, Nguyen and Stern present a low-density attack on the hidden subset sum problem, exploiting the low density of the hidden sequence $a_1, \cdots, a_n$. After running a series of experiments, they conclude that, for the case where $d = 1$, the attack is already very unlikely to succeed, even if sparse subset sums are chosen.

We remark that, although there are no formal hardness results for the hidden subset sum problem, we have strong reasons to believe that it is at least as hard as the classical subset sum problem.

In order to throw some light on this discussion, we must precisely define what it means to solve an instance of the hidden subset sum problem. If by solving we mean simply outputting the set $\mathcal{A}$, then we are left with the problem of checking the validity of the solution, since we would need to solve m instances of the subset sum problem. Hence, we consider that an algorithm solves an instance of the hidden subset sum problem if it outputs the set $\mathcal{A}$ *and* the subsets $\mathcal{S}_i$, for $i = 1, \cdots, m$, so that the decisional version of the problem can be considered as a member of NP.

Provided with a precise notion of what it means to solve the hidden subset sum problem, we may state the following conjecture regarding its hardness:

Conjecture 2.1 *Let $\mathcal{O}$ be an oracle that takes as input three integers M, m, n and a set $\mathcal{B} = \{b_1, \cdots, b_m\} \subset \mathbb{Z}_M$, and outputs $\mathcal{A} = \{a_1, \cdots, a_n\} \in \mathbb{Z}_M$, together with the*

 © 2017 Sociedade Brasileira de Computação

subsets $\mathcal{S}_i \subset \mathcal{I}_n$ such that

$$b_i \equiv \sum_{j \in \mathcal{S}_i} a_j \pmod{M} \tag{10}$$

for $i = 1, \cdots, m$. Given $\mathcal{A} = \{a_1, \cdots, a_n\} \in \mathbb{Z}_M$ and an integer $t \in \mathbb{Z}_M$, can we find a subset $\mathcal{S} \subset \mathcal{I}_n$ such that

$$t \equiv \sum_{j \in \mathcal{S}} a_j \pmod{M}, \tag{11}$$

given access to $\mathcal{O}$?

We sketch a possible proof that the answer is yes, although there are some loose ends that need tying up. We build an instance $\mathcal{B} = \{b_1, \cdots, b_m\}$ of the hidden subset sum problem, where $b_1 = t$ and each b_i, for $i > 1$, is a randomly chosen subset sum of $\mathcal{A}$. Clearly, *there is a probability* that the oracle finds the desired solution, whenever $\mathcal{A}, t$ is a "yes" instance.

In fact, if $\mathcal{A}, t$ is a "yes" instance, then $\mathcal{A}$ is a solution for the hidden subset sum instance, which also contains the subset whose sum is t. Hence, we can solve a subset sum by solving a hidden subset sum. However, this may not be the only solution, since there are no guarantees as for the uniqueness of the solution for a hidden subset sum instance. For example, the instance $\mathcal{B} = \{0, 2, 5, 7\}$ has at least two solutions in $\mathbb{Z}_8$, namely $\{2, 3, 5, 7\}$ and $\{1, 2, 4, 6\}$.

The question remains open as for the number of solutions for a given instance of the hidden subset sum problem. If this number is polynomially bounded, and if we consider an oracle capable of finding all the possible solutions, then the answer for the conjecture above is "yes", and the hidden subset sum problem is at least as hard as the classical subset sum.

In this paper, we propose a variant of the hidden subset sum problem, in which every b_i is the sum of exactly $n/2$ elements of $\mathcal{A}$. We refer to this variant as the *balanced hidden subset sum problem*.

Problem 2.3 (Balanced Hidden Subset Sum Problem) *Let n, m, M be positive integers and a set $\mathcal{B} = \{b_1, \cdots, b_m\} \subset \mathbb{Z}_M$. The goal is to find $\mathcal{A} = \{a_1, \cdots, a_n\} \subset \mathbb{Z}_M$ such that, for all $i = 1, \cdots, m$, there exists $\mathcal{S}_i \subset \mathcal{I}_n$ such that $|\mathcal{S}_i| = n/2$ and*

$$b_i \equiv \sum_{j \in \mathcal{S}_i} a_j \pmod{M}. \tag{12}$$

For instance, if $M = 16$, $n = 4$, $m = 4$ and $\mathcal{B} = \{9, 6, 11, 4\}$, the set $\mathcal{A} = \{2, 7, 4, 13\}$ is a solution, with $\mathcal{S}_1 = \{1, 2\}$, $\mathcal{S}_2 = \{1, 3\}$, $\mathcal{S}_3 = \{2, 3\}$ and $\mathcal{S}_4 = \{2, 4\}$.

As for the hardness of this variant, we may consider a procedure, which is similar to what we have done before, in order to solve the equal partition problem, given access to an oracle that solves the balanced variant of the hidden subset sum problem. Therefore, we assume that for sufficiently high density, namely $d \geq 1$, the balanced variant is hard.

2.3. Interactive Zero-Knowledge Proofs

Zero-knowledge proofs were introduced by [Goldwasser et al. 1985]. In a zero-knowledge proof, also referred to as zero-knowledge protocol, a party called the Prover,

denoted by P, proves to another party (the Verifier, denoted by V) that a given statement S is true, without conveying any information apart from the fact that S is true. In this sense, V gains no *knowledge* from the proof, except for the trueness of the statement.

In a formal sense, this **zero-knowledge property** implies that V could *simulate* his interaction with P, and this simulation would be indistinguishable from a real interaction. This gives us the intuition that the information exchanged between P and V during the protocol must look random. In fact, if everything that V receives from P appears to be random, he gains no extra knowledge, because random data convey no information. Hence, V is able to generate by himself everything he sees from his interaction with P.

Besides the aforementioned property, zero-knowledge proofs must be **complete** and **sound**. Completeness means that, if P is honest and both P and V follow the protocol, then V will accept the proof with probability 1. The protocol is sound if any dishonest prover P' is caught with high probability, provided that the protocol is repeated a sufficiently large number of times. The reader may consult [Quisquater and Berson 1990] for a very interesting illustration of a zero-knowledge proof.

As already mentioned, zero-knowledge proofs may be applied to authentication protocols. In an authentication system, one party must prove its identity to a second party, usually by showing that it knows some secret information (a password, for instance). The problem with this scenario is that, in order to prove knowledge of some secret, typically one must convey information about this secret. For example, one party computes the hash of its secret password and sends it to the second, which gives an adversary the opportunity of launching, for example, a dictionary-based attack against the system.

Zero-knowledge proofs offer the advantage of conveying no information at all about the secret of which one wishes to prove knowledge. Another practical scenario of application includes the proof that a given public key is good (satisfies all the security criteria), without revealing the secret key. For example, proving that an RSA key is the product of two safe primes.

3. A Zero-Knowledge Proof for the Subset Sum Problem

In this section, we review Blocki's zero-knowledge proof for the subset sum problem [Blocki 2009]. In his thesis, Blocki presents a protocol for the equal partition variant, by which the prover P is able to convince the verifier V that he knows a solution for the problem, without conveying any information about this solution.

Let $\mathcal{A} = \{a_1, \cdots, a_n\}$ be the input set, and t the target integer. Remind that the prover wishes to convince a verifier that he knows a set $\mathcal{S} \subset \mathcal{I}_n$ such that $|\mathcal{S}| = n/2$ and

$$\sum_{j \in \mathcal{S}} a_j = \sum_{j \in \mathcal{I}_n \setminus \mathcal{S}} a_j = t. \tag{13}$$

Defining

$$M = \sum_{i=1}^{n} a_i, \tag{14}$$

the protocol works as follows:

 © 2017 Sociedade Brasileira de Computação

1. The prover generates $\beta_1, \cdots, \beta_n \in \mathbb{Z}_{M+1}$ uniformly at random and computes $\gamma_1, \cdots, \gamma_n \in \mathbb{Z}_{M+1}$ such that
$$a_i \equiv \beta_{\sigma(i)} + \gamma_{\sigma(i)} \pmod{M+1}. \tag{15}$$
for $i = 1, \cdots, n$, where σ is a random permutation, which P does not reveal.
2. The prover computes the partial sums
$$b = \sum_{j \in \mathcal{S}_\sigma} \beta_j \tag{16}$$
and
$$c = \sum_{j \in \mathcal{S}_\sigma} \gamma_j, \tag{17}$$
where
$$\mathcal{S}_\sigma = \{\sigma(i) \mid i \in \mathcal{S}\}. \tag{18}$$
3. In the next step, the verifier asks to see exactly one of the following:
   (a) All of the triples $(a_i, \beta_{\sigma(i)}, \gamma_{\sigma(i)})$, checking whether
   $$a_i \equiv \beta_{\sigma(i)} + \gamma_{\sigma(i)} \pmod{M+1}; \tag{19}$$
   (b) $\beta_1, \cdots, \beta_n, b, c$ and $\mathcal{S}_\sigma$, checking whether
   $$\sum_{j \in \mathcal{S}_\sigma} \beta_j \equiv b \pmod{M+1} \tag{20}$$
   and
   $$b + c \equiv t \pmod{M+1}; \tag{21}$$
   (c) $\gamma_1, \cdots, \gamma_n, b, c$ and $\mathcal{S}_\sigma$, checking whether
   $$\sum_{j \in \mathcal{S}_\sigma} \gamma_j \equiv c \pmod{M+1} \tag{22}$$
   and
   $$b + c \equiv t \pmod{M+1}. \tag{23}$$
4. If any of the checks fails, the verifier rejects the proof.

It is easy to see that the protocol is complete. In fact, any prover who knows the subset $\mathcal{S}$ is able to correctly build the subset $\mathcal{S}_\sigma$ and, by following the protocol, he always convinces the verifier, because

$$b + c = \sum_{j \in \mathcal{S}_\sigma} \beta_j + \sum_{j \in \mathcal{S}_\sigma} \gamma_j = \sum_{j \in \mathcal{S}_\sigma} (\beta_j + \gamma_j) = \sum_{i \in \mathcal{S}} (\beta_{\sigma(i)} + \gamma_{\sigma(i)}) = \sum_{i \in \mathcal{S}} a_i = t. \tag{24}$$

The soundness of the protocol comes from the fact that, if a dishonest prover P' is cheating, then he does not know the subset $\mathcal{S}$, which means that he is not able of choosing the proper subset $\mathcal{S}_\sigma$. Instead, he chooses a random subset $\mathcal{S}'$ (which is the wrong subset with overwhelming probability), which leaves him with a probability at least $1/3$ of being caught (at least one of the checkings will fail if the wrong subset was chosen).

Finally, the protocol also has the zero-knowledge property, because each β_i is just a random number, while each γ_i is also a random number without β_i. Thus, the verifier is able to generate by himself everything that the prover shows him at each step of the protocol. Then, he is able to simulate any of the choices (a), (b) and (c) as follows:

© 2017 Sociedade Brasileira de Computação

1. In order to simulate choice (a), he simply generates $\beta_1, \cdots, \beta_n$ at random and computes $\gamma_1, \cdots, \gamma_n$ such that $\beta_i + \gamma_i \equiv a_i \pmod{M+1}$;
2. In order to simulate choice (b), he generates $\beta_1, \cdots, \beta_n$ uniformly at random, chooses a random subset $\mathcal{S}' \subset \mathcal{I}_n$ such that $|\mathcal{S}'| = n/2$, computes

$$b = \sum_{j \in \mathcal{S}'} \beta_j \tag{25}$$

   and picks c such that $b + c \equiv k \pmod{M+1}$.
3. Analogously, in order to simulate choice (c), he generates $\gamma_1, \cdots, \gamma_n$ uniformly at random, chooses a random subset $\mathcal{S}' \subset \mathcal{I}_n$ such that $|\mathcal{S}'| = n/2$, computes

$$c = \sum_{j \in \mathcal{S}'} \gamma_j \tag{26}$$

   and picks b such that $b + c \equiv k \pmod{M+1}$.

4. Extending the Idea to the Hidden Subset Sum Problem

In this section, we extend Blocki's protocol and present a novel zero-knowledge proof for the hidden subset sum problem. Let us denote the number of combinations of x elements taken y at a time by $\binom{x}{y}$. From now on, let m, n, M be positive integers such that

$$n! \geq \binom{M}{n} \tag{27}$$

and

$$n \geq \log_2 M. \tag{28}$$

The previous inequality implies that the hidden subset has density equal to or greater than 1, in order to mitigate low-density attacks. Furthermore, assume that

$$\left(\frac{m+2}{2m}\right)^m \tag{29}$$

is a negligible amount. A set $\mathcal{B} = \{b_1, \cdots, b_m\} \subset \mathbb{Z}_M$ is given, and P secretly knows another set $\mathcal{A} = \{a_1, \cdots, a_n\} \subset \mathbb{Z}_M$, of which he wishes to prove knowledge, such that each b_i is a subset sum modulo M of exactly $n/2$ elements of $\mathcal{A}$. The protocol works as follows:

1. V randomly chooses $\mathcal{A}_0 = \left\{a_1^{(0)}, \cdots, a_n^{(0)}\right\} \subset \mathbb{Z}_M$ from the uniform distribution and sends it to P.
2. P reveals the set

$$\mathcal{A}_1 = \left\{a_1^{(1)}, \cdots, a_n^{(1)}\right\} \subset \mathbb{Z}_M \tag{30}$$

   such that, for all $i = 1, \cdots, n$,

$$a_i \equiv a_i^{(0)} + a_{\sigma(i)}^{(1)} \pmod{M}, \tag{31}$$

   where σ is a random permutation which P keeps secret.

 © 2017 Sociedade Brasileira de Computação

3. P computes the subsets $\mathcal{S}_i^{(0)}, \mathcal{S}_i^{(1)} \subset \mathcal{I}_n$, satisfying $|\mathcal{S}_i^{(0)}| = |\mathcal{S}_i^{(1)}| = n/2$ for all $i = 1, \cdots, m$, and computes

$$b_i^{(0)} \equiv \sum_{j \in \mathcal{S}_i^{(0)}} a_j^{(0)} \pmod{M} \tag{32}$$

and

$$b_i^{(1)} \equiv \sum_{j \in \mathcal{S}_i^{(1)}} a_j^{(1)} \pmod{M}, \tag{33}$$

where

$$b_i \equiv b_i^{(0)} + b_i^{(1)} \pmod{M} \tag{34}$$

for all $i = 1, \cdots, m$.
4. P reveals the pairs $(b_i^{(0)}, b_i^{(1)})$, for $i = 1, \cdots, m$. If (34) does not hold, V immediately rejects.
5. V sends to P a random bit $w \in \{0, 1\}$ and a random integer $1 \leq k \leq m$.
6. P reveals $\mathcal{S}_k^{(w)}$.
7. V accepts if, and only if

$$\sum_{j \in \mathcal{S}_k^{(w)}} a_j^{(w)} \equiv b_k^{(w)} \pmod{M} \tag{35}$$

and $|\mathcal{S}_k^{(w)}| = n/2$.

The protocol is repeated at least m times. Now we prove that the presented protocol has the properties of completeness, soundness and zero-knowledge.

Theorem 4.1 (Completeness) *The protocol is complete.*

Proof: in fact, let us assume that P is an honest prover, who actually knows the set $\mathcal{A}$. Hence, for each $i = 1, \cdots, m$, he knows $\mathcal{S}_i \subset \mathcal{I}_n$ such that

$$\sum_{j \in \mathcal{S}_i} a_j \equiv b_i \pmod{M}. \tag{36}$$

Hence, for $i = 1, \cdots, m$, he may build the subsets $\mathcal{S}_i^{(0)}$ and $\mathcal{S}_i^{(1)}$ as follows:

$$\mathcal{S}_i^{(0)} = \mathcal{S}_i \tag{37}$$

and

$$\mathcal{S}_i^{(1)} = \{\sigma(j) \in \mathcal{I}_n | j \in \mathcal{S}_i\}. \tag{38}$$

Thus, P is able to correctly compute the pairs $(b_i^{(0)}, b_i^{(1)})$ and, in fact,

$$b_i^{(0)} + b_i^{(1)} = \sum_{j \in \mathcal{S}_i^{(0)}} a_j^{(0)} + \sum_{j \in \mathcal{S}_i^{(1)}} a_j^{(1)} = \sum_{j \in \mathcal{S}_i} a_j^{(0)} + \sum_{j \in \mathcal{S}_i} a_{\sigma(j)}^{(1)}. \tag{39}$$

Since both sums are over the same index, we may write

$$b_i^{(0)} + b_i^{(1)} = \sum_{j \in \mathcal{S}_i} (a_j^{(0)} + a_{\sigma(j)}^{(1)}) = \sum_{j \in \mathcal{S}_i} a_j \equiv b_i \pmod{M}. \tag{40}$$

Therefore, V always accepts the proof, provided that P is honest and both follow the protocol. □

Theorem 4.2 (Soundness) *The protocol is sound.*

Proof: we must show that a dishonest prover P' is caught with high probability. In fact, after receiving the set $\mathcal{A}_0$, a smart cheater could choose m subsets $\mathcal{S}_i^{(0)}$ and compute

$$b_i^{(0)} \equiv \sum_{j \in \mathcal{S}_i^{(0)}} a_j^{(0)}, \tag{41}$$

for $i = 1, \cdots, m$. At this point, the integers $b_i^{(1)}$ are already determined, but the cheater may conveniently choose $n/2$ integers whose sum is, say, $b_1^{(1)}$, and other $n/2$ integers whose sum is $b_2^{(1)}$. It determines the set $\mathcal{A}_1$ along with the subsets $\mathcal{S}_1^{(1)}$ and $\mathcal{S}_2^{(1)}$. In order to determine the subsets $\mathcal{S}_i^{(1)}$ for $i > 2$, he must solve $m - 2$ instances of the subset sum problem over $\mathcal{A}_1$, hence we assume that he is not capable of doing so. Nevertheless, at each iteration, the dishonest prover has two possibilities of being successful at cheating:

1. the verifier chooses the bit 0 at step 5, which happens with probability $1/2$;
2. the verifier chooses the bit 1 and one of the integers 1 or 2 at step 5, which happens with probability $1/m$,

which gives him a probability $1/2 + 1/m$ of successfully cheating. In order to deceive the verifier, the cheater must be successful in all iterations, which will happen with probability at most

$$\left(\frac{m+2}{2m}\right)^m. \tag{42}$$

Since we are assuming that this is a negligible amount, the cheating prover is caught with high probability. □

Theorem 4.3 (Zero Knowledge) *The protocol is zero knowledge.*

Proof: intuitively, everything that V sees looks random. Hence, he is able to generate by himself everything that he receives from P, which means that he may simulate his interaction with P as follows:

1. He chooses $\mathcal{A}_0 = \left\{a_1^{(0)}, \cdots, a_n^{(0)}\right\} \subset \mathbb{Z}_M$ from the uniform distribution.
2. The set $\mathcal{A}_1$ is just a random set, because $\mathcal{A}$ and the permutation σ are unknown. Thus, V simply chooses another random subset

$$\mathcal{A}_1 = \left\{a_1^{(1)}, \cdots, a_n^{(1)}\right\} \subset \mathbb{Z}_M. \tag{43}$$

3. V generates a random string $w \in \{0,1\}^m$, chooses a random subset $\mathcal{S}_i \subset \mathcal{I}_n$ with exactly $n/2$ elements, for all $i = 1, \cdots, m$, and computes

$$b_i^{(w_i)} \equiv \sum_{j \in \mathcal{S}_i} a_j^{(w_i)} \pmod{M}. \tag{44}$$

   Next, he computes $b_i^{(w_i \oplus 1)}$ such that

$$b_i \equiv b_i^{(0)} + b_i^{(1)} \pmod{M}. \tag{45}$$

4. V has simulated the pairs $(b_i^{(0)}, b_i^{(1)})$, for $i = 1, \cdots, m$.

 © 2017 Sociedade Brasileira de Computação

5. V chooses a random integer $1 \leq k \leq m$ and picks the bit w_k from the random string generated at step 3.
6. V simulates the disclosure of the subset $\mathcal{S}_k$.
7. In fact, by construction, we have

$$\sum_{j \in \mathcal{S}_k} a_j^{(w_k)} \equiv b_k^{(w_k)} \pmod{M} \tag{46}$$

and $|\mathcal{S}_k| = n/2$.

Note that, after his interaction with P, V knows that each element of $\mathcal{A}$ is the sum of an element of $\mathcal{A}_0$ with some element of $\mathcal{A}_1$. However, because he does not know the permutation σ chosen by P, the desired solution is among $n!$ possibilities. Assuming that (27) holds, this knowledge provides him with no better method than brute force to find the solution. Thus, in practice, the protocol conveys no extra knowledge for the verifier.

5. Security and Performance Analysis

We have seen that, at each round of the protocol, a dishonest prover successfully cheats with probability $1/2 + 1/m$. Hence, in order to achieve acceptable levels of security, the number of iterations must be sufficiently large. For instance, if $m = 256$, then after 256 rounds a dishonest prover can be caught with probability $1 - 2^{-253}$.

As for the memory space requirements, the prover must store $n(m + \log M)$ bits of secret information, corresponding to the set $\mathcal{A}$, which can be represented with $n\log M$ bits, and the subsets $\mathcal{S}_i$ for $i = 1, \cdots, m$, which can be represented as m strings of n bits. The only public information is the set $\mathcal{B}$, which requires $m\log M$ bits.

During a single round of the protocol, $2\log M(m+n)+n+33$ bits are transmitted (the subsets $\mathcal{A}_0$ and $\mathcal{A}_1$, the pairs $(b_i^{(0)}, b_i^{(1)})$ the bit w and the number k, interpreted as a 32-bit integer, and the subset $\mathcal{S}_k^{(w)}$). The prover must also temporarily store $2mn$ bits corresponding to the subsets $S_i^{(0)}$ and $S_i^{(1)}$, for $i = 1, \cdots, m$.

The parameters $M = 1024$, $n = 256$ and $m = 256$ yield roughly 8.5kB of memory space to store the secret key, 16kB to store temporary information and 1.3kB of transmitted data per round.

6. Final Remarks and Conclusions

We presented a zero-knowledge proof for a special case of the hidden subset sum problem, which can be applied to cryptographic protocols such as authentication systems. The protocol is an adaptation of a zero-knowledge proof for the subset sum problem. The proposal is still not optimal in terms of key sizes and memory requirements, and further research is highly encouraged. Possible optimizations, based on compact representations and variants which require fewer rounds, are part of ongoing research.

Acknowledgements

We would like to thank the anonymous referees for their helpful comments and suggestions.

© 2017 Sociedade Brasileira de Computação

References

Blocki, J. (2009). Direct zero-knowledge proofs. Senior Research Thesis, B.S. in Computer Science, Carnegie Mellon University.

Boyko, V., Peinado, M., and Venkatesan, R. (1998). Speeding up discrete log and factoring based schemes via precomputations. In *Advances in Cryptology - Eurocrypt '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 221–235.

Chor, B. and Rivest, R. L. (1988). A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5):901–909.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition.

Coster, M. J., Joux, A., LaMacchia, B. A., Odlyzko, A. M., Schnorr, C.-P., and Stern, J. (1992). Improved low-density subset sum algorithms. *Computational Complexity*, 2(2):111–128.

El Gamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA. Springer-Verlag New York, Inc.

Goldwasser, S., Micali, S., and Rackoff, C. (1985). The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, pages 291–304, New York, NY, USA. ACM.

Lagarias, J. C. and Odlyzko, A. M. (1985). Solving low-density subset sum problems. *J. Assoc. Comp. Mach.*, 32(1):229–246.

Lenstra, A. K., Lenstra Jr, H. W., and Lovász, L. (1982). Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534.

Merkle, R. C. and Hellman, M. E. (1978). Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24:525–534.

Nguyen, P. and Stern, J. (1999). *The Hardness of the Hidden Subset Sum Problem and Its Cryptographic Implications*, pages 31–46. Springer Berlin Heidelberg, Berlin, Heidelberg.

NIST (1994). FIPS publication 186: Digital signature standard.

Quisquater, Jean-Jacques, G. L. C. and Berson, T. A. (1990). How to explain zero-knowledge protocols to your children. In *Advances in Cryptology*, CRYPTO '89: Proceedings, pages 628–631. ACM.

Schnorr, C. P. (1994). Block reduced lattice bases and successive minima. *Combinatorics, Probability and Computing*, 3:507–522.

T. Okamoto, K. T. and Uchiyama, S. (2000). Quantum public-key cryptosystems. In *Advances in Cryptology: Proceedings of CRYPTO 2000 (M. Bellare, ed.)*, Lecture Notes in Computer Science, pages 147–165, New York. Springer-Verlag.

 © 2017 Sociedade Brasileira de Computação

A Longitudinal and Retrospective Study on How Developers Misuse Cryptography in Online Communities

Alexandre Braga[1,2], **Ricardo Dahab**[2]

[1] Fundação CPqD Centro de Pesquisa e Desenvolvimento em Telecomunicações
R. Dr. Ricardo Benetton Martins, 1.000, Parque II do Polo de Alta Tecnologia
Campinas, SP, Brazil, Zip Code 13086-510

[2]Institute of Computing, State University of Campinas
Av. Albert Einstein, 1251, Cidade Universitária Zeferino Vaz
Campinas, SP, Brazil, Zip Code 13083-852

ambraga@cpqd.com.br,rdahab@ic.unicamp.br

Abstract. *Software developers participating in online communities benefit from quick solutions to technology specific issues and, eventually, get better in troubleshooting technology malfunctioning. In this work, we investigate whether developers who are part of online communities for cryptography programming are getting better in using cryptography with time. This is a crucial issue nowadays, when "real-world crypto" is becoming a topic of serious investigation, not only academically but in security management as a whole: cryptographic programming handled by non-specialists is an important and often invisible source of vulnerabilities [RWC]. We performed a retrospective and longitudinal study, tracking developers' answers about cryptography programming in two online communities. We found that cryptography misuse is not only common in online communities, but also recurrent in developer's discussions, suggesting that developers can learn how to use crypto APIs without actually learning cryptography. In fact, we could not identify significant improvements in cryptography learning in many daily tasks such as avoiding obsolete cryptography. We conclude that the most active users of online communities for cryptography APIs are not learning the tricky details of applied cryptography, a quite worrisome state of affairs.*

Resumo. *Desenvolvedores de software, participantes de comunidades on-line, costumam se beneficiar de soluções rápidas para problemas tecnológicos e, eventualmente, melhoram suas habilidades na resolução de problemas de mau funcionamento da tecnologia. Neste trabalho, investigamos se desenvolvedores de software criptográfico que participam de comunidades on-line se tornam melhores no uso de criptografia com o tempo. Esse é um aspecto de segurança crucial nos dias de hoje, em que "real-world crypto" se tornou um tópico de interesse sério, não só academicamente, mas no gerenciamento de segurança como um todo: programação criptográfica feita por não-especialistas é uma fonte frequente e muitas vezes invisivel de vulnerabilidades [RWC]. Realizamos um estudo retrospectivo e longitudinal para rastrear as respostas dos desenvolvedores sobre programação de criptografia em duas comunidades on-line. Descobrimos que o uso indevido da criptografia é não apenas comum em comunidades on-line, mas também é recorrente nas discussões, sugerindo que os desenvolvedores aprendem a usar as APIs criptográficas sem realmente aprender criptografia. Não conseguimos identificar melhora alguma na percepção e aprendizado de vulnerabilidades criptográficas, mesmo em tarefas simples como a de evitar o uso de criptografia obsoleta. Concluimos, assim, que os usuários ativos nas comunidades on-line para APIs criptográficas não tem evoluído no seu aprendizado dos detalhes e armadilhas do uso da criptograpfia, um estado de coisas muito preocupante.*

 © 2017 Sociedade Brasileira de Computação

1. Introduction

Software developers are regular users of security mechanisms (e.g., security APIs, protocols, and tools), but are, by no means, security experts. They, however, do make security decisions that have a huge impact on end-user and system security. Developers are also frequent users of online communities for programming. The agility in problem solving provided by many question-and-answer communities brings benefits to ordinary programmers lacking knowledge in specific topics, such as secure coding. In this work, we investigate whether developers participating in online communities for cryptography programming are getting better in using cryptography with time. Also, we investigate whether cryptography misuse is persistent in posts of specific developers.

Cryptography misuse is a programming bad practice frequently found in misuse cases of cryptographic software, ultimately leading to vulnerabilities, but also associated to design flaws and insecure architectural choices [Braga and Dahab 2016]. Improvement in cryptography knowledge can be evidenced by a time series showing a steady decrease in the number of cryptography misuses. Also, persistence of a specific cryptography misuse can be illustrated by the repetition by developers of the same misuse over and over.

We tracked users (i.e. developers) of two online communities for programming, along with their questions and answers, from posts in a data set provided by a previous study [Braga and Dahab 2016]. We recorded the occurrence of known cryptography misuses for selected developers over a period of five years. We then computed statistics of cryptography misuse associated to specific moments in time for each tracked developer.

We found that the use of weak cryptography (e.g., broken algorithms or misconfigured implementations of standards) is not only common in online communities, but also recurrent in developers' discussions, suggesting that they learn how to use crypto APIs without actually learning cryptography. We also found that the lack of knowledge in cryptography is a recurrent source of coding bugs in API usage and does not depend on how long developers use cryptography APIs. We observed that platform issues dominate design concerns and obfuscate many complex cryptography misuses, which go unnoticed by developers in long lifespans. In summary, we conclude that users of online communities are not actually learning cryptography, despite their immediate gains in solving current programming issues related to cryptographic APIs.

This study is longitudinal, i.e., it performed repeated observations of the same developers over a period of time. Also, it is retrospective because it looks back in time using existing data. As far as we know, this is the first such study of cryptography misuse in online communities. The main contributions of this work are the following: *(i)* a method for clustering developers' posts from their asynchronous lifespans in online communities; *(ii)* a longitudinal study of selected developers of two communities, showing similarities and differences of these communities concerning how developers misuse cryptography; *(iii)* evidence that cryptography misuse is persistent across communities and developer lifespans; and *(iv)* evidence that developers learn how to use cryptography APIs without learning cryptography.

This text is organized as follows. Section 2 analyses related work and Section 3 details our research method. Section 4 explains our results and findings, while Section 5 details two users' lifespans. Section 6 discusses our findings and Section 7 shows our conclusions.

2. Related Work

[Fahl et al. 2012] investigated the SSL/TLS protocol usage in Android apps from Google Play and discovered security threats posed by misuses of that protocol. [Egele et al. 2013] were among the first to perform large-scale experiments in Google Play App Store to measure cryptographic

 © 2017 Sociedade Brasileira de Computação

misuse in Android with the standard Crypto API. Their main contribution was a broad view of the prevalence of misused cryptographic functionality in Android apps. These two pioneering works were followed by others on related topics (e.g., [Lazar et al. 2014, Shuai et al. 2014, Georgiev et al. 2012, Chatzikonstantinou et al. 2015]).

[Wang and Godfrey 2013] were among the first to analyze API-related posts from a questions-and-answers (Q&A) website for mobile app development. They discovered repetitive scenarios with obstacles in API usage to developers, not specifically related to security. In a recent work [Wang et al. 2015], they investigated methods and proposed a methodology to distill and rank Q&A posts with API-related issues that would be valuable to API designers.

[Nadi et al. 2016] performed an empirical investigation into the obstacles developers face while using the Java cryptography APIs and the programming tasks they perform (e.g., authenticate users, store login data, establish secure connections, and encrypt data). By triangulating data from Stack Overflow posts, GitHub repositories, and developers' surveys, they found that developers find it difficult to use cryptographic algorithms correctly, despite being confident with cryptography concepts. They also found that cryptographic APIs are generally perceived as too low-level and not task-oriented.

[Acar et al. 2016b, Acar et al. 2016a] systematically analyzed the impact to code security of information resources commonly used by developers. They surveyed app developers who have published in the Google Play market, conducted a lab study with Android developers, analyzed 139 Stack Overflow threads accessed by developers during the lab study, and statically analyzed a random sample of Google Play apps. They concluded that real-world developers use Q&A communities as a major resource for solving programming problems, including security problems, suggesting that those online communities help developers to arrive at functional solutions more quickly than other resources. However, because online communities contain many insecure answers, developers who rely on this resource are likely to create less secure code. Also, access to quick solutions via a Q&A community may also inhibit developers' security thinking or reduce their focus on security.

[Braga and Dahab 2016] performed a transversal study to analyze how developers misuse cryptography in two online communities: Oracle Java Cryptography (OJC) and Google Android Developers (GAD). That work showed not only the most frequent cryptography misuses (e.g., weak cryptography, coding bugs, etc.), but also relationships among misuses through strong associations of double or triple misuses that appear together with non-negligible probabilities.

Most of the above-mentioned studies focus on the same online community or app store, with little variation. Also, none of these works study the behavior of frequent users over time, in order to examine whether developers are getting better in using cryptography with time.

3. Methodology

We analyzed data collected by a previous study [Braga and Dahab 2016] and observed that repeated measures were made for some developers. This fact motivated us to perform a retrospective, longitudinal study to analyze developers' behavior from a series of observations already made about them. This study is longitudinal because it performed repeated observations of the same developers (and their posts) over a period of time. Also, it is retrospective because it looks back in time using existing data.

Roughly speaking, our method segments the set of posts (collected for specific developers with determined lifespans) into a predefined number of clusters. When ordered chronologically, these clusters determine the phases a developer is supposed to pass for learning cryptography. These phases are then analyzed for the occurrence of cryptography misuses which are well-known in secure software development and secure coding.

© 2017 Sociedade Brasileira de Computação

The following subsections detail our methodology in four topics: classification of cryptography misuse, selection of communities and posts, selection of developers to evaluate, and method for clustering posts from a developer's lifespan.

3.1. Classification of Cryptography Misuse

The original study [Braga and Dahab 2016] introduced a classification of cryptography misuses in order to capture how software developers actually misuse cryptography in practice. The classification has nine categories: Weak Cryptography (WC), Bad Randomness (BR), Coding and Implementation Bugs (CIB), Program Design Flaws (PDF), Improper Certificate Validation (ICV), Public-Key Cryptography (PKC) issues, Poor Key Management (PKM), Cryptography Architecture Issues (CAI), and IV/Nonce Management (IVM) issues. Table 1 details categories in descriptive subsets.

The classification collected cryptography misuse from various sources, including software security books (e.g., [Viega and McGraw 2001, Howard and LeBlanc 2003, Chess and West 2007, Howard et al. 2009, Howard and Lipner 2006, Shostack 2014]), studies on cryptography misuse (e.g, [Lazar et al. 2014, Chatzikonstantinou et al. 2015, Egele et al. 2013, Shuai et al. 2014, Braga and Dahab 2015, Georgiev et al. 2012, Fahl et al. 2012]), newly discovered misuses (e.g., [Alashwali 2013, Bos et al. 2014, Mart and Hern 2013, Adrian et al. 2015]), and industry initiatives for software security (e.g., [Safecode 2011, OWASP , CYBSI 2014]). Table 1 shows the grouping of misuse categories, misuse main categories, and subsets.

Cryptography misuses are not all equally difficult to avoid [Braga and Dahab 2016]: some are easier to find and correct than others, depending on the involved complexity to identify and fix misuses. There are three complexity groupings for the nine misuse categories (in Table 1):

1. **Low complexity** misuses are related to coding activities and issues in APIs, and could be easily found by simple code reviews and skilled developers (supported by tools). This group includes Weak Crypto (WC), Coding Bugs (CIB), and Bad Randomness (BR).
2. **Medium complexity** misuses are related to flaws in program design affecting a few programs and may be difficult to identify due to feature distribution across programs. This group includes Improper Certificate Validation (ICV) issues, Program Design Flaws (PDF), and Public-Key Crypto (PKC) issues.
3. **High complexity** is related to flaws in system design and architecture, and requires understanding of system architecture to analyze underlying cryptosystems. This group includes Poor Key Management (PKM), IV and Nonce Management (IVM) issues, and Crypto Architecture Issues (CAI).

3.2. Selection of Communities and Posts

The original study [Braga and Dahab 2016] selected two programming communities possibly supported by experts in applied cryptography: Oracle Java Cryptography (OJC) [OJC], a forum aimed at programming with Java Cryptographic Architecture (JCA), and Google Android Developers (GAD) [GAD], a forum for Android programming.

The reasons to choose these two communities follows. Both OJC and GAD share the same Java-based API for the Java Cryptographic Architecture (JCA) [Oracle], thus limiting the knowledge required by a code reviewer to four aspects: Java programming, JCA, Android security, and applied cryptography. Also, JCA offers a stable and generic API, which has been used **for a long time** by a large number of developers for both server-side applications and mobile devices. Furthermore, JCA was adopted by the Android platform as its main API for cryptographic services. These two communities together reach a large number of ordinary developers, most

 © 2017 Sociedade Brasileira de Computação

Table 1. Classification of cryptography misuse from a developer's viewpoint.

Low complexity		Medium complexity		High complexity	
Cat.	**Misuse subtype**	**Cat.**	**Misuse subtype**	**Cat.**	**Misuse subtype**
WC	-Risky/broken crypto -Proprietary crypto -Determin. symm. enc. -Risky/broken hash -Risky/broken MAC -Custom implement.	PDF	-Insec. default behavior -Insecure key handling -Streamcipher:insec. use -Insecure combo encr./auth -Insecure combo encr./hash -Side-channel attacks	IVM	-CBC w/ non-random IV -CTR with static counter -Hard-coded or const. IV -Reused nonce in encrypt.
CIB	-Wrong configs for PBE -Common coding errors -Buggy IV generation -Null cryptography -Leak/Print of keys	ICV	-No validation of certs -Broken SSL/TLS channel -Incomplete cert. valid. -Improper valid. host/user -Wildcards certs -Self-signed certs	PKM	-Short/improper key size -Hard-coded/const. keys -Hard-coded PBE passw. -Streamcipher:reused key -Use of expired keys -Key distrib. issues
BR	-Use of Statistic PRNG -Predictible seeds -Low-entropy seeds -Static, fixed seeds -Reused seeds	PKC	-Determ. encryp. RSA -Insec. padding RSA enc. -Weak configs RSA enc. -Insec. padding RSA sign. -Weak RSA sign. -Weak ECDSA sign -Key agr.: DH/ECDH -ECC: insecure curves	CAI	-Crypto agility issues -API misunderstanding -Multiple access points -Randomness issues -PKI and CA issues

of them are supposed to be non experts in cryptography. These assumptions may not hold for specialized communities with other APIs, such as openssl [OpenSSL] or bouncy castle [BC].

Collected posts comprised a time period of five years, from January 2011 to December 2015. Posts were listed by date (newest first) and manually saved as PDF files.

OJC was the most active community, with the most posts in the selected time period. GAD is very active in general, but showed less activity for cryptographic matters. For OJC, 310 posts were collected, and the 155 most viewed were selected for further analysis (50% of total). In GAD, a pre-analysis showed that specific keywords, such as "encryption", "hash" and "sign", were covered by the more general keywords "cryptography" and "encryption", which were used to select posts. For GAD, 170 posts were collected and the 100 most viewed were selected for analysis.

The manual inspection with code review was the method to analyze each single post. Posts were inspected by a cryptography expert with the skills mentioned above. Each post was inspected for occurrences of misuse. Many posts were discarded for not being related to cryptography programming, showing only discussions about threats or attacks. After discarding, OJC data set was reduced to 140 posts and GAD achieved 71 posts.

A few topics related to environment and platform specific issues were identified: configuration and installation issues, key storage and recovery issues, bug found or reported, tool misuse or misunderstanding, interoperation issues (e.g., platforms, versions, etc.), and hardware integration issues.

3.3. Selection of Developers to Evaluate

We found that most developers just ask one question to the community and never return. On the other hand, a few developers answered most questions. This fact made it possible to track users' answers and determine whether they had learned cryptography with time. These developers not only failed in giving good answers to questions related to cryptography; sometimes, they also omitted information that could prevent cryptography misuse.

In OJC, we counted 43 distinct developers who answered at least one question. Only 8 of these

© 2017 Sociedade Brasileira de Computação

developers answered 3 or more questions, corresponding to 74.5% of all answers. One developer asked 11 questions, totalling 9 evaluated developers in OJC. In GAD, we counted 97 distinct developers who answered at least one question. Only 14 of them answered 4 or more questions, corresponding to 52% of all answers. One developer asked 6 questions, totalling 15 evaluated developers in GAD. In summary, this study was conducted for two cohorts: one for a cryptography-specific forum (OJC) with 11 subjects and the other for a general-purpose forum (GAD) with 15 subjects.

The average lifespan for OJC developers was about 22 months with a standard deviation of around 9 months. The average lifespan for GAD developers was about 20 months with a standard deviation around 11 months. Table 2 shows lifespans and number of posts for both OJC and GAD. OJC developers are identified by J# and GAD developers are identified by G#.

Table 2. Selected developers, their lifespans (in months) and number of posts.

OJC Developers			GAD Developers					
OJC#	**Lifespan**	**# of posts**	**GAD#**	**Lifespan**	**# of posts**	**GAD#**	**Lifespan**	**# of posts**
J#1	8.4	7	**G#1**	14.4	6	**G#10**	8.1	6
J#2	30.5	14	**G#2**	22.5	6	**G#11**	10.9	4
J#3	26.4	33	**G#3**	25.1	6	**G#12**	17.2	4
J#4	19.9	6	**G#4**	17.1	5	**G#13**	7.3	4
J#5	27.1	13	**G#5**	36.0	8	**G#14**	7.1	5
J#6	29.6	36	**G#6**	22.3	11	**G#15**	11.9	5
J#7	31.1	11	**G#7**	28.8	13	**-**	-	-
J#8	8.3	5	**G#8**	45.4	29	**-**	-	-
J#9	16.1	3	**G#9**	23.9	11	**-**	-	-

3.4. Method for Clustering Developer's Lifespan

In online communities, interaction among users has a chronological order, but does not have to follow simultaneous events for synchronization of activities. For instance, a developer can show a very active participation for a few months and never return, while another can have a consistent participation for a couple of years.

We noticed that users participate in communities within different lifespans, diverse in length (duration) and number of posts. A user's lifespan is counted from the first to the last participation found in the period of study. In order to capture developers' distinct lifespans within the studied time period, we adopted a simple clustering technique to split the activity in each developer's lifespan (e.g., all posts for a user) into a defined number of clusters, as described next.

Clustering is a technique for combining observed objects into groups, segments, or clusters [Murthy 2015]. Its goal is to partition the observations into groups ("clusters") so that the differences among elements assigned to the same cluster tend to be smaller than among elements in different clusters [Friedman et al. 2009].

Clustering results need to be tied to specific semantic interpretations and applications [Murthy 2015]. Therefore, it is important to utilize expert knowledge to identify clusters [Murthy 2015]. We observed a natural fit between clustering methods and the life cycle presented next.

We devised a method to normalize lifespans and compare them. Our method consists in associating a life cycle to a lifespan. A life cycle is a qualitative sequence of phases. Lifespans use absolute time scales and are quantitative, while life cycles are relative and subjective, being qualitative in nature. A life cycle is divided into five phases according to the progress the user in evaluation is supposed to have had in his lifespan. The appropriate number of phases was apparent from prior knowledge about the data set. The five phases are the following:

 © 2017 Sociedade Brasileira de Computação

1. an **entrant** or newbie is a new member or an inexperienced newcomer;
2. a beginner or **novice** is starting to learn crypto and taking part in communities;
3. an accustomed **fellow** is a regular user involved in the same activities of others;
4. an **expert** has knowledge or skills in cryptography gained over a period of time;
5. a **veteran** has long experience in cryptography and is considered knowledgeable.

This is an optimistic life cycle because it supposes developers improve their skills in cryptography with time. In fact, this is a strong assumption that may not hold for most developers.

The general idea of our clustering method is to apply a divisive clustering technique in order to distribute posts through five phases according to that user lifespan. The segmentation starts with all posts in one cluster (the whole lifespan) and iteratively splits existing clusters into two smaller clusters, until it satisfies a termination condition, e.g. the desired number of clusters. Roughly speaking, the lifespan is the bigger cluster, which we divide in five slices (life-cycle phases) in three iterations of the clustering method.

The distribution of posts through phases (i.e. cluster assignments, e.g. the assignment of posts to clusters) were refined by repeatedly attempting subdivision, and keeping the best resulting splits. In our method, a short lifespan lasts less than one year and a long lifespan lasts more than one year. We managed to distribute posts uniformly whenever it is possible, according to the following rules:

1. when there were less than five posts in the lifespan, we managed to put two at the ends and, optionally, two (or one) near the middle;
2. posts within the same month (a small time lapse) were put in the same phase (cluster);
3. when there was a large time lapse (bigger than 1 or 2 months in short lifespans or bigger than 4 to 6 months in long lifespans) between two consecutive posts, we adopted this gap as a phase separator (split point).

In general, divisive clustering methods, as the one we adopted, encounter difficulties regarding the selection of split points, leading to low-quality clusters [Murthy 2015]. We achieved a balanced distribution in general and a similar amount of posts into clusters, for both communities, with slightly more posts found at middle phases (Novice, Fellow, and Expert), and fewer posts found in first and last phases (Entrant and Veteran). Figure 1 shows the distribution of posts throughout the five phases for OJC and GAD.

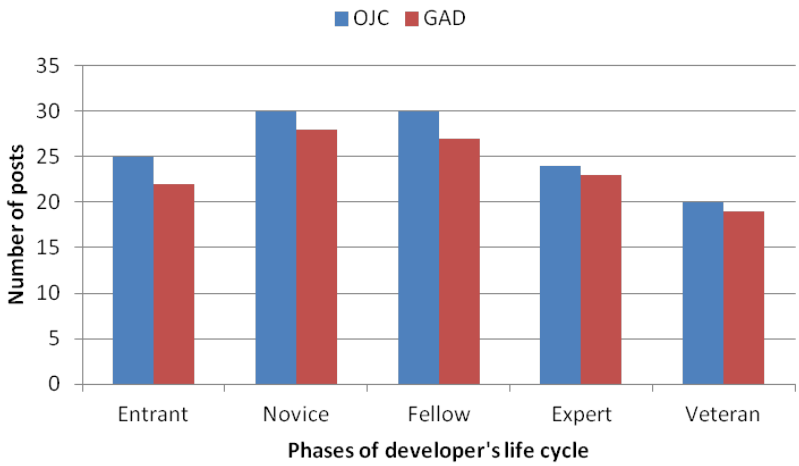


Figure 1. Distribution of posts through life cycle phases.

© 2017 Sociedade Brasileira de Computação

4. Results and Findings

This section analyses results for both communities in two distinct measures: misuse count and misuse density. Also, misuse density is analyzed in the context of the complexity groupings in Table 1. Before, we show general statistics about cryptography misuse in Table 3, that uses data from [Braga and Dahab 2016].

Table 3. Crypto misuse in online communities, from [Braga and Dahab 2016].

Categories of Cryptography Misuse	Communities	
	OJC	GAD
Weak Cryptography (WC)	26%	21%
Coding and Implementation Bugs (CIB)	17%	17%
Bad Randomness (BR)	0%	1%
Program Design Flaws (PDF)	6%	8%
Improper Certificate Validation (ICV)	4%	3%
Public-Key Cryptography (PKC) issues	16%	10%
Poor Key Management (PKM)	11%	4%
IV/Nonce Management (IVM) issues	5%	6%
Crypto Architecture Issues (CAI)	20%	1%
Platform Specific Issues (PSI)	60%	32%

The occurrence of cryptography misuse for each community is summarized in Table 3, which shows that weak cryptography (WC) is the most common misuse in both OJC and GAD. Also, both communities suffer negative influence from platform-specific issues (PSI). Besides specific issues, OJC suffers the most influence from weak cryptography (WC, 26%), architectural issues (CAI, 20%), coding bugs (CIB, 17%), public-key issues (PKC, 16%), and poor key management (PKM, 11%). These numbers are due to API misuse, lack of knowledge in applied cryptography, and complexity of JCA.

GAD suffers most from weak cryptography (WC, 21%), coding bugs (CIB, 17%), and public-key issues (PKC, 10%). These numbers are due to API misuse and lack of knowledge in cryptography programming. Despite preserving the Java API, Android has its own architecture for enabling cryptographic libraries, which simplifies installation and configuration, but brings new interoperational issues. Also, misuse of Pseudo-Random Number Generators (PRNGs) was barely mentioned in both communities, suggesting that developers have no doubts about simple uses of Java's SecureRandom API.

4.1. Cryptography Misuse Count per Life Cycle Phase

Figure 2 shows misuse counts for OJC (left) and GAD (right), for all misuse categories, distributed along the five phases of developer's life cycle. These communities have distinct behavior.

In OJC (left), developers start with a shy participation with relatively few misuses in Entrant phase. It is possible to observe that Novice is the phase with most misuses, with a notable presence of categories CAI, PKC, and WC. Also, OJC developers seem to be improving their skills in cryptography, because the total count of misuses gradually decreases from Novice to Veteran phase. However, the numbers for simple misuses (e.g., WC and CIB) are relatively stable (not decreasing), suggesting that simple misuses are recurrent and developers are not getting better at them. PKM and PKC are common issues in early phases (Novice and Fellow), with higher values, suggesting that developers are improving in these categories. Also, the number of CAI issues decreases from Novice to Veteran, suggesting that knowledge about Java's crypto API increases with time.

In GAD (right side), developers start in the Entrant phase with an expressive participation, having most misuses in WC and CIB. The notable decrease in crypto misuse for Novice and

 © 2017 Sociedade Brasileira de Computação

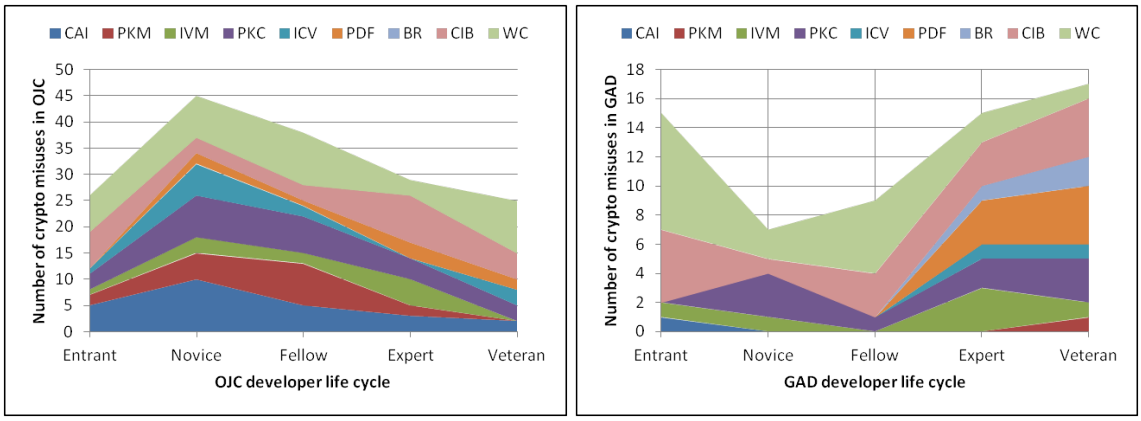


Figure 2. Crypto misuse counts for life cycle phases: OJC (left) and GAD (right).

Fellow phases suggests developers have a fast learning curve for those misuses less influenced by platform specific issues. However, in the Expert and Veteran phases the number of misuses increases again due to the influence of platform issues associated to sophisticate and complex misuses (e.g., PDF, PKC, ICV, and IVM). Four misuse categories (WC, CIB, PKC, and IVM) are recurrent in all phases, suggesting Android's diversity of hardware and software makes it difficult to learn cryptography not only for simple misuses, but also for complex misuses.

4.2. Cryptography Misuse Density per Life Cycle Phase

In order to measure crypto misuse density, we adapted the traditional metric for issue density per unit of size, which is also named defect density [Pandian 2003], bug density [Hutcheson 2003], and fault density [Bourque and Fairley 2014]. In order to measure how developers misuse cryptography, we counted the number of misuses that have been detected in posts for a developer and normalized this measure by the total number of posts for the developer in question, obtaining a value for misuse density. A straightforward analysis on misuse density over time was used to evidence a learning curve for developers as well as to show misuse reduction (or growing) as a trend.

Figure 3 shows the misuse density for OJC throughout the life cycle in two charts. On the left, crypto misuse per post (c.m.p.p) is compared to platform issues per post (p.i.p.p). In this chart, it is possible to observe that misuse density is relatively stable over time, despite a gradual reduction in density for platform issues. The chart on the right shows misuse density for low-, medium- and high-complexity misuses. In this chart, it is possible to observe that density of simple misuses (WC, CIB, and BR) increases over time, while density of moderate (PDF, PKC, and ICV) misuses has a small decrease, and density for high-complexity misuses (CAI, PKM, and IVM) shows a gradual decrease.

This behavior suggests that simple misuses are recurrent in OJC and do not depend on the actual knowledge of Java's crypto API. On the other hand, medium-complexity misuses are the most influenced by platform issues, closely following p.i.p.p behavior. Then, complex misuses (related to system design and architecture) decreases over time, suggesting a gradual improvement in developer's knowledge about Java's crypto API.

Similarly, Figure 4 shows the misuse density for GAD throughout the life cycle in two charts. On the left, it is possible to observe that misuse density increases over time (the opposite behavior of OJC), despite a relatively stable density of platform issues. The chart on the right shows that density for complex misuses is relatively stable, while density for low- and medium-complexity misuses grows over time. In fact, GAD developers start with a high density for simple misuses and,

© 2017 Sociedade Brasileira de Computação

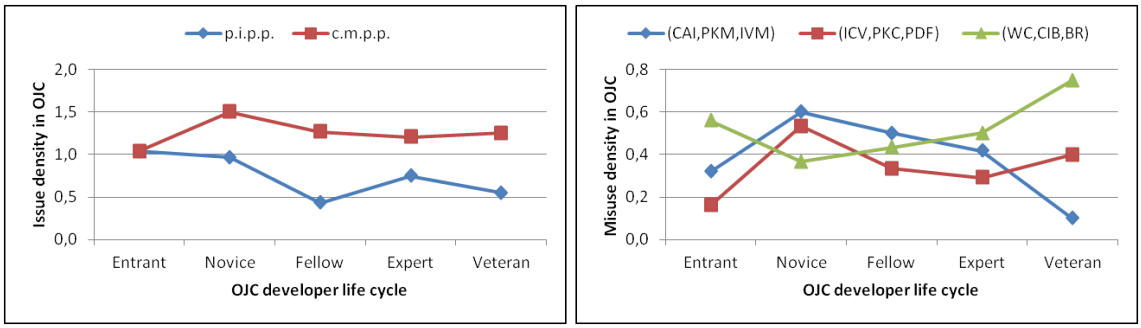


Figure 3. Misuse density in OJC compared to issue density and complexity groups.

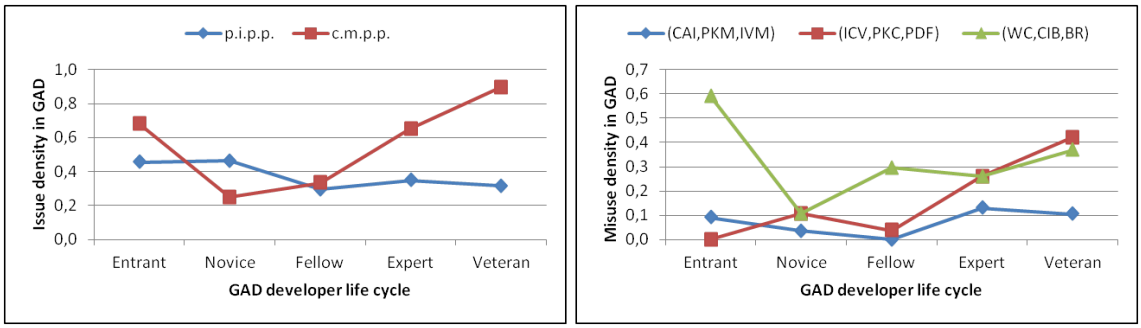


Figure 4. Misuse density in GAD compared to issue density and complexity groups.

after a sharp drop, this density gradually grows. This behavior suggests that Android developers are not getting better in cryptography over time.

5. Two Life Cycles in Detail

This section details the life cycles of two developers. The objective is to illustrate with real cases the recurrence of crypto misuse in developers' lifespans. We selected two users with more posts and longer lifespans, being good representatives of their communities: J#6 from OJC with a lifespan of 29.6 months and 36 posts, and G#8 from GAD with a lifespan of 45.4 months and 29 posts. These developers not only failed to give good answers, but sometimes omitted information that could prevent crypto misuse. Also, we could not identify improved cryptography skills for these developers.

5.1. Life Cycle for a Java Developer (J#6)

In the entrant phase, J#6 contributed to posts related to incompatible crypto providers, buggy hardware modules, cross-platform verification of certificates, and cross-language (e.g., from C++ to Java) encryption. Misuses were associated to hard-coded keys and IVs, wrong ciphertext encoding, weak cryptography with custom implementations, short keys, misconfigured RSA, and deterministic encryption with RSA.

In the novice phase, J#6 experienced misuses associated to deterministic symmetric encryption (AES/ECB), broken cryptography (DES), proprietary cryptography with custom key agreement, and coding errors. Other misuses were associated to unsafe defaults, insecure padding for RSA, deterministic encryption or short keys for RSA, and issues for DH and ECC. Complex misuses were associated to insufficient length and key distribution issues, as well as IV misconfiguration and design flaws in cryptographic architecture.

In the fellow phase, J#6 contributed to posts related to fails in TLS authentication, RSA encryption, digital signature verification, and proprietary encryption. Misuses were associated

© 2017 Sociedade Brasileira de Computação

to buggy PKI software, proprietary cryptography, risky cryptography (3DES in ECB), coding errors in insecure key handling, insecure padding or inadequate key length for RSA.

In the expert phase, J#6 still talked about coding errors when using AES with password-based encryption, hard-coded IVs and deterministic encryption, misunderstanding digital certification, deterministic signatures with RSA, and insecure key derivation. Platform issues led to misunderstand of PKI functions. Other misuses were associated to flawed IV generation, broken hash function, unsafe default, insecure padding for RSA, non-random or constant IVs, and reuse of keys with stream ciphers.

In the veteran phase, J#6 could not give correct answers to posts related to MAC with broken hash (e.g., MD5), encoding of keys, issues in key generation, and misconfigured PKI software. Other misuses were associated to public-key issues (insecure padding for RSA, and misconfigured DH), unsafe defaults, improper certificate validation (non-validated hostname and self-signed certificates), coding errors disabling cryptography, and deterministic symmetric encryption.

5.2. Life Cycle for an Android Developer (G#8)

In the entrant phase, G#8 was involved in several discussions about password-based encryption, errors when decrypting data from strings, use of SHA1 to generate keys from passwords, use of AES in CBC mode to encrypt files, and Android's full encryption. Misuses were related to broken encryption and hashes, misconfigured PBE, insecure deterministic encryption, ciphertext encoding errors, custom implementation of PBE, and constant IVs.

In the novice phase, G#8 was involved in posts related to buggy implementations of ECC in SSLv2 and signature verification on crypto libraries, proprietary implementation of SSL, and use of RSA encryption. Several misuses were associated to insecure padding and deterministic encryption for RSA, custom implementation of SSL, and attempts to use buggy implementations of ECC.

In the fellow phase, G#8 discussed cryptography adopted by Google Drive, errors when using the wrap method for protection of keys with PBE in specific versions of Android, and cross-platform verification of signatures (Java and dotNET). Misuses involved misconfigured PBE with small parameters, use of risky hashes and broken encryption, errors in ciphertext encodings, and insecure padding and deterministic encryption for RSA.

In the expert phase, G#8 was involved in discussions about several errors related to bad padding in encryption with AES, parsing keys from certificates for RSA encryption, and backward incompatibility of encryption algorithms in Android. Misuses associated to improper certificate validation, insecure defaults, deterministic encryption with RSA, non-random or constant IVs, and misconfigured PBE.

In the veteran phase, G#8 was involved in posts related to cryptography issues in Android, such as storage and recovery of keys from the device's keystore, cross-version decryption of files, and encoding ciphertext as integers. Misuses were associated to improper certificate validation with self-signed root certificates, misconfigured PBE with small parameters, insecure defaults for AES, non-random IVs, and ciphertext encoding errors.

6. Discussion

We are aware that our analysis have to be put in context and is restricted to the main subject of the two communities evaluated. That said, we tried to generalize our conclusions.

For developers, as much as for end-users, security is a secondary concern. Developers usually have priorities (e.g., functional correctness, time to market, maintainability, economics, compliance

© 2017 Sociedade Brasileira de Computação

with other corporate policies) that often appear to conflict with security. Frequently, developers look for quick, but insecure solutions and online communities favor this behavior.

Ideally, developers should not be forced to learn cryptography in order to correctly use cryptographic APIs, specially for simple use cases. However, in practice, crypto APIs are unable to foster their correct use without domain knowledge obtained from elsewhere but online communities.

Java has a stable API, with a very predictable behavior, favoring developers with enough time to understand its particularities. In general, developers improve their skills in misuse categories affected by platform issues, but this does not happen for simple misuses. On the other hand, Android uses the same crypto API of the Java platform, but this fact alone is not enough to promote a positive learning curve for cryptography. Many issues related to diversity of both hardware and software negatively affect how developers learn cryptography in Android.

Java cryptographic architecture presents issues with installation and configuration that divert developers from actual tasks of cryptographic programming. Also, specific issues showed up when integrating Java programs to cryptographic hardware or communicating with applications in other platforms. These troublesome issues frequently obfuscate crypto misuses in the same code. For instance, in Java, a worst-case scenario occurs when developers write buggy code for encrypting data and use weak cryptography by accidentally adopting insecure defaults.

Android solved many issues faced by Java developers, but brought to daily troubleshooting several interoperation issues due to the diversity of both hardware and software in that platform. Developers, confused by these issues, are distracted from actual cryptographic pitfalls. For instance, developers suffering from interoperation issues among devices used weak cryptography to protect stored passwords, and derived keys directly from password hashes.

Developers learn how to make APIs work, but this does not mean cryptography was used correctly. In fact, coding bugs are persistent issues when using general-purpose (function-based) crypto APIs to implement application-specific use cases, because developers are forced to make insecure choices without actually understanding the whole situation. This suggests developers would benefit from high-level cryptographic frameworks (oriented toward use cases) or task-based APIs that could avoid simple misuses and insecure design decisions.

The overabundance of complex options for security leads to disengagement when confronted by other concerns. We have noticed that complex architectures distract developers from actual cryptographic misuse and contribute to perpetuate issues in cryptographic programming. For instance, one curious reason developers gave to use homemade code for cryptographic algorithms is to avoid dependencies to external libraries.

Finally, we did not see in developers' lifespans any posts concerned with the insecure combination of encryption and authentication, padding-oracle attacks, or selection of insecure elliptic curves. This suggests that these developers never learned about design flaws for authenticated encryption, side-channel attacks or obsolete implementations for elliptic curve cryptography. Another remarkable absence in lifespans was the concern with weak parameters in public-key cryptography (e.g., RSA, DH), suggesting that developers always take for granted the quality of parameters generated by tools.

7. Concluding Remarks

Ordinary software developers are used to obtain quick solutions to daily problems from fellows in online communities. Those communities associated to cryptographic programming are good for finding solutions to platform-specific issues (e.g., implementation bugs, incompatible hardware, and misconfigured software) as well as to clarify obscured aspects of API usage. On the other

 © 2017 Sociedade Brasileira de Computação

hand, developers willing to learn cryptography from online communities, if lucky, receive only shallow advice and usually do not have a positive learning curve over time. Our study shows that cryptography misuse is perpetuated in online communities and frequently reappear in recurrent issues, because these communities favor quick, but insecure solutions and even active developers (of those communities) take security as a secondary concern.

We believe this longitudinal study effectively contributes to better understanding how cryptography is handled by ordinary developers of two online communities, bringing to light their attitudes and priorities concerning cryptography-based security over time. It is paramount to improve APIs to increase usability and to foster best practices. Also, there are opportunities for future work in behavioral experiments of cryptographic programming, surveys with actual developers, as well as replication studies focusing on other communities and crypto APIs.

Acknowledgements

We thank Intel and CNPq for the financial support, as well as CPqD and UNICAMP for the institutional support.

References

Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., and Stransky, C. (2016a). You Get Where You're Looking For: The Impact Of Information Sources on Code Security. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 289–305. IEEE.

Acar, Y., Fahl, S., and Mazurek, M. L. (2016b). You Are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users.

Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., and Others (2015). Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM.

Alashwali, E. S. (2013). Cryptographic vulnerabilities in real-life web servers. In *Third International Conference on Communications and Information Technology (ICCIT)*, pages 6–11. Ieee.

BC. The Legion of the Bouncy Castle.

Bos, J. W., Halderman, J. A., Heninger, N., Moore, J., Naehrig, M., and Wustrow, E. (2014). Elliptic curve cryptography in practice. In *Financial Cryptography and Data Security*, pages 157–175. Springer.

Bourque, P. and Fairley, R., editors (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE Computer Society, version 3. edition.

Braga, A. and Dahab, R. (2015). Introdução à Criptografia para Programadores: Evitando Maus Usos da Criptografia em Sistemas de Software. In *Caderno de minicursos do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2015*, pages 1–50.

Braga, A. and Dahab, R. (2016). Mining Cryptography Misuse in Online Forums. In *2nd International Workshop on Human and Social Aspect of Software Quality*.

Chatzikonstantinou, A., Ntantogian, C., Xenakis, C., and Karopoulos, G. (2015). Evaluation of Cryptography Usage in Android Applications. *9th EAI International Conference on Bio-inspired Information and Communications Technologies*.

Chess, B. and West, J. (2007). *Secure programming with static analysis*.

CYBSI (2014). Avoiding The Top 10 Software Security Design Flaws.

Egele, M., Brumley, D., Fratantonio, Y., and Kruegel, C. (2013). An empirical study of cryptographic misuse in android applications. *ACM SIGSAC conference on Computer & comm. security (CCS'13)*, pages 73–84.

© 2017 Sociedade Brasileira de Computação

Fahl, S., Harbach, M., and Muders, T. (2012). Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *ACM conference on Computer and communications security*, pages 50–61.

Friedman, J., Hastie, T., and Tibshirani, R. (2009). *The elements of statistical learning*, volume 2. Springer-Verlag.

GAD. Google Android Developers.

Georgiev, M., Iyengar, S., and Jana, S. (2012). The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, pages 38–49.

Howard, M. and LeBlanc, D. (2003). *Writing secure code*.

Howard, M., LeBlanc, D., and Viega, J. (2009). *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. McGraw-Hill Education.

Howard, M. and Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA.

Hutcheson, M. L. (2003). *Software testing fundamentals: methods and metrics*. John Wiley & Sons.

Lazar, D., Chen, H., Wang, X., and Zeldovich, N. (2014). Why Does Cryptographic Software Fail?: A Case Study and Open Problems. In *5th Asia-Pacific Workshop on Systems*, APSys '14, pages 7:1—–7:7, New York, NY, USA. ACM.

Mart, V. G. and Hern, L. (2013). Implementing ECC with Java Standard Edition 7. *International Journal of Computer Science and Artificial Intelligence*, 3(4):134–142.

Murthy, M. R. (2015). *Introduction to Data Mining and Soft Computing Techniques*. Laxmi Publications.

Nadi, S., Krüger, S., Mezini, M., and Bodden, E. (2016). "Jumping Through Hoops": Why do Java Developers Struggle With Cryptography APIs? *The 38th International Conference on Software Engineering*.

OJC. Oracle Java Cryptography.

OpenSSL. OpenSSL Cryptography and SSL/TLS toolkit.

Oracle. Java Cryptography Architecture (JCA) Reference Guide.

OWASP. Cryptographic Storage Cheat Sheet.

Pandian, C. R. (2003). *Software metrics: A guide to planning, analysis, and application*. CRC Press.

RWC. Real World Crypto Symposium.

Safecode (2011). Fundamental Practices for Secure Software Development.

Shostack, A. (2014). *Threat modeling: Designing for security*. John Wiley & Sons.

Shuai, S., Guowei, D., Tao, G., Tianchang, Y., and Chenjie, S. (2014). Modelling Analysis and Auto-detection of Cryptographic Misuse in Android Applications. In *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 75–80.

Viega, J. and McGraw, G. (2001). *Building Secure Software: How to Avoid Security Problems the Right Way*.

Wang, W. and Godfrey, M. W. (2013). Detecting API Usage Obstacles : A Study of iOS and Android Developer Questions. pages 61–64.

Wang, W., Malik, H., and Godfrey, M. W. (2015). Recommending posts concerning api issues in developer q&a sites. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 224–234. IEEE Press.

 © 2017 Sociedade Brasileira de Computação

Processador de Uso Específico para o Algoritmo SHA-1

Carlos E. B. Santos Júnior [1], **Marcelo A. C. Fernandes**[1]

[1]Departamento de Engenharia de Computação e Automação - DCA
Universidade Federal do Rio Grande do Norte (UFRN)
Caixa Postal 1524 – 59.078-970 – Natal – RN – Brazil

ceduardobsantos@gmail.com, mfernandes@dca.ufrn.br

Abstract. *This work proposes a Application-Specific System Processor (ASSP) hardware for the SHA-1 hash algorithm. The proposed hardware was implemented on a Field Programmable Gate Array (FPGA) Xilinx Virtex 6 xc6vlx240t-1ff1156. The processing time (throughput) and occupied area were analyzed for various parallel instance implementations of the SHA-1 hashing algorithm. The results showed that SHA-1 proposed hardware achieved a throughput about* 0.644 *Gbps for a unique instance and about* 28 *Gbps for* 48 *instances of the SHA-1 processor embedded on a FPGA. Several applications as password recovery, password validation and verification of big data integrity can be executed efficiently and fast with SHA-1 ASSP.*

Resumo. *Este trabalho propõe um hardware de aplicação específica (*Application-Specific System Processor, ASSP*) para o algoritmo* Secure Hash Algorithm *1 (SHA-1). O hardware proposto foi implementado em uma* Field Programmable Gate Array *(FPGA) Xilinx Virtex 6 xc6vlx240t-1ff1156. O tempo de processamento (throughput) e a área ocupada foram analisados para várias implementações em instâncias paralelas do algoritmo de hash. Os resultados mostraram que o hardware proposto para o SHA-1 alcançou um throughput de* $0,644 Gbps$ *para uma única instância e um pouco maior que* $28 Gbps$ *para* 48 *instâncias em um único FPGA. Várias aplicações como, recuperação de senha, validação de senha e verificação de integridade de grande volume de dados podem ser executadas de forma eficiente e rápida com um ASSP para o SHA1.*

1. Introdução

O *Secure Hash Algorithm* versão um, o SHA-1, é um algoritmo utilizado para verificar a integridade de sequências de dados de comprimento variável a partir de uma operação chamada de função *hash*. Uma função *hash* possui como saída um código de comprimento fixo C dada uma mensagem de comprimento variável K como entrada. Pode-se dizer que a saída da função *hash*, também chamada de código *hash*, é uma assinatura da mensagem de entrada, conhecido na literatura como *fingerprint*. Estas características são usadas em Códigos de Autenticação de Mensagem (MAC), principalmente os *Keyed-Hash Message Authentication Code* (HMAC), os quais fazem o amplo uso do SHA-1 [Kakarountas et al. 2006]. Este que é a versão revisada

 © 2017 Sociedade Brasileira de Computação

do SHA-0, substituto do algoritmo MD5 (*Message Digest 5*) em 1995 pelo NIST (*National Institute of Standards and Technology*), sendo então, o SHA-1 publicado como um padrão de processamento de informações federais (FIPS) de número 180-1 [Stallings 2015].

A função hash SHA-1 além de ter sido selecionada para o Algoritmo de Assinatura Digital (DSA), conforme padronizado pela FIPS 186-4 [NIST 2013], atua na verificação de sequência de dados associados a mensagens de protocolos de comunicação, arquivos e armazenamento de senhas e era usada em certificados digitais antes de ser substituída pelo SHA-2. O SHA-1 gera códigos *hash* de $C = 160$ bits para qualquer tamanho de K. Todavia, com o advento das áreas de Big Data, Internet das coisas (*Internet of Things* - IoT) e outras emergentes faz-se necessário que o código *hash* seja gerado de forma rápida para situações associadas a um grande volume de dados e/ou com um custo energético pequeno no casos de dispositivos dentro de uma rede de sensores no contexto de IoT. Assim, este projeto tem como objetivo uma proposta de implementação em hardware dedicado para o algoritmo SHA-1. O hardware proposto pode ser visto como um processador de aplicação específica também chamado na literatura por *Application-Specific System Processor* (ASSP).

O hardware apresentado foi desenvolvido em um FPGA (*Field-programmable gate array*), plataforma de hardware reconfigurável formada por milhares de células lógicas, que após um processo de síntese comporta-se como um hardware específico associado a um dado algoritmo. O FPGA tem sido uma ferramenta indispensável no desenvolvimento de ASSPs e circuitos integrados dedicados, também chamados de ASIC (*Application-Specific Integrated Circuit*) e ainda como plataforma para aceleração de algoritmos complexos como apresentado em [de Souza and Fernandes 2014, da Silva et al. 2016, Noronha and Fernandes 2016, Torquato and Fernandes 2016, Shi et al. 2012]. Uma das vantagens em desenvolver circuitos específicos é a redução do *clock* quando comparado a implementações em sistemas com processadores de uso geral ou GPP (*General Purpose Processor*). O algoritmo SHA-1 em hardware poderá ser utilizado no desenvolvimento do ASIC para aplicações em IoT ou com utilização no próprio FPGA objetivando acelerar o cálculo do código *hash* em várias aplicações como recuperação de senha, validação de senha, verificação de integridade em grandes volumes de dados e outros.

2. Trabalhos Relacionados

O trabalho apresentado em [Jarvinen 2004] faz uso de um FPGA Xilinx Virtex-II XC2V2000-6 para implementar o SHA-1 com *Itarative Looping* (IL). Nesta implementação a proposta ocupou em torno de 1.275 *Logic Cells* (LC) operando a um *throughput* de 734 Mbps. Já os trabalhos apresentados em [Michail et al. 2005, Kakarountas et al. 2006] também utilizam um FPGA Xilinx Virtex-II XC2V2000-6 para implementar o SHA-1. Estas propostas apresentam um esquema utilizando *Full Pipeline* (FL) que ocupa em torno de 3.519 LC para um *throughput* de 2,5267 Gbps. Comparando com a proposta apresentada em [Jarvinen 2004] o *throughput* é 4 vezes maior devido a utilização de 4 módulos do SHA-1 em pipeline, todavia a área de ocupação também é em torno de 4 vezes maior. Em [Lee et al. 2009] também é apresentada um proposta utilizando FL que consegue atingir um *throughput* de

 © 2017 Sociedade Brasileira de Computação

5,9 Gbps.

Em [Iyer and Mandal 2013] é realizada uma implementação do SHA-1 em um FPGA Xilinx Virtex 5 Xc5vlx50t com linguagem de descrição de hardware *Verilog*. A implementação realizada com IL é semelhante a apresentada em [Jarvinen 2004], todavia, possui uma taxa de ocupação um pouco maior, em torno de 1.351 LC e *throughput* também um pouco maior em torno de 786 Mbps.

O trabalho apresentado por [Khan et al. 2014] tem o objetivo de trazer uma solução do SHA-1 em FPGA com baixa potência energética para usos em dispositivos desprovidos de alta capacidade de energia e com alto *throughput*, além de um tamanho de área pequeno em relação as implementações comparadas. Para isto, os autores se baseiam nos trabalhos apresentados em [Michail et al. 2005, Kakarountas et al. 2006]. Neste trabalho o número de LC é reduzido fazendo uma implementação mais serial, reduzindo também o *throughput*. Uma outra abordagem baseada na implementação descrita em [Michail et al. 2005, Kakarountas et al. 2006] é mostrada em [Michail et al. 2016], no qual, se propõem uma implementação em TSMC 90 nm *Application-Specific Integrated Circuit* (ASIC). Nesta proposta observa-se um *throughput* em torno de 15 Gbps.

Uma comparação entre várias plataformas de FPGA Xilinx com implementação do SHA-1 é apresentada em [Michail et al. 2014]. A implementação é baseada na proposta apresentada em [Michail et al. 2005, Kakarountas et al. 2006] e observa-se um *throughput* máximo em torno de 14,3 Gbps para um FPGA Xilinx Virtex 7.

Trabalhos com implementação do SHA-1 em outras plataformas de hardware podem ser encontrados em [Marks and Niewiadomska-Szynkiewicz 2014, Al-Kiswany et al. 2009] no qual comparações entre GPUs e CPUs (GPP) são realizadas. As GPUs do tipo NVIDIA Tesla M2050 com 448 CUDA *cores* e AMD FirePro V7800 com 1440 *stream processors* podem conseguir picos de *throughput* de até 1,5 Gbps.

A proposta aqui desenvolvida utilizou como alvo o FPGA Virtex 6 xc6vlx240t-11156 e o resultados mostraram um *throughput* de 652Mbps para um único módulo SHA-1. A implementação utilizou a estratégia *Iterative Looping* que ocupa menos área de circuito quando comparada a outras estratégias [Michail et al. 2005, Kakarountas et al. 2006] e diferentemente dos resultados apresentados na literatura, foi possível sintetizar até 48 módulos SHA-1 em uma única FPGA gerando um *throughput* de $28,160$ Gbps.

3. Algoritmo SHA-1

O SHA-1 é um algoritmo de *hash* descrito pela FIPS (*Federal Information Processing Standards Publication*) 180-4 [NIST 2015] e pela RFC 3174 [Network Working Group 2001], o qual opera com mensagens de entrada de comprimento variável. Para cada i-ésima mensagem de entrada, $\mathbf{m}_i$, (de comprimento K_i bits), expressa como

$$\mathbf{m}_i = \left[\begin{array}{cccc} m_0 & m_1 & \ldots & m_{K_i-1} \end{array} \right] \text{ onde } m_k \in \{0,1\} \, \forall \, k, \tag{1}$$

© 2017 Sociedade Brasileira de Computação

o algoritmo SHA-1 gera uma mensagem de saída, $\mathbf{h}_i$, chamada de código *hash*, de tamanho fixo $C = 160$ *bits*, caracterizado como

$$\mathbf{h}_i = \left[\begin{array}{cccc} h_0 & h_1 & \ldots & h_{C-1} \end{array} \right] \text{ onde } h_k \in \{0,1\} \, \forall \, k. \tag{2}$$

A i-ésima mensagem de entrada, $\mathbf{m}_i$, de K_i bits é estendida através da inserção de duas palavras binárias. A primeira, chamada aqui de $\mathbf{p}_i$, possui P_i bits e é inserida em uma operação chamada de Inserção do Preenchimento (ou *Append Padding*). Já a segunda, chamada aqui de $\mathbf{v}_i$, possui T bits e é inserida em uma operação chamada de Inserção do Comprimento (ou *Append Length*). Assim, o cálculo do código *hash*, $\mathbf{h}_i$, para cada i-ésima mensagem de entrada é realizado em uma mensagem estendida, chamada aqui de z_i, que corresponde a uma concatenação da mensagens $\mathbf{m}_i$, $\mathbf{p}_i$ e $\mathbf{v}_i$, ou seja, $\mathbf{z}_i = [\mathbf{m}_i \, \mathbf{p}_i \, \mathbf{v}_i]$. Cada i-ésima mensagem $\mathbf{z}_i$ possui $Z_i = K_i + P_i + T$ bits que podem ser divididos em L_i blocos de comprimento $M = 512$ bits, ou seja,

$$L_i = \frac{Z_i}{M} = \frac{K_i + P_i + T}{512}. \tag{3}$$

O pseudo-código apresentado no Algoritmo 1 apresenta a sequência de etapas necessárias para geração do código *hash* e estas etapas estão descritas em detalhes nas subseções a seguir.

Algoritmo 1 SHA-1 para cada i-ésima mensagem $\mathbf{W}_i$

1: $\mathbf{z}_i \leftarrow [\mathbf{m}_i]$
2: $\mathbf{p}_i \leftarrow \mathit{GeraçãoPreenchimento}(K_i)$
3: $\mathbf{z}_i \leftarrow [\mathbf{m}_i \, \mathbf{p}_i]$
4: $\mathbf{v}_i \leftarrow \mathit{GeraçãoComprimento}(K_i)$
5: $\mathbf{z}_i \leftarrow [\mathbf{m}_i \, \mathbf{p}_i \, \mathbf{v}_i]$
6: $\mathbf{h_i} \leftarrow \mathit{InicializaçãoHash}(\,)$
7: **para** $j \leftarrow 0$ **até** $L_i - 1$ **faça**
8: $\quad \mathbf{b}_j \leftarrow \mathit{DivisãoMensagem}(\mathbf{z}_i)$
9: $\quad n \leftarrow -1$
10: $\quad \mathbf{H}(n) \leftarrow \mathit{InicializaVariáveisHash}(\,)$
11: $\quad$ **para** $n \leftarrow 0$ **até** 79 **faça**
12: $\quad\quad \mathbf{w}(n) \leftarrow \mathit{CálculoFunçãoW}(n, \mathbf{b}_j)$
13: $\quad\quad \mathbf{f}(n) \leftarrow \mathit{CálculoFunçãoF}(n, \mathbf{B}(n), \mathbf{C}(n), \mathbf{D}(n))$
14: $\quad\quad \mathbf{H}(n) \leftarrow \mathit{AtualizaVariáveisHash}(\mathbf{H}(n))$
15: $\quad$ **fim para**
16: $\quad \mathbf{h}_i \leftarrow \mathit{AtualizaHash}(\mathbf{H}(n))$
17: **fim para**

3.1. Inserção do Preenchimento

Esta etapa (ver linhas 2 e 3 do Algoritmo 1) é feita antes do cálculo do código *hash* e tem como função deixar o comprimento da i-ésima mensagem, $\mathbf{m}_i$, divisível por $M = 512$ após a etapa de inserção do comprimento. A mensagem de preenchimento, $\mathbf{p}_i$, associada a i-ésima mensagem de entrada, é também chamada de *Padding*, e

 © 2017 Sociedade Brasileira de Computação

é formada por uma palavra binária de P_i bits no qual o bit mais significativo é 1 e o resto dos bits são 0. A geração da mensagem de preenchimento é realizada pela função *GeraçãoPreenchimento*(K_i) apresentada na linha 2 do Algoritmo 1. O cálculo do valor de P_i pode ser expresso por

$$P_i = \begin{cases} 448 - (K_i \bmod 512) & \text{para } (K_i \bmod 512) < 448 \\ 512 - (K_i \bmod 512) + 448 & \text{para } (K_i \bmod 512) \geq 448 \end{cases}, \quad (4)$$

onde a operação a mod b retorna o resto inteiro da divisão entre a e b. Assim, $\mathbf{p}_i$ pode ser expresso como

$$\mathbf{p}_i = \left[\begin{array}{cccc} p_0 & p_1 & \dots & p_{P_i-1} \end{array} \right], \quad (5)$$

onde, $p_0 = 1$ e $p_i = 0$ para $i = 1 \dots P_i - 1$.

3.2. Inserção do Comprimento

Nesta etapa (linhas 4 e 5 do Algoritmo 1) é adicionada a mensagem $\mathbf{v}_i$ caracterizada por uma palavra binária de $T = 64$ bits e expressa como

$$\mathbf{v}_i = \left[\begin{array}{cccc} v_0 & v_1 & \dots & v_{T-1} \end{array} \right] \text{ onde } v_k \in \{0,1\} \, \forall \, k. \quad (6)$$

A geração da mensagem de comprimento é realizada pela função *GeraçãoComprimento*(K_i) apresentada na linha 4 do Algoritmo 1. A mensagem $\mathbf{v}_i$ armazena o valor do comprimento da i-ésima mensagem de entrada $\mathbf{m}_i$, ou seja,

$$\mathbf{v}_i = Binary(K, T) \quad (7)$$

onde $Binary(a, b)$ é uma função que retorna um vetor de tamanho b com a representação binária de um número decimal qualquer a em b bits no padrão *big-endian*.

A norma da FIPS 180-4 [NIST 2015], supõem que o tamanho, K_i, da maioria das mensagens pode ser representado por 64 bits, ou seja, $K_i < 2^T$.

Finalmente, ao final da segunda etapa a menagem, $\mathbf{z}_i$, que é uma extensão da i-ésima mensagem original de entrada $\mathbf{m}_i$, é formada (linha 5 do Algoritmo 1). Neste trabalho a mensagem $\mathbf{z}_i$ é caracterizada por um vetor de Z_i bits expresso como

$$\mathbf{z}_i = \left[\begin{array}{cccc} z_0 & z_1 & \dots & z_{Z-1} \end{array} \right] \text{ onde } z_k \in \{0,1\} \, \forall \, k. \quad (8)$$

3.3. Inicialização do Código Hash

A inicialização do código *hash* (ver linha 6 do Algoritmo 1) é padronizada pela FIPS 180-4 [NIST 2015] de acordo com as seguintes expressões:

$$\mathbf{ha} = \left[\begin{array}{ccc} h_0 & \dots & h_{31} \end{array} \right] = Binary(1732584193, 32), \quad (9)$$

$$\mathbf{hb} = \left[\begin{array}{ccc} h_{32} & \dots & h_{63} \end{array} \right] = Binary(4023233417, 32), \quad (10)$$

$$\mathbf{hc} = \left[\begin{array}{ccc} h_{64} & \dots & h_{95} \end{array} \right] = Binary(2562383102, 32), \quad (11)$$

$$\mathbf{hd} = \left[\begin{array}{ccc} h_{96} & \dots & h_{127} \end{array} \right] = Binary(0271733878, 32), \quad (12)$$

e

$$\mathbf{he} = \left[\begin{array}{ccc} h_{128} & \dots & h_{159} \end{array} \right] = Binary(3285377520, 32), \quad (13)$$

onde

$$\mathbf{h}_i = \left[\begin{array}{ccccc} \mathbf{ha} & \mathbf{hb} & \mathbf{hc} & \mathbf{hd} & \mathbf{he} \end{array} \right]. \quad (14)$$

© 2017 Sociedade Brasileira de Computação

3.4. Divisão da Mensagem

Nesta etapa, linha 8 do Algoritmo 1, a mensagem $\mathbf{z}_i$ é quebrada em L_i blocos de $M = 512$ bits, ou seja,

$$\mathbf{z}_i = \left[\begin{array}{cccc} \mathbf{b}_0 & \mathbf{b}_1 & \ldots & \mathbf{b}_{L_i-1} \end{array} \right], \tag{15}$$

onde cada j-ésimo bloco associado a i-ésima mensagem é expresso como

$$\mathbf{b}_j = \left[\begin{array}{cccc} b_{j,0} & b_{j,1} & \ldots & b_{j,M-1} \end{array} \right] \text{ onde } b_{j,k} \in \{0,1\} \forall\, k. \tag{16}$$

O j-ésimo bloco, $\mathbf{b}_j$, também pode representado como

$$\mathbf{b}_j = \left[\begin{array}{cccc} \mathbf{u}_j[0] & \mathbf{u}_j[1] & \ldots & \mathbf{u}_j[15] \end{array} \right], \tag{17}$$

onde $\mathbf{u}_j[k]$ é uma mensagem de 32 bits, ou seja,

$$\mathbf{u}_j[k] = \left[\begin{array}{cccc} u_j[k,0] & u_j[k,1] & \ldots & b_j[k,31] \end{array} \right] \tag{18}$$

onde $u_j[k,l] \in \{0,1\} \forall\, l$.

3.5. Inicialização das Variáveis Hash H(n)

O algoritmo SHA-1 possui cinco variáveis de 32 bits, chamadas de $\mathbf{A}(n)$, $\mathbf{B}(n)$, $\mathbf{C}(n)$, $\mathbf{D}(n)$ e $\mathbf{E}(n)$ que se atualizam durante as iterações do algoritmo. Estas variáveis são caracterizadas neste trabalho como vetores expressos como

$$\mathbf{X}(n) = \left[\begin{array}{cccc} x_0 & x_1 & \ldots & x_{31} \end{array} \right] \text{ onde } x_k \in \{0,1\} \forall\, k, \tag{19}$$

onde, $\mathbf{X}(n) \in \{\mathbf{A}(n), \mathbf{B}(n), \mathbf{C}(n), \mathbf{D}(n), \mathbf{E}(n)\}$. A junção destas cinco variáveis compõem um vetor de 160 posições caracterizado como

$$\mathbf{H}(n) = \left[\begin{array}{ccccc} \mathbf{A}(n) & \mathbf{B}(n) & \mathbf{C}(n) & \mathbf{D}(n) & \mathbf{E}(n) \end{array} \right]. \tag{20}$$

A inicialização destas variáveis, no instante $n = -1$, (ver linha 10 do Algoritmo 1) de acordo com a FIPS 180-4 [NIST 2015] ocorre com o recebimento dos mesmos valores que iniciam o *hash* $\mathbf{h}_i$, logo $\mathbf{A}(-1) = \mathbf{ha}$, $\mathbf{B}(-1) = \mathbf{hb}$, $\mathbf{C}(-1) = \mathbf{hc}$, $\mathbf{D}(-1) = \mathbf{hd}$ e $\mathbf{E}(-1) = \mathbf{he}$.

3.6. Cálculo da variável w(n)

No SHA-1 são necessárias 80 iterações para uma saída válida, $\mathbf{h}_i$, associada a cada i-ésima mensagem (ver linha 11 do Algoritmo 1). Em cada n-ésima iteração é calculada uma variável, $\mathbf{w}(n)$ expressa como

$$\mathbf{w}(n) = \begin{cases} \mathbf{u}_j[n] & \text{para } 0 \leq n \leq 15 \\ \mathbf{sw}[n] & \text{para } 16 \leq n \leq 79 \end{cases}, \tag{21}$$

$$\mathbf{Hash/s} = \begin{cases} 1\ hash \text{ — } 9,932 \cdot 80 \cdot 10^{-9} \\ H\ hash \text{ — } 1s \end{cases} \tag{22}$$

$$\mathbf{Hash/s} = 80,548 \text{ Mhash/s} \tag{23}$$

 © 2017 Sociedade Brasileira de Computação

$$\mathbf{Hash/s} = \begin{cases} 48\ hash \text{ — } 10,909 \cdot 80 \cdot 10^{-9} \\ H\ hash \text{ — } 1s \end{cases} \tag{24}$$

$$\mathbf{Hash/s} = 3,52 \text{ Ghash/s} \tag{25}$$

onde

$$\mathbf{sw}[n] = \text{lr}\left(\mathbf{u}_j[n-3] \oplus \mathbf{u}_j[n-8] \oplus \mathbf{u}_j[n-14] \oplus \mathbf{u}_j[n-16], 1\right) \tag{26}$$

onde $\oplus$ é a operação ou exclusivo bit a bit e $\text{lr}(\mathbf{r}, \mathbf{s})$ representa a função *leftrotate* que é expressa como

$$\text{lr}(\mathbf{r}, \mathbf{s}) = (\mathbf{r} \ll \mathbf{s}) \vee (\mathbf{r} \gg (32 - \mathbf{s})), \tag{27}$$

onde $\vee$, $\ll$ e $\gg$ são os operadores OU e de deslocamento bit a bit a esquerda e a direita, respectivamente.

3.7. Cálculo da Função $f(\cdot)$

Em cada n-ésima iteração de cada j-ésimo bloco, $\mathbf{b}_j(n)$ é calculada uma função não linear, $f(\cdot)$, a partir das informações das variáveis *hash* $\mathbf{B}(n)$, $\mathbf{C}(n)$ e $\mathbf{D}(n)$. A saída da função, $f(\cdot)$ é armazenada no vetor $\mathbf{f}(n)$ (linha 13 do Algoritmo 1), expresso como

$$\mathbf{f}(n) = f(n, \mathbf{B}, \mathbf{C}, \mathbf{D}) = \begin{cases} \alpha(n) \text{ para } n = 0 \ldots 19 \\ \beta(n) \text{ para } n = 20 \ldots 39 \\ \gamma(n) \text{ para } n = 40 \ldots 59 \\ \delta(n) \text{ para } n = 60 \ldots 79 \end{cases}, \tag{28}$$

onde

$$\alpha(n) = (\mathbf{B}(n-1) \wedge \mathbf{C}(n-1)) \vee (\neg\mathbf{B}(n-1) \wedge \mathbf{D}(n-1)), \tag{29}$$

$$\beta(n) = \mathbf{B}(n-1) \oplus \mathbf{C}(n-1) \oplus \mathbf{D}(n-1), \tag{30}$$

$$\gamma(n) = (\mathbf{B}(n-1) \wedge \mathbf{C}(n-1)) \vee (\mathbf{B}(n-1) \wedge \mathbf{D}(n-1)) \vee (C(n-1) \wedge D(n-1)) \tag{31}$$

e

$$\delta(n) = \mathbf{B}(n-1) \oplus \mathbf{C}(n-1) \oplus \mathbf{D}(n-1), \tag{32}$$

onde $\neg$ e $\wedge$ são operações de negação e E bit a bit, respectivamente.

3.8. Atualização das Variáveis Hash

Também em cada n-ésima iteração de cada j-ésimo bloco $\mathbf{b}_j(n)$ é atualizado o valor das variáveis, $\mathbf{A}(n)$, $\mathbf{B}(n)$, $\mathbf{C}(n)$, $\mathbf{D}(n)$ e $\mathbf{E}(n)$ após o cálculo de $\mathbf{f}(n)$ (ver linha 14 do Algoritmo 1). A atualização destas variáveis é representada pelas seguintes equações:

$$\mathbf{E}(n) = \mathbf{D}(n-1), \tag{33}$$

$$\mathbf{D}(n) = \mathbf{C}(n-1), \tag{34}$$

$$\mathbf{C}(n) = \text{lr}(\mathbf{B}(n-1), 30), \tag{35}$$

$$\mathbf{B}(n) = \mathbf{A}(n-1) \tag{36}$$

© 2017 Sociedade Brasileira de Computação

e

$$\mathbf{A}(n) = \mathbf{V}(n) + \mathbf{Z}(n) + \mathrm{lr}(A(n-1), 5), \tag{37}$$

no qual,

$$\mathbf{Z}(n) = \mathbf{W}(n) + \mathbf{E}(n-1) \tag{38}$$

e

$$\mathbf{V}(n) = \mathbf{f}(n) + \mathbf{k}(n). \tag{39}$$

O SHA-1 possui quatro constantes $\mathbf{k}(n)$ de 32 bits, as quais são usadas na n-ésima iteração de cada j-ésimo bloco $\mathbf{b}_j(n)$, conforme especificado por

$$K(n) = \begin{cases} 1518500249 & \text{para } n = 0 \ldots 19 \\ 1859775393 & \text{para } n = 20 \ldots 39 \\ 2400959708 & \text{para } n = 40 \ldots 59 \\ 3395469782 & \text{para } n = 60 \ldots 79 \end{cases}. \tag{40}$$

3.9. Atualização do código *hash*

Para cada j-ésimo bloco, $\mathbf{b}_j$, o SHA executa 80 iterações e ao final de cada j-ésimo bloco o código *hash* é atualizado de forma linear seguindo as seguintes expressões:

$$\mathbf{ha} = \mathbf{ha} + \mathbf{A}(79), \tag{41}$$

$$\mathbf{hb} = \mathbf{hb} + \mathbf{B}(79), \tag{42}$$

$$\mathbf{hc} = \mathbf{hc} + \mathbf{B}(79), \tag{43}$$

$$\mathbf{hd} = \mathbf{hd} + \mathbf{D}(79), \tag{44}$$

e

$$\mathbf{he} = \mathbf{he} + \mathbf{E}(79). \tag{45}$$

Assim para cada i-ésima mensagem, $\mathbf{m}_i$, o valor do código *hash* associado, $\mathbf{h}_i$, é encontrado em

$$N_i = L_i \times 80 \tag{46}$$

iterações, onde N_i é definido neste trabalho como o número total de interações para o cálculo do *hash* associado a uma mensagem $\mathbf{m}_i$.

4. Implementação proposta

A Figura 1 apresenta a proposta deste artigo para a arquitetura geral do SHA-1 implementado em hardware. A estrutura permite a visualização do algoritmo em *Register Transfer Level* (RTL), no qual pode-se observar o fluxo de sinais (ou variáveis) entre os componentes de *datapath* e os registradores RA, RB, RC, RD e RE. O hardware inicia com a i-ésima mensagem $\mathbf{m}_i$ entrando em um módulo chamado de INIT que é responsável pelas funcionalidades apresentadas entre linhas 1 e 6 do Algoritmo 1, o controle dos dois *loops* (linhas 7 e 11) e a inicialização das variáveis *hash* ($\mathbf{A}(n)$, $\mathbf{B}(n)$, $\mathbf{C}(n)$, $\mathbf{D}(n)$ e $\mathbf{E}(n)$) a cada j-ésimo bloco, $\mathbf{b}_j$, através do sinal $\mathbf{h}0$.

 © 2017 Sociedade Brasileira de Computação

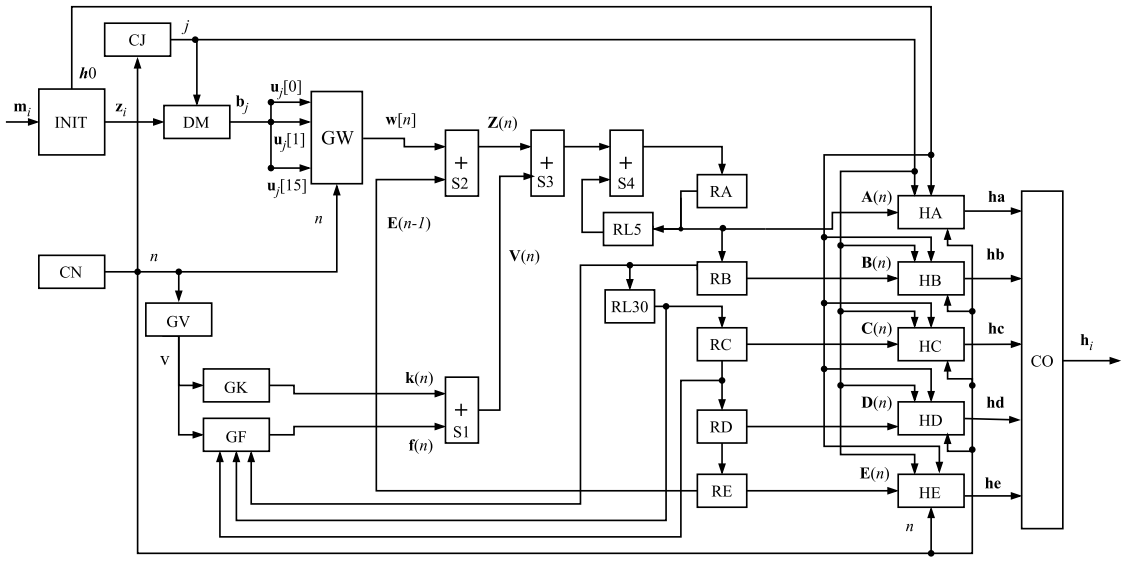


Figura 1. Arquitetura geral da implementação em hardware do SHA-1 proposto.

Os blocos CJ e CN são contadores de $\log_2(L)$ e 7 bits, respectivamente. O contador CN é responsável pela iteração do *loop* da linha 11 do Algoritmo 1, gerando o sinal n. Já o contador CJ é incrementado pelo contador CN e controla a iteração do *loop* da linha 7 do Algoritmo 1, através do sinal j. Com base na linha 8 do Algoritmo 1 e subseção 3.4, o módulo DM faz a divisão da i-ésima mensagem $\mathbf{z}_i$ em L blocos de $M = 512$ bits, no qual cada j-ésimo bloco é caracterizado na Figura 1 pelo sinal $\mathbf{b}_j$. O sinal $\mathbf{b}_j$, de $M = 512$ bits é, então, dividido igualmente em 16 barramentos de 32 bits, no qual, cada i-ésimo barramento é representado pelo sinal $\mathbf{u}_j[n]$. Após esta etapa é gerado sinal $\mathbf{w}[n]$ pelo módulo GW (linha 12 do Algoritmo 1) a partir do sinal do contador CN.

Os módulos GF, GK e GW representam as operações expressas pelas Equações 28, 40 e 21, respectivamente. Já o módulos RL5 e RL30 representam operações de leftrotate expressa nas Equações 37 e 35. Observa-se que diferentemente das implementações em processadores sequenciais como GPP, uC (Micro-controladores) e outros, estas equações são executadas em paralelo, acelerando o algoritmo SHA-1. Os detalhes da implementação dos módulos GF, GK, GW, RL5 e RL30 estão detalhados nas subseções a seguir.

4.1. Módulo GF

O módulo GF implementa a função descrita na subseção 3.7 e apresentado na linha 13 do Algoritmo 1. Este módulo é formado por um multiplexador chamado de GF-MUX que seleciona o tipo função a partir do valor de n de acordo com a Equação 28 e detalhado na Figura 2.

A seleção do tipo da função no multiplexador GF-MUX é controlada pelo módulo GV, por meio de uma lógica binária com comparadores e portas lógicas

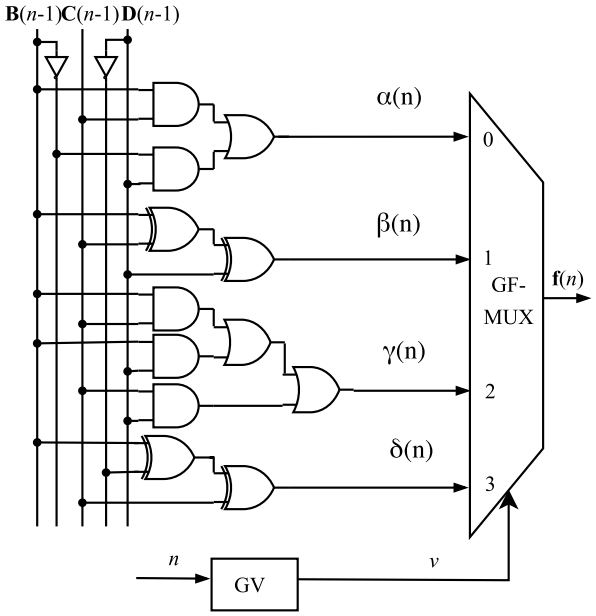


Figura 2. Arquitetura do módulo GF**.**

correspondente a cada intervalo, tendo as seguintes saídas,

$$GV = \begin{cases} 0 & \text{para } n = 0 \ldots 19 \\ 1 & \text{para } n = 20 \ldots 39 \\ 2 & \text{para } n = 40 \ldots 59 \\ 3 & \text{para } n = 60 \ldots 79 \end{cases} . \tag{47}$$

Cada qual selecionando uma função $\mathbf{f}(n)$ baseado no contador n de 7 bits do módulo CN.

4.2. Módulo GW

O módulo GW é composto por 16 mensagens de 32 bits $\mathbf{u}_j[n]$ na entrada, originadas de $\mathbf{b}_j[n]$, conforme a Equação 17 da Subseção 3.4, e possui a função de realizar a operação demonstrada pela Equação 21 descrita na subseção 3.6 e linha 12 do Algoritmo 1.

A Figura 3 detalha o módulo que é formado por um multiplexador de 80 entradas, chamado de W-MUX que é selecionado a partir do sinal n. Para os valores de n de 16 a 79 é gerado o sinal $\mathbf{sw}[n]$ expresso pela equação 26 através do módulo SWk onde $k = 16 \ldots 79$, especificado na Figura 4.

Cada k-ésimo módulo SWk é formado por um registrador, chamado aqui de RWk, um módulo de leftrotate (ver Equação 27) chamado de LR1, uma porta ou exclusivo (XOR) e um comparador. O registrador RWk armazena o valor do sinal $\mathbf{sw}[n]$ através do comparador quando $n = k - 3$. A porta XOR realiza a operação descrita na Equação 26 e o módulo LR1 realiza a função de leftrotate para $\mathbf{s} = 1$. A Figura 5 detalha um módulo genérico associado a implementação do leftrotate com base na Equação 27.

 © 2017 Sociedade Brasileira de Computação

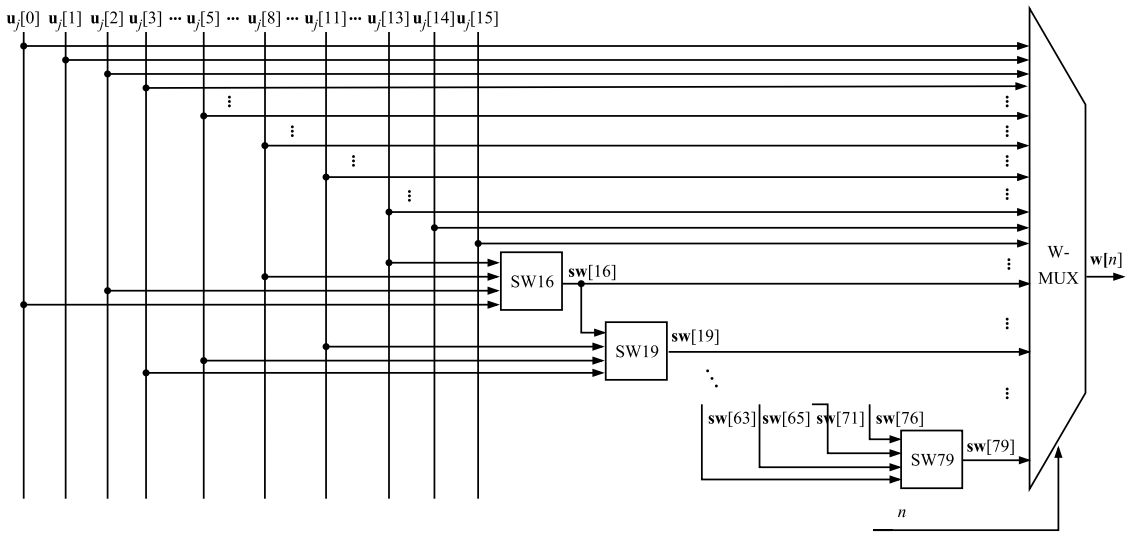


Figura 3. Arquitetura do módulo GW

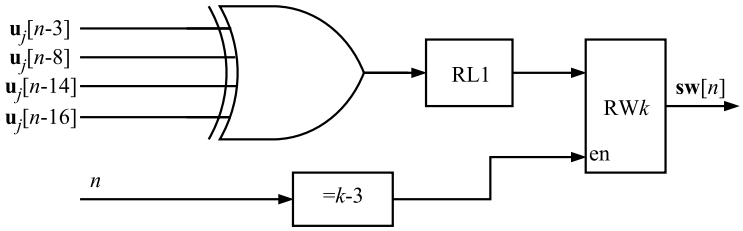


Figura 4. Operações do módulo SWk

Os módulos RL5(s) e RL30 realizam a operação de leftrotate para $\mathbf{s} = 5$ (ver Equação 37) e $\mathbf{s} = 30$ (ver Equação 35), respectivamente. A implementação destes módulos também é caracteriza pela Figura 5.

4.3. Processamento do hash $\mathbf{h_i}$

Após a geração dos sinais $\mathbf{w}(n)$, $\mathbf{k}(n)$, $\mathbf{f}(n)$, em cada n-ésima iteração, e do valor $\mathbf{E}(n-1)$, os sinais $\mathbf{Z}(n)$ e $\mathbf{V}(n)$, ambos de 32 bits, são calculados através dos módulos de soma S1 e S2, executados em paralelo, posteriormente S3 e S4. Todos os módulos de soma utilizados na implementação são circuitos específicos de 32 bits, o que otimiza o tempo de processamento e o espaço ocupado pelo circuito total. O cálculo dos sinais $\mathbf{Z}(n)$ e $\mathbf{V}(n)$ é expresso pelas Equações 38 e 39 e é executado durante a linha 14 do Algoritmo 1. A última etapa de cada n-ésima iteração é a atualização das variáveis *hash*, $\mathbf{A}(n)$, $\mathbf{B}(n)$, $\mathbf{C}(n)$, $\mathbf{D}(n)$ e $\mathbf{E}(n)$, armazenada nos registradores RA, RB, RC, RD e RE, respectivamente. Outra etapa importante no SHA-1 é a descrita na 4.2, na qual o valor do registrador RC é atualizado por meio da operação $\mathrm{lr}(r, 30)$, expressa em detalhes pela Equação 27. A cada iteração de $\mathbf{n}$ os valores dos registradores deslocam-se entre si atualizando as outras variáveis *hash* de acordo com as Equações 33 a 37. Este procedimento é executado durante a linha 14 do Algoritmo 1.

Ao final das 80 iterações do *loop* em n (linha 11 do Algoritmo 1) as partes que compõem o código *hash*, **ha**, **hb**, **hc**, **hd** e **he** (ver Equação 14), são atualizadas

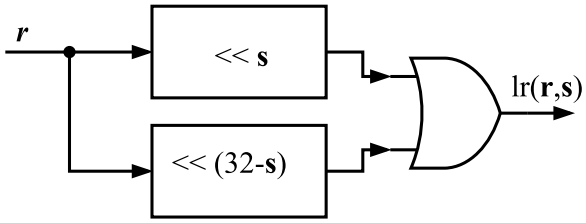


Figura 5. Arquitetura do módulo RLs.

pelos módulos HA, HB, HC, HD e HE, respectivamente. Esta etapa é executada na linha 16 do Algoritmo 1. Finalmente ao final das N_i iterações (ver Equação 46) tem-se o valor final do código *hash*, $\mathbf{h}_i$, associado a i-ésima mensagem. O módulo CO têm a função de concatenar os 5 barramentos de 32 bits formados pelos sinais **ha**, **hb**, **hc**, **hd** e **he** e gerar um sinal serial com o código *hash* $\mathbf{h}_i$.

5. Resultados

A Tabela 1 apresenta os resultados obtidos após o processo de síntese do hardware proposto neste trabalho (ver Figura 1). São apresentados resultados relativos a ocupação do hardware no FPGA alvo (Virtex 6 xc6vlx240t-11156) bem como os resultados associados a latência e o *throughput* alcançado após o processo de síntese. Foram gerados resultados para várias implementações em paralelo do algoritmo SHA-1 de acordo com Figura 1 seguindo uma linha diferente dos trabalhos apresentados em [Michail et al. 2005, Kakarountas et al. 2006, Khan et al. 2014, Michail et al. 2016], os quais utilizam estruturas seriadas (em *pipeline*). Já a proposta apresentada aqui faz utilização de vários módulos do SHA-1 em paralelo, permitindo acelerar o *throughput* principalmente em casos de recuperação de senha por força bruta, no qual existem um grande número de códigos *hash* a serem gerados.

Tabela 1. Resultados relativos a ocupação, taxa de amostragem e throughput para várias implementações paralelas do algoritmo SHA-1.

NI	NR	PR (%)	NLUT	PLUT (%)	T_s (ns)	R_s (Gbps)
1	2.154	0,71	2.605	1,72	9,932	0,644
4	8.575	2,84	10, 388	6,89	9,961	2,570
8	17.136	5,68	20.662	13,71	9,965	5,138
16	34.255	11,36	43.263	28,70	9,994	10,246
32	68.498	22,72	86.873	57,64	9,994	18,296
48	102.733	34,08	129.902	86,19	10,909	28,160

A primeira coluna da tabela, chamada de NI, indica o número de implementações paralelas realizadas. A segunda coluna, NR, apresenta o número de registradores utilizados em hardware implementado no FPGA e a terceira coluna, chamada de PR, representa a porcentagem de registradores utilizados em relação ao total disponível no FPGA designado que são de 301.440. Já a quarta e quinta coluna, chamadas de NLUT e PLUT, representam a quantidade de células lógicas

 © 2017 Sociedade Brasileira de Computação

utilizadas como LUTs (*Lookup Tables*) para construção de circuitos digitais e a porcentagem de LUTs utilizados em relação ao total disponível no FPGA proposto que são de 150.720. Finalmente, a sexta e sétima coluna mostram os resultados, obtidos para várias implementações, de taxa de amostragem, T_s e *throughput*, R_s, respectivamente.

O hardware, desenvolvido de modo paralelo como mostrado na Seção 4, com barramentos de 32 bits para que o tempo de amostragem, T_s, seja correspondente ao *clock*, isto é, cada n-ésima iteração (ver *Loop* da linha 11 do Algoritmo 1) é realizada em um tempo de pulso de *clock*, chamado aqui de $t_{\text{CLK}} = T_s$. O valores de T_s estão exibidos na sexta coluna da Tabela 1. É possível verificar que não há uma mudança significativa com o aumento de NI, ou seja, para um incremento de 48 vezes de NI houve apenas um incremento de menos de 1 ns em T_s, o que representa um aumento de quase 32 vezes no *throughput* de geração do *hash*.

Com base no Algoritmo 1 e arquitetura apresentada na Figura 1, para cada j-ésimo bloco $\mathbf{b}_j$ de $M = 512$ bits são executadas 80 iterações (ver Equação 46), assim, o *throughput* do hardware proposto pode ser calculado como

$$R_s = \frac{M \times \text{NI}}{80 \times T_s} = \frac{512 \times \text{NI}}{80 \times T_s} = \frac{64 \times \text{NI}}{10 \times T_s}. \tag{48}$$

É importante destacar que valores de *throughput* maiores que 15 Gbps são inéditos na literatura (NI = 32 e NI = 48). Um *throughput* de 28, 16 Gbps equivale a recuperar uma senha totalmente desconhecida (usando o método de força bruta) de 6 dígitos apenas numéricos no máximo em 20 ms ou um senha também de 6 dígitos alfa numéricos (cada dígito com 62 possibilidades) a partir de um código *hash* no máximo em 17, 4 minutos.

6. Conclusões

Este trabalho apresentou uma proposta de implementação em hardware de algoritmo SHA-1. A estrutura proposta, também chamada de ASSP, foi sintetizada em um FPGA objetivando validar o hardware proposto. Todos os detalhes de implementação da proposta foram apresentados e analisados em termos de área de ocupação e tempo de processamento. Os resultados obtidos são bastante significativos e apontam para novas possibilidades de utilização de algoritmos de *hash* em hardware dedicado para aplicações de tempo real e de grande volume de dados.

Referências

Al-Kiswany, S., Gharaibeh, A., Santos-Neto, E., and Ripeanu, M. (2009). On gpu's viability as a middleware accelerator. *Cluster Computing*, 12(2):123–140.

da Silva, L., Torquato, M., and Fernandes, M. A. C. (2016). Proposta de arquitetura em hardware para fpga da técnica q-learning de aprendizagem por reforço. In *Encontro Nacional de Inteligência Artificial e Computacional - ENIAC 2016*, Recife, PE.

de Souza, A. and Fernandes, M. (2014). Parallel fixed point implementation of a radial basis function network in an fpga. *Sensors*, 14(10):18223–18243.

© 2017 Sociedade Brasileira de Computação

Iyer, N. C. and Mandal, S. (2013). Implementation of secure hash algorithm-1 using fpga. *International Journal of Information and Computation Technology*, 3(8):757–764.

Jarvinen, K. (2004). Design and implementation of a sha-1 hash module on fpgas.

Kakarountas, A. P., Michail, H., Milidonis, A., Goutis, C. E., and Theodoridis, G. (2006). High-speed fpga implementation of secure hash algorithm for ipsec and vpn applications. *The Journal of Supercomputing*, 37(2):179–195.

Khan, S., ul Abideen, Z., and Paracha, S. S. (2014). An ultra low power and high throughput fpga implementation of sha-1 hash algorithm. *International Journal of Computer Science and Information Security*, 12(8):80–86.

Lee, E.-H., Lee, J.-H., Park, I.-H., and Cho, K.-R. (2009). Implementation of high-speed sha-1 architecture. *IEICE Electronics Express*, 6(16):1174–1179.

Marks, M. and Niewiadomska-Szynkiewicz, E. (2014). Hybrid CPU/GPU platform for high performance computing. In *28th European Conference on Modelling and Simulation, ECMS 2014, Brescia, Italy, May 27-30, 2014*, pages 508–514.

Michail, H., Athanasiou, G., Theodoridis, G., and Goutis, C. (2014). On the development of high-throughput and area-efficient multi-mode cryptographic hash designs in fpgas. *Integration, the VLSI Journal*, 47(4):387 – 407.

Michail, H., Kakarountas, A. P., Koufopavlou, O., and Goutis, C. E. (2005). A low-power and high-throughput implementation of the sha-1 hash function. In *2005 IEEE International Symposium on Circuits and Systems*, pages 4086–4089 Vol. 4.

Michail, H. E., Athanasiou, G. S., Theodoridis, G., Gregoriades, A., and Goutis, C. E. (2016). Design and implementation of totally-self checking sha-1 and sha-256 hash functions' architectures. *Microprocessors and Microsystems*, 45:227 – 240.

Network Working Group (2001). Request for Comments: 3174. http://www.faqs.org/rfcs/rfc3174.html.

NIST (2013). Digital signature standard (dss). *FIPS PUB 186-4*.

NIST (2015). Secure Hash Standard (SHS). http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

Noronha, D. and Fernandes, M. A. C. (2016). Implementação em fpga de máquina de vetores de suporte (svm) para classificação e regressão. In *Encontro Nacional de Inteligência Artificial e Computacional - ENIAC 2016*, Recife, PE.

Shi, Z., Ma, C., Cote, J., and Wang, B. (2012). Hardware implementation of hash functions. In *Introduction to Hardware Security and Trust*, pages 27–50. Springer.

Stallings, W. (2015). *Criptografia e segurançã de redes*. Pearson Education do Brasil, 6th edition.

Torquato, M. F. and Fernandes, M. A. C. (2016). Proposta de implementação paralela de algoritmo genético em fpga. In *XXI Congresso Brasileiro de Automática (CBA 2016)*, Vitória, ES.

 © 2017 Sociedade Brasileira de Computação

Eliminação Segura de Arquivos em Memória Não-Volátil

Julia S. Weber[1], Avelino F. Zorzo[1]

[1]Faculdade de Informática –PUCRS – Porto Alegre – RS - Brasil

julia.weber@acad.pucrs.br, avelino.zorzo@pucrs.br

Abstract. *The Internet of Things (IoT) and the popularization of mobile devices bring new challenges regarding the removal of files. Techniques traditionally employed in magnetic media are not effective when applied to non-volatile memories, such as flash memory. Consequently, new methods of safe removal have been developed, which employ Zero Override and Block Erase. This work analyzes these methods and proposes a new one, with better performance. The proposed method is hybrid, combining overwriting and deletion operations to obtain a balanced use of these operations, avoid unnecessary deletion of unused blocks and reduce premature memory wear.*

Resumo. *A Internet das Coisas (IoT) e a popularização de dispositivos móveis traz novos desafios quanto a remoção de arquivos. Técnicas tradicionalmente empregadas em meios magnéticos não são efetivas para memórias não voláteis, como a memória flash. Consequentemente, novos métodos de remoção segura foram desenvolvidos, que empregam operações de Sobrescrita com Zeros e de Apagamento de Blocos. Este trabalho analisa estes métodos e propõe um novo, com melhor desempenho. O método proposto é um híbrido, combinando de forma equilibrada operações de sobrescrita e apagamento, para evitar o apagamento desnecessário de blocos ainda não utilizados e reduzir o desgaste prematuro da memória.*

1. Introdução

Memória não-volátil é um tipo especial de memória que pode reter informações mesmo em caso de perda de energia. Entre estes tipos de memória, a memória *flash* se destaca, pois fornece grande capacidade de armazenamento de dados que é persistente e que pode ser acessada mais rapidamente do que os discos rígidos tradicionais. Com o crescente uso de memória *flash* em celulares, *smartphones, tablets* e *pen drives*, cresce também a necessidade de garantir a proteção dos dados armazenados. Muitos arquivos nestes dispositivos contêm dados pessoais e privados, que não devem ser acessados por pessoas não autorizadas, o que pode acontecer em caso de roubo ou perda do dispositivo, inclusive para arquivos que foram removidos. Arquivos removidos não devem ser recuperados por terceiros, pois isto também compromete a privacidade.

Dispositivos de armazenamento são fundamentais em sistemas de computação, e uma regra básica dos projetistas é que os dados devem ser protegidos. Contudo não somente informações sigilosas devem ser protegidas, mas a privacidade do usuário também é muito importante. Com a facilidade de expor em redes públicas informações sigilosas, o usuário deve se resguardar de atacantes que vasculham informações que já deveriam ter sido apagadas do meio de armazenamento. Como muitas vezes este

 © 2017 Sociedade Brasileira de Computação

apagamento é somente lógico e não físico, informações logicamente removidas podem ser recuperadas através da análise do meio físico. Assim, para preservar a privacidade do usuário, deve haver a possibilidade de informações pessoais serem imediata e permanentemente apagadas de dispositivos de armazenamento. Este requisito desempenha um papel crítico em todos os sistemas de práticas de gerenciamento de dados [13].

Eliminação segura de dados consiste em suprimir permanentemente os dados digitais de um meio físico de tal modo que os dados se tornem irrecuperáveis [7]. Assim, por exemplo, métodos antiforenses [6] são utilizados para eliminar permanentemente arquivos específicos ou sistemas de arquivos inteiros. Para a eliminação de dados, algumas soluções usam deleção por sobrescrita (*overwriting*) [2] [7] [8], sendo estas as mais comuns no momento, ou mesmo por criptografia [9].

Contudo, isso dá origem a outro problema. O uso de dispositivos com memórias não voláteis aumentou significativamente nos últimos anos. Por questões de privacidade dos usuários, garantir que os dados contidos nestas memórias sejam devidamente apagados também se tornou uma preocupação. Quando *pendrives*, celulares, *tablets* e afins são roubados, perdidos ou descartados, os dados continuam armazenados na memória. Mesmo que o usuário tenha o cuidado de excluir os arquivos, os dados permanecem na memória *flash* e ainda podem ser recuperados [16]. Garantir a remoção segura de dados é uma preocupação que atinge tanto empresas e órgão governamentais, por questões de sigilo, como também usuários comuns, por questões de privacidade.

Assim, este artigo apresenta uma proposta para remoção segura de arquivos em memórias não-voláteis que emprega operações de sobrescrita com zeros e de apagamento de blocos. O método proposto é híbrido, combinando de forma equilibrada operações de sobrescrita e apagamento, para evitar o apagamento desnecessário de blocos ainda não utilizados e reduzir o desgaste prematuro da memória.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta as características de memórias *flash*; a Seção 3 descreve alguns dos principais métodos de remoção segura de dados em memórias não-voláteis, e apresenta o método proposto; a Seção 4 apresenta os experimentos realizados para comparar métodos no estado da arte com o método proposto; e, a Seção 5 apresenta as conclusões deste trabalho.

2. Características de memórias *flash*

A memória *flash* é um meio de armazenamento não-volátil que consiste em uma série de componentes eletrônicos que armazenam informações [1]. Como tal, dispositivos portáteis usam quase exclusivamente memória *flash*. Entretanto, memória *flash* tem diferenças significativas com memória volátil (RAM) e com tecnologias de disco magnético, requerendo *drivers* de software e sistemas de arquivos especiais. A Figura 1 ilustra como a arquitetura de uma memória *flash* é relacionada com aplicativos e sistemas de arquivos.

Sistemas de arquivo são normalmente implementados em memória *flash* com tecnologia NAND. A memória *flash* NAND é dividida em dois níveis de granularidade. O primeiro é chamado nível de unidades de apagamento, que são da ordem de 64KB ou 128 KB [13] no tamanho. Cada unidade de apagamento é dividida em blocos (*blocks* ou

 © 2017 Sociedade Brasileira de Computação

pages), que são da ordem de 2 KB ou 4 KB de tamanho. Uma unidade de apagamento é a região mínima de apagamento e os blocos são a unidade de operações de leitura e escrita [5]. Não se pode gravar dados em um bloco de memória *flash*, a menos que este bloco tenha sido apagado anteriormente. A operação de apagamento executada em uma unidade de apagamento prepara os *blocos* que ela contém para uma escrita futura.

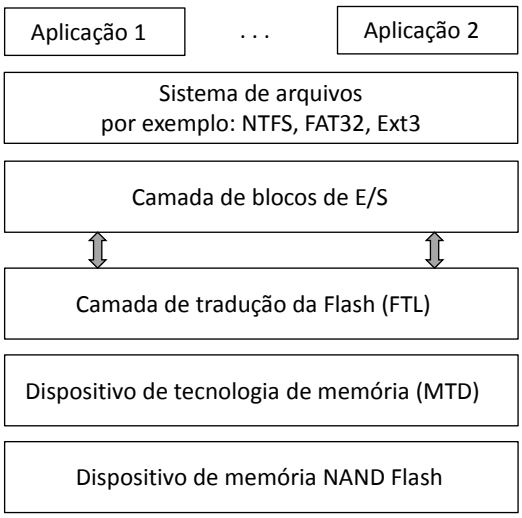


Figura 1 – Arquitetura de uma memória *flash* do tipo NAND

Apagar memória *flash* causa desgaste físico significativo [11]. Blocos de *flash* toleram entre 10^4 a 10^5 apagamentos antes que se tornem blocos não confiáveis [3]. Para promover uma vida mais longa do dispositivo, apagamentos devem ser nivelados uniformemente sobre os blocos de apagamento. Para atender estas características, um sistema de armazenamento *flash* normalmente consiste de duas camadas, a FTL (Flash Translation Layer) e a MTD (Memory Technology Device), como ilustrado na Figura 1. A função principal da FTL é redirecionar endereços lógicos do sistema de arquivos do sistema operacional para endereços físicos em *flash* NAND, utilizando uma tabela de mapeamento. FTL também provê componentes úteis como *garbage collector* e *wear-level* para otimizar o espaço utilizado e manter o mesmo nível de desgaste para cada bloco na *flash* NAND. A camada MTD implementa as funções primitivas na memória *flash*, funções de escrever, ler e apagar.

Memória *flash* apenas permite dois estados: apagado e não apagado. No estado de apagado, um byte de uma memória flash NAND tem todos os bits ligados (0xFF). Um bit em "1" (apagado) pode ser escrito e transformado em "0" (não apagado), mas não pode voltar para um. Para retornar o bit para o seu estado de apagado, um bloco significativamente maior da mídia *flash*, chamado de unidade de apagamento deve ser apagada. A FTL torna estes detalhes transparentes para o sistema de arquivos e remapeia os dados passados para ela escrevendo em áreas de dados não utilizadas na mídia *flash*. Isto apresenta a ilusão para o sistema operacional que um bloco de dados é simplesmente substituído quando ele é modificado quando, na realidade, os novos dados foram gravados em outro lugar na mídia. A FTL também cuida de recuperar os blocos de dados descartados para reutilização.

Um bloco de memória *flash* é a unidade básica para a operação de leitura/escrita, enquanto uma unidade é a quantidade mínima para uma operação de apagamento. O desempenho destas três operações é bastante diferente, como mostrado na Tabela 1.

© 2017 Sociedade Brasileira de Computação

Uma operação de apagamento leva muito mais tempo do que uma escrita ou de leitura. Com a realização de operações de escrita na memória *flash*, o espaço livre diminui e operações de coleta de lixo são invocadas para recuperar algum espaço livre para reutilização.

Tabela 1 – Especificações de Memória flash NAND [15]

Características	SLC NAND Flash	MLC NAND Flash
Leitura	30 μs	250 μs
Escrita	250 μs	2700 μs
Apagamento	3ms	4ms
Tamanho de bloco	4.328 Bytes	8.568 Bytes
Tamanho de unidade	270,5 KBytes	1.606,5 KBytes

3. Métodos para remoção segura

O ponto deste trabalho é a eliminação segura de arquivos. Como um sistema operacional faz a remoção de um arquivo simplesmente liberando os blocos que o arquivo ocupava em disco (*deletion by unlinking*), a sua recuperação com ferramentas forenses é possível, desde que os blocos ainda não tenham sido alocados e sobrescritos por outro arquivo. Para a eliminação segura, ou seja, para impedir que um arquivo seja recuperado por técnicas forenses, três métodos básicos que podem ser utilizados:

1. Uso de um sistema de arquivos seguro, que proteja todos os blocos do sistema de arquivos, tanto aqueles que estejam em uso como aqueles que foram liberados porque um arquivo foi removido.
2. Uso de criptografia para realizar a eliminação, onde um arquivo é armazenado de forma criptografada e a sua remoção é realizada eliminando-se a chave de criptografia.
3. Uso de sobrescrita para realizar a eliminação, onde os blocos pertencentes a um arquivo que deve ser removido são sobrescritos com padrões binários.

Como não foi objetivo implementar ou analisar sistemas de arquivos seguros, mas sim realizar a remoção segura de arquivos, limitou-se a pesquisa a métodos de eliminação por sobrescrita.

O problema causado por dados que permanecem no meio de armazenamento foi descoberto pela primeira vez em meio magnético [7]. Mesmo se a informação é sobrescrita várias vezes em discos e fitas, ainda pode ser possível extrair os dados iniciais. Isto levou ao desenvolvimento de métodos especiais para a remoção de informações confidenciais de forma confiável para a mídia magnética.

Pelas características da FTL, remover blocos por sobrescrita não é uma técnica efetiva, uma vez que o mapeamento da FTL redirecionaria a operação de sobrescrita para outro bloco lógico, e não para o bloco físico que se deseja sobrescrever. Entretanto, realiza-se aqui uma análise para este caso, pois o seu custo fornece um valor teórico ideal para métodos de sobrescrita.

Independente do sistema de arquivos utilizado pelo sistema operacional, um arquivo pode ser modelado como uma lista de blocos, onde um bloco é uma unidade de

 © 2017 Sociedade Brasileira de Computação

alocação, ou seja, a menor porção do meio de armazenamento que pode ser reservada para guardar um arquivo.

Para análise de custos são utilizadas as seguintes grandezas:

- L_t : Tempo de uma leitura (valor de referência básica).
- E_t : Tempo de uma escrita (varia de 8 a 11 vezes o tempo de uma leitura).
- A_t : Tempo de um apagamento (varia de 16 a 100 vezes o tempo de uma leitura).

Observe-se que o tempo de remoção do arquivo pelo sistema operacional depende do sistema de arquivos utilizado, mas é igual para todos os métodos, e portanto não será considerado neste trabalho.

Como uma memória *flash* é organizada em unidades de apagamento, os números de blocos descritos acima devem ser definidos para cada unidade de apagamento. Neste trabalho é utilizado um índice quando for necessário referenciar uma unidade de apagamento específica.

Técnicas de remoção segura que são geralmente utilizadas para discos magnéticos não são aplicáveis para a memória *flash*. Devido a essa limitação, existem alguns métodos existentes para a eliminação segura de memória *flash*, detalhados nas próximas seções.

3.1. Método de apagamento híbrido (Método de Sun)

Sun [14] propôs um método híbrido adaptativo que combina sobrescritas com zero (*zero-overwriting*) e apagamento de blocos (*erase blocks*). Sobrescritas com zero é um método de eliminação que substitui dados nos blocos por 0x00 para que os dados existentes possam ser apagados de forma segura. Em contraste, apagamento de blocos exclui todos os dados que são armazenados em uma unidade de apagamento através da realização de uma operação de apagamento. Neste momento, se blocos válidos permanecem nas unidades de apagamento, estes blocos devem ser transferidos para outra unidade de apagamento, são necessárias operações adicionais de modo de leitura/gravação.

O esquema híbrido adaptativo proposto por Sun [14] combina os dois métodos anteriores para minimizar o custo da operação, realizando os passos abaixo.

1. Procura um bloco a ser deletado.
2. Verifica se o custo da substituição por zeros (*zero-overwriting*) nos blocos deletados é mais barato do que apagar a unidade de apagamento (*erase blocks*) que contém os blocos excluídos. Se o custo de substituir é mais barato do que apagar a unidade de apagamento, a substituição com zeros é executada.
3. Caso contrário, os blocos válidos são copiados para outra unidade de apagamento e o apagamento da unidade é aplicado para excluir dados.

Assim, o custo de uma sobrescrita em uma unidade *i* é o tempo necessário para realizar uma escrita com *zero-overwriting*:

$$T_{zo}(i) = N_{blocos_a_remover}(i) * E_t$$

$$T_{zo}total = \Sigma\ T_{zo}(i)$$

© 2017 Sociedade Brasileira de Computação

O custo de apagamento é o tempo necessário para copiar os blocos ainda válidos para outra unidade de apagamento e a seguir apagar a unidade *i*:

$$T_A(i) = N_{blocos_válidos}(i) * (L_t + E_t) + A_t$$

$$T_A total = \Sigma\ T_A(i)$$

O método considera ainda um benefício associado à operação de apagamento, pois cria blocos livres que podem ser usados futuramente. Já uma operação de sobrescrita não cria nenhum bloco livre, então seu benefício futuro é zero:

$$Benefício_{zo}(i) = 0$$

$$Benefício_A(i) = (N_{blocos_ganhos}(i) / N_{blocos_unidade}) * A_t$$

onde $N_{blocos_ganhos}(i) = N_{blocos_unidade} - N_{blocos_válidos}(i)$

então $Benefício_A(i) = ((N_{blocos_unidade} - N_{blocos_válidos}(i)) / N_{blocos_unidade}) * A_t$

ou $Benefício_A(i) = (1 - (N_{blocos_válidos}(i) / N_{blocos_unidade})) * A_t$

Portanto, o benefício de um apagamento vai aumentando à medida que o número de blocos válidos na unidade de apagamento *i* vai diminuindo.

Se $N_{blocos_válidos}(i) = N_{blocos_unidade}$, então:

$$Benefício_A(i) = (1-(N_{blocos_válidos}(i)/N_{blocos_unidade})*A_t = (1-1)*A_t = 0$$

Por outro lado, se $N_{blocos_válidos}(i) = 0$, então:

$$Benefício_A(i) = (1 - (N_{blocos_válidos}(i) / N_{blocos_unidade}) * A_t = (1-0)*A_t = A_t$$

Ou seja, o benefício de um apagamento ($Benefício_A$) varia entre zero e A_t. Já o benefício de uma sobrescrita é sempre zero, como já foi dito anteriormente.

Assim, o custo de uma operação ou de outra é definido como o tempo necessário para realizar a operação, subtraído do benefício obtido. Para a sobrescrita, tem-se:

$$Custo_{zo}(i) = T_{zo}(i) - Benefício_{zo}(i)$$

$$Custo_{zo}(i) = T_{zo}(i) = N_{blocos_a_remover}(i) * E_t$$

E para o apagamento tem-se:

$$Custo_A(i) = T_A(i) - Benefício_A(i)$$

$$Custo_A(i) = (N_{blocos_válidos}(i)*(L_t+E_t)+A_t) - ((1 - (N_{blocos_válidos}(i)/N_{blocos_unidade})*A_t)$$

$$Custo_A(i) = N_{blocos_válidos}(i) * ((L_t + E_t) + A_t / N_{blocos_unidade})$$

Um uso simplificado do método de Sun seria somar os custos de sobrescrita e de apagamento de cada unidade de apagamento, e a seguir realizar para todas as unidades sempre sobrescrita ou sempre apagamento, baseado na operação de menor custo global. O método híbrido proposto por Sun, entretanto, realiza a comparação dos dois para cada unidade de apagamento *i*, e usa nesta unidade a operação de menor custo.

3.2. Método de apagamento duplo (Método de Huang)

Huang [10] propõe um método que utiliza duas passagens. Na primeira passagem, os blocos do arquivo são sobrescritos com zero (*zero overwriting*). Na segunda passagem,

 © 2017 Sociedade Brasileira de Computação

realiza-se o apagamento da unidade de apagamento que contém estes blocos, após copiar os blocos válidos (de outros arquivos) para uma nova posição. Huang realiza estas duas passagens para ficar em conformidade com os principais padrões sugeridos para limpeza de dados [4], [12].

Se por um lado o método segue os padrões internacionais para oferecer mais segurança na remoção de um arquivo, por outro lado este método sacrifica a vida útil de uma memória *flash* ao realizar mais operações de escrita e apagamento.

O método duplo de Huang [10] realiza tanto uma operação de sobrescrita como uma operação de apagamento, e portanto o seu tempo é dado pelo somatório das operações em todas unidades de apagamento que contêm blocos do arquivo a ser removido.

3.3. Métodos propostos (ZOP-EBP e ZO-EBP)

O método híbrido de remoção de arquivos proposto por Sun utiliza uma função de custo para decidir entre uma sobrescrita ou um apagamento. Os custos são basicamente calculados a partir do tempo necessário para realização das operações, e o método utiliza um fator de benefício para o apagamento, para considerar o fato de blocos apagados estarem prontos para serem reutilizados, enquanto o mesmo não ocorre com blocos sobrescritos.

Entretanto, o método de Sun não considera a existência de setores livres quando calcula o custo de um apagamento. Da mesma forma, também não considera que blocos removidos por sobrescrita deverão ser futuramente reciclados por um apagamento antes de estarem novamente disponíveis para uso. Assim, este trabalho propõe uma análise mais detalhada das funções de custo.

Na análise que Sun realiza do seu método, ele somente considera blocos a serem removidos e blocos válidos. Isto causa problemas no cálculo do custo do apagamento ($Custo_A$) se não existem blocos válidos e a unidade de apagamento somente contiver blocos a serem removidos e blocos livres. Nesta situação, o método de Sun considera que o $Custo_A$ é zero. Por considerar que blocos livres sendo apagados antes de serem escritos é um desperdício e acelera o desgaste da *flash*, propõe-se introduzir um fator de penalidade, proporcional ao número de blocos livres:

$$Penalidade_A(i) = N_{blocos_livres}(i) * E_t$$

Este fator indica que se perdeu a oportunidade de fazer N_{blocos_livres} escritas.

De forma semelhante, o método de Sun também tem problemas no cálculo do custo de sobrescrita ($Custo_{zo}$). Os blocos que são sobrescritos (com zeros) vão precisar passar por uma operação de apagamento no futuro, antes de poderem ser usados novamente. Assim, aqui propõe-se introduzir um fator de penalidade proporcional ao número de blocos sobrescritos:

$$Penalidade_{zo}(i) = (N_{blocos_a_remover}(i) / N_{blocos_unidade}) * A_t$$

Com estas alterações, o método de Sun continua sendo realizado da mesma forma, mas o custo de cada operação é acrescido de um fator de penalidade:

Custo = Tempo – Benefício + Penalidade

© 2017 Sociedade Brasileira de Computação

O método proposto é um método híbrido de Sobrescrita e Apagamento e utiliza uma função de custo para decidir entre sobrescrever com zeros ou apagar a unidade. Como resultado definiu-se para o método proposto os seguintes parâmetros:

- Para a operação de sobrescrita, o seu custo é igual ao tempo necessário para sobrescrever os blocos a serem removidos na unidade i:

  $Custo_{ZO}(i) = T_{zo} = N_{blocos_a_remover}(i) * E_t$

- Para a operação de sobrescrita, o seu custo inicial é igual ao tempo necessário para copiar os blocos válidos para outra unidade e a seguir realizar o apagamento da unidade i:

  $T_A = N_{blocos_válidos} * (L_t + E_t) + A_t$

- Do custo inicial do apagamento subtrai-se um fator de Benefício, para representar que o apagamento produziu blocos livres, com exceção dos blocos válidos copiados para outra unidade (ou seja, o apagamento poupou uma futura reciclagem):

  $Benefício_A(i) = ((N_{blocos_unidade} - N_{blocos_válidos}(i))/ N_{blocos_unidade}) * A_t$

- Entretanto, o apagamento é prejudicial se existirem blocos livres na unidade, então ao custo inicial deve ser somado um fator de penalidade:

  $Penalidade_A(i) = N_{blocos_livres}(i) * E_t$

- Assim, o custo de um apagamento é dado por:

  $Custo_A(i) = T_A - Benefício_A(i) + Penalidade_A(i)$

  ou $Custo_A(i) = N_{blocos_válidos}(i) * ((L_t + E_t) + A_t/N_{blocos_unidade}) + N_{blocos_livres} * E_t$

Para cada unidade de apagamento que contém blocos do arquivo a remover, calcula-se $Custo_{ZO}(i)$ e $Custo_A(i)$, e a seguir realiza-se a operação de menor custo associado. Estes métodos foram denominados de ZOP-EBP, que considera as duas penalidades, e ZO-EBP, que considera somente a penalidade de apagamento. A próxima seção apresenta experimentos descrevendo as vantagens e desvantagens de cada método proposto.

4. Experimentos

Para realizar uma análise quantitativa e qualitativa dos métodos estudados, bem como uma análise dos aperfeiçoamentos propostos foi implementado um simulador na linguagem de programação C. Este simulador foi desenvolvido para simular as operações básicas de um sistema de arquivos, particularmente a criação e remoção de arquivos. Foram simuladas as operações da FTL, ou seja, mapeamento de blocos virtuais para físicos, gerência do estado de blocos físicos e *garbage collector*, assim como as operações sobre blocos físicos: leitura, escrita, sobrescrita e apagamento. Além disto, foram simulados os diversos métodos de remoção segura de arquivos.

Para comparar os vários métodos de remoção, são contabilizadas as operações de apagamentos, sobrescritas com zeros, blocos livres apagados, leituras e escritas de blocos, assim como a quantidade total de blocos operados em remoção de arquivo.

 © 2017 Sociedade Brasileira de Computação

Com o uso do Simulador foi possível validar os métodos de remoção em diferentes cenários de uso e com arquivos de diversos tamanhos. Para todas as simulações realizadas foram utilizados doze cenários distintos, que efetuavam a remoção de um total de 2.437 arquivos. Para os experimentos realizados, foram empregados os mesmos tempos de leitura de um bloco em 10 µs, escrita de um bloco em 50 µs e apagamento em 3000 µs, assim como 64 blocos por unidade de apagamento. Para cada método foram contabilizados os blocos copiados entre unidade de apagamento (leituras e escritas), as unidades apagadas (apagamentos), os blocos sobrescritos (sobrescritas com zeros), os blocos livres que foram prematuramente apagados (livres apagados) e o total de blocos operados, quer por leitura, escrita, sobrescrita ou apagamento (blocos operados).

A remoção por sobrescrita de zeros também marca os blocos físicos do arquivo como obsoletos, mas adicionalmente substitui os seus conteúdos com zeros, impedindo assim sua recuperação posterior. Desta maneira, o custo deste método de remoção corresponde a tantas operações de escrita quantos forem os blocos do arquivo.

A remoção por apagamento realiza uma operação de apagamento (*erase blocks*) na unidade de apagamento (*erase unit*) onde os blocos físicos estiverem localizados. Como esta unidade de apagamento também pode conter blocos de outros arquivos, estes blocos válidos devem ser previamente copiados para outra unidade de apagamento, antes da unidade atual poder ser apagada. Isto implica em uma série de operações de leitura e escrita (tantas quantas forem os blocos válidos a serem movidos) e posteriormente em uma operação de apagamento. Comparada com a remoção por sobrescrita de zeros, esta remoção por apagamento é potencialmente de maior custo, mas apresenta a grande vantagem da unidade de apagamento ficar pronta para uso imediato e não depender de uma reciclagem futura. Por outro lado, a realização de um apagamento de forma imediata acelera o desgaste do meio físico.

Os métodos de sobrescrita com zeros (*zero-overwrite*) e de apagamento (*erase*) foram simulados somente para estabelecer os limites superiores das operações de sobrescrita e de apagamento, respectivamente. A partir destes limites pode-se estimar o desempenho de cada método, ou seja, as reduções nas quantidades de operações necessárias.

Tabela 2 – Resultados da simulação do refinamento do método proposto

Método	Apagamentos		Sobrescritas com zeros		Livres apagados		Blocos operados	
Zero-over	1271	20,89%	220260	100%	0	0%	301604	37,86%
Erase	6083	100%	0	0%	75563	100%	576290	72,35%
Sun	4510	74,14%	21067	9,56%	64111	84,84%	318857	40,03%
ZOP-EBP	3889	63,93%	21701	9,85%	18712	24,78%	291371	36,58%
ZO-EBP	3575	58,77%	34380	15,61%	4449	5,89%	272262	34,18%
Huang	6083	100%	220260	100%	75563	100%	796550	100%

O método de Sun, em função do cálculo de sua função de custo, realizou 21.067 sobrescritas com zero, e 4.510 apagamentos de unidades. Os apagamentos realizados reciclaram prematuramente 64.111 blocos livres. Desta forma, o método de Sun apresenta um desempenho intermediário entre o mínimo determinado pela sobrescrita e o máximo determinado pelo apagamento. Este desempenho é esperado, pois para cada

© 2017 Sociedade Brasileira de Computação

bloco o método de Sun aplica uma função de custo para escolher entre sobrescrita e apagamento.

O método proposto ZOP-EBP aplica uma penalidade no cálculo do custo da sobrescrita e no custo do apagamento, com o objetivo de reduzir a quantidade de blocos livres apagados prematuramente. Este objetivo é atingido, pois a quantidade total de blocos livres apagados foi de 18.712, contra 64.111 do método de Sun. Com a penalidade aplicada sobre o custo de um apagamento, a quantidade total de apagamentos foi reduzida de 4.510 (pelo método de Sun) para 3.889, o que explica a redução observada no apagamento de blocos livres, pois a penalidade proposta visa justamente fornecer um custo favorável à sobrescrita (sobre o apagamento) quando existem muitos blocos livres na unidade a ser apagada. Em comparação com o método de Sun, a quantidade de blocos sobrescritos com zero aumentou levemente (21.701 contra 21.067 de Sun), o que se explica pela redução observada na quantidade de apagamentos realizados. Mesmo assim, a quantidade total de operações realizadas foi de 291.371, o que é menor que as operações requeridas pelo método de Sun (318.857 operações sobre blocos).

O método de Huang realiza tanto sobrescrita com zeros como apagamento, e portanto apresenta um resultado que é a soma dos métodos de sobrescrita (220.260 blocos sobrescritos com zeros) e apagamento (6.083 unidades apagadas). Por causa disto, é o método que envolve a maior quantidade de blocos operados, com um total de 796.550 blocos.

Apesar do desempenho do método proposto ZOP-EBP ser superior ao do método de Sun, principalmente na redução de blocos livres apagados, o grande aumento na quantidade de blocos copiados, ou seja, lidos de uma unidade a ser apagada e escritos em outra unidade disponível, levou à necessidade de uma análise mais completa dos fatores de benefício e de penalidade utilizados no cálculo do custo das operações de sobrescrita e de apagamento. Como ambas as penalidades forneciam um fator que era adicionado ao custo das duas operações, decidiu-se investigar o efeito de não utilizar a penalidade para a operação de sobrescrita com zeros, e somente adicionar a penalidade para a operação de apagamento, pois esta operação é a responsável tanto pela quantidade de cópias de blocos (blocos válidos precisam ser copiados para outra unidade antes do apagamento) como pelo apagamento de blocos livres. O método proposto originalmente é identificado pela sigla ZOP-EBP, onde a letra P indica a Penalidade tanto na sobrescrita com zeros (ZO) como no apagamento (E). O Segundo método é identificado pela sigla ZO-EBP, para indicar que o custo de uma sobrescrita não é penalizado, enquanto o custo de um apagamento é calculado com o benefício sugerido por Sun (letra B) e com a penalidade proposta (letra P).

Como pode ser verificado na Tabela 2, uma análise comparativa do método de Sun, o primeiro método proposto (sigla ZOP-EBP) e o segundo método proposto (sigla ZO-EBP) mostra que o método ZO-EBP realizou 3.575 apagamentos, o que é uma redução significativa em relação ao método de Sun (4.510 apagamentos) e também uma redução em relação ao método ZOP-EBP.

O método ZO-EBP foi, entre os três métodos, o que realizou a menor quantidade de apagamento de blocos livres. Foram apagados somente 4.449 blocos livres, contra 64.111 blocos apagados pelo método de Sun, e 18.712 blocos apagados pelo método

 © 2017 Sociedade Brasileira de Computação

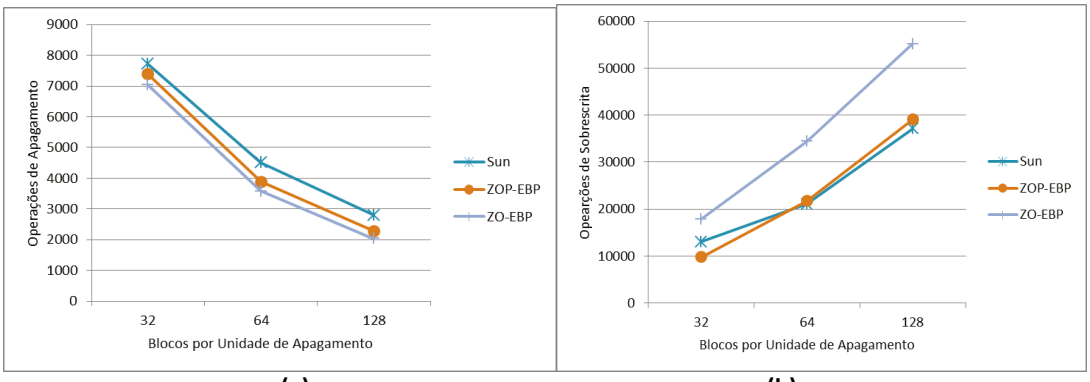
ZOP-EBP. O método ZO-EBP, entretanto, apresentou um aumento considerável das operações de sobrescrita, realizando 34.380 sobrescritas com zeros, contra 21.067 sobrescritas do método de Sun e 21.701 do método ZOP-EBP. Este aumento era esperado, pois a retirada do fator de penalidade da operação de sobrescrita faz com que o seu custo fique potencialmente menor do que o de uma operação de apagamento, e sobrescritas sejam realizadas mais vezes. Como este aumento é uma consequência da redução observada nas operações de apagamento, o resultado geral pode ser considerado benéfico, pois apagamentos implicam no desgaste da mídia e redução da sua vida útil.

Finalmente, o método ZO-EBP também foi o que realizou a menor quantidade de operações sobre blocos. Foram 272.262 blocos operados, contra 318.857 blocos do método de Sun e 291.371 blocos do método ZOP-EBP. Como o limite inferior de blocos operados é de 220.260, definido pelo método de sobrescrita, o método ZO-EBP foi dentre os três o que mais se aproximou deste limite.

Por estes resultados, o novo método proposto ZO-EBP foi o que apresentou os melhores resultados. Em termos percentuais, é o que apresentou a maior redução nas operações de apagamento (58,77%) e apagou o menor percentual de blocos livres (5,89%). Perde para o método de Sun e o método ZOP-EB em relação à redução de sobrescritas (15,61% contra 9,56% e 9,85%), mas esta desvantagem é compensada por ser o método que operou sobre o menor percentual de blocos (34,18%).

4.1 Análise da influência do tamanho da unidade de apagamento

Memórias *flash* atualmente têm unidades de apagamento que agrupam 32, 64 ou 128 blocos, sendo que 32 blocos por unidade são encontrados somente em memórias de pequena capacidade (abaixo de 128 MBytes). Para avaliar a influência do tamanho da unidade de apagamento no desempenho dos diversos métodos, foram simuladas as mesmas operações de remoção de arquivos para tamanhos de 32, 64 e 128 blocos por unidade de apagamento.



(a) **(b)**

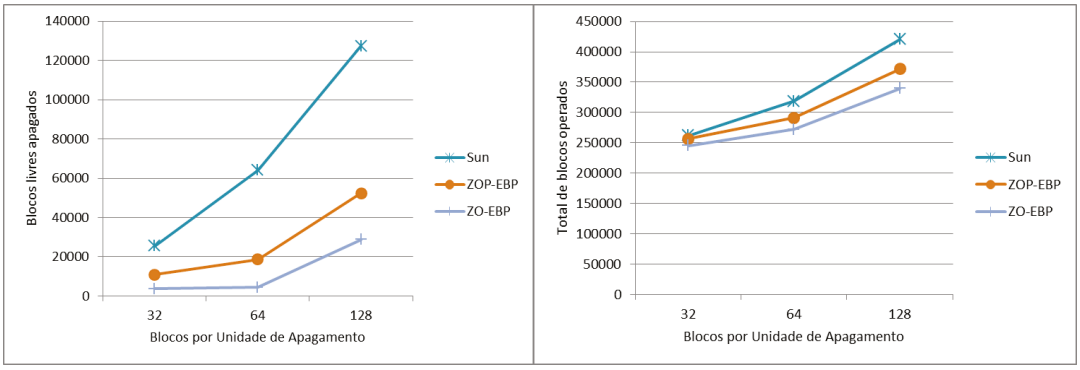
Figura 2 – Quantidade de apagamentos e sobrescritas por tamanho de unidade

Como pode ser observado na Figura 2a, a quantidade de apagamentos realizada diminui conforme o tamanho da unidade aumenta. Este resultado é esperado, pois se a unidade de apagamento contém mais blocos, os arquivos de mesmo tamanho tendem a ocupar menos unidades de apagamento. Por outro lado, como visto na Figura 2b, a quantidade de blocos sobrescritos aumenta conforme o tamanho da unidade também aumenta. Este resultado comprova que os métodos garantem um bom balanceamento entre apagamentos e sobrescritas, pois a redução das operações de apagamento implica

© 2017 Sociedade Brasileira de Computação

em um aumento correspondente nas operações de sobrescrita. Todos os métodos apresentam o mesmo comportamento conforme o tamanho da unidade de apagamento aumenta.

A Figura 3a mostra a quantidade de blocos livres que foram apagados antes de serem utilizados por um arquivo. Como pode ser observado, o método de Sun é o que desperdiça a maior quantidade de blocos livres. Esta característica era esperada, pois o método não considera a existência de blocos livres ao calcular o custo de cada operação. Já o fator de Penalidade introduzido nos métodos propostos considera uma desvantagem apagar blocos livres, e em consequência os métodos "EBP" apresentam desempenho melhor neste aspecto. Embora os métodos apresentem o mesmo comportamento, pode-se verificar que o método de Sun apresenta uma quantidade significativamente maior de blocos livres apagados. A Figura 3b mostra a quantidade total de setores operados (apagados, sobrescritos ou copiados) em cada um dos métodos. Como pode ser visto, esta quantidade aumenta em função do tamanho da unidade de apagamento. Todos os métodos variam da mesma forma, e o método de Sun tem o maior aumento, pelo fato de apagar uma quantidade significativa de setores livres.



(a) (b)

Figura 3 – Blocos livres apagados e blocos operados por tamanho de unidade

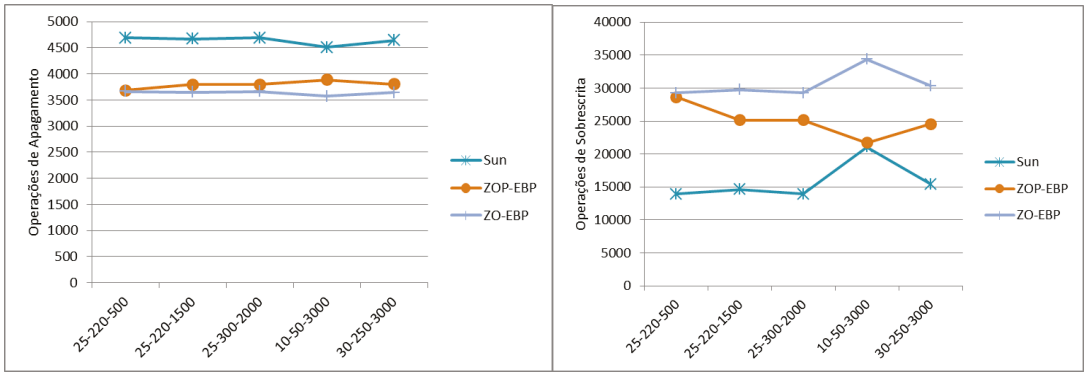
4.2 Análise da influência do tempo das operações

Os três métodos escolhidos também devem ser analisados em função dos tempos das operações de uma memória *flash*, ou seja, os tempos de leitura de um bloco, de escrita de um bloco e de apagamento de uma unidade. Para realizar esta análise, as remoções de arquivos foram simuladas para cinco características distintas das memórias *flash*:

- Leitura de 25 µs, escrita de 220 µs e apagamento de 500 µs (25-220-500)
- Leitura de 25 µs, escrita de 220 µs e apagamento de 1500 µs (25-220-1500)
- Leitura de 25 µs, escrita de 300 µs e apagamento de 2000 µs (25-220-500)
- Leitura de 10 µs, escrita de 50 µs e apagamento de 3000 µs (10-50-3000)
- Leitura de 30 µs, escrita de 250 µs e apagamento de 3000 µs (30-250-3000)

A Figura 4a mostra a quantidade de operações de apagamentos realizadas por cada um dos três métodos. Não são notadas variações significativas em cada um dos métodos, indicando que os métodos permanecem relativamente constantes em função da variação dos tempos das operações.

© 2017 Sociedade Brasileira de Computação

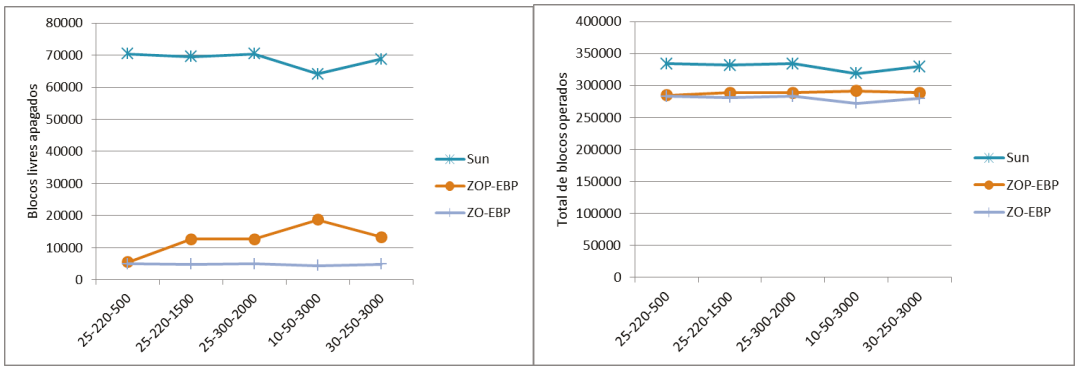


(a) (b)

Figura 4 – Quantidade de apagamentos e sobrescrita por tempos de operações

Na Figura 4b são mostradas as quantidades de operações de sobrescrita de blocos realizadas por cada um dos três métodos. Em contraste com o número de operações de apagamento, agora existem variações significativas em alguns métodos.

Em relação à quantidade de blocos livres apagados prematuramente pelo apagamento da unidade em que se encontram, a Figura 5a mostra como se comportam cada um dos métodos. O método de Sun não sofre muita variação, mas é o que mais desperdiça blocos livres, justamente por não considerar este tipo de bloco nas suas funções de custo. O método ZO-EBP é o mais independente dos tempos das operações, além de apresentar as menores quantidades de blocos livres que são apagados.



(a) (b)

Figura 5 – Blocos livres apagados e total de blocos operados por tempos de operações

A Figura 5b apresenta a quantidade total de blocos operados. Não se observam variações significativas em função dos tempos das operações.

5. Conclusão

Pelas análises realizadas, um método é considerado como tendo bom desempenho se apresenta um equilíbrio entre Sobrescritas e Apagamentos, realiza uma quantidade relativamente pequena de operações sobre blocos e minimiza a quantidade de blocos livres que são prematuramente apagados. O método inicialmente proposto, identificado pela sigla ZOP-EBP, foi refinado em função do desempenho observado no simulador, resultando no método ZO-EBP. Para decidir entre a realização de uma Sobrescrita ou um Apagamento, o método calcula o custo temporal da sobrescrita e compara com o custo temporal do apagamento, porém considerando que um apagamento tem o benefício de já realizar a reciclagem de blocos, mas recebe uma penalidade se for

© 2017 Sociedade Brasileira de Computação

aplicado a uma unidade de apagamento que contenha muitos blocos livres. Estes dois métodos apresentam desempenho melhor do que os métodos atualmente descritos na literatura especializada [10] e [14], além de, comparativamente, reduzirem o desgaste da mídia.

6. Referências

[1] Ban, A. "Flash file system". US Patent, no. 5404485, 1995.

[2] Bauer, S.; Priyantha, N. B. "Secure Data Deletion for Linux File Systems". In: Usenix Security Symposium, 2001, pp. 153–164.

[3] Breeuwsma, M.; Jongh, M.; Klaver, C.; van der Knijff, R.; Roeloffs, M. "Forensic data recovery from flash memory". Small Scale Digital Device Forensics Journal, vol. 1, 2007, pp. 1-17.

[4] DOE. "Media Sanitization Manual". US Department of Energy, DOE M 205.1-6. Capturado em: <http://www.directives.doe.gov>, Junho 2016.

[5] Gal, E.; Toledo, S. "Algorithms and Data Structures for Flash Memories". ACM Computing Surveys, vol. 37, 2005, pp. 138–163.

[6] Garfinkel, S. "Anti-Forensics: Techniques, Detection and Countermeasures". In: 2nd International Conference on i-Warfare and Security, 2007, pp. 77-84

[7] Gutmann, P. "Secure deletion of data from magnetic and solid-state memory". In Proceedings of the 6th USENIX UNIX Security Symposium, 1996, pp. 77-90.

[8] Gutmann, P. "Data Remanence in Semiconductor Devices" In: Proceedings of the 10th conference on USENIX Security Symposium, 2001, pp. 4.

[9] Hao, F.; Clarke, D.; Zorzo, A.F. "Deleting secret data with public verifiability", IEEE Transactions on Dependable and Secure Computing, vol. 13, num. 6, 2016, pp. 617-629.

[10] Huang, N.; He, J.; Zhao, B. "Secure Data Sanitization for Android Device Users". International Journal of Security and its Applications, vol. 9(5), 2015, pp. 61-68.

[11] Micron Technology, Inc. "Technical Note: Design and Use Considerations for NAND Flash Memory". 2006.

[12] NIST. "Guidelines for Media Sanitization". NIST Special Publication 800-88. Capturado em: <http://dx.doi.org/10.6028/NIST.SP.800-88r1>, Maio 2016.

[13] Peterson. Z.N.J.; Burns. R.; Herring. J.; Stubblefield. A.; Rubin. A.D. "Secure Deletion for a Versioning File System". In: Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies (FAST), 2005, vol. 4, pp. 143-154.

[14] Sun, K.; Choi, J.; Lee, D.; Noh, S. "Models and Design of an Adaptive Hybrid Scheme for Secure Deletion of Data in Consumer Electronics". IEEE Transactions on Consumer Electronics, vol. 54, 2008, pp. 100–104.

[15] Wang, Y.; Qin, Z.; Shao, Z.; Wang, Q.; Li, S.; Yang, L. T. "A Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems". IEEE Transactions on Multi-Scale Computing Systems, vol. 2, no. 1, January-March 2016, pp. 17-28.

[16] Wei, M.; Grupp, L. M.; Spada, F. M.; Swanson, S. "Reliably Erasing Data from Flash-Based Solid State Drives". In: Proceedings of the 9th USENIX conference on File and Storage Technologies, Berkeley, CA, USA, 2011, pp. 105–117.

 © 2017 Sociedade Brasileira de Computação

Um método de identificação de navegadores *Web* baseado na *Web Audio* API

Jordan S. Queiroz[1], **Eduardo L. Feitosa**[1]

[1]Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM)
CEP 69.077-000 – Manaus – AM – Brasil

{jsq,efeitosa}@icomp.ufam.edu.br

Abstract. *Web fingerprinting is a process in which the user is, with high likelihood, uniquely identified by the extracted characteristics of his / her device, generating an identifier key (fingerprint). Although it can be used for malicious purposes, Web fingerprinting can also be used for good intention: enhance usability in Web pages, enhance two-factor authentication and so on. This paper investigates the Web Audio API as a Web fingerprinting method capable of identifying the devices' class. As outcome, it is found that the method is capable of identifying the device's class, based on features like device's type, Web browser's version and rendering engine.*

Resumo. *Web fingerprinting é o processo no qual o usuário é, com alta probabilidade, identificado de forma única a partir das características extraídas de seu dispositivo, gerando uma chave identificadora (fingerprint). Embora possa ser usado para propósitos maliciosos, Web fingerprinting também pode ser usado com boas intensões: melhorar usabilidade em páginas Web, melhorar a autenticação de dois fatores e assim por diante. Esse trabalho investiga a Web Audio API como um método de Web Fingerprinting capaz de identificar a classe do dispositivo. Como resultado, foi descoberto que o método é capaz de identificar a classe do dispositivo, com base em características como tipo do dispositivo, versão e motor de renderização do navegador Web.*

1. Introdução

É fato que a *Web* é o meio mais utilizado para a realização de diferentes atividades e que todo esse conjunto de possibilidades foi alcançado devido ao rápido avanço das tecnologias *Web* (linguagens, APIs, mecanismos, entre outros) que proporcionam experiências de uso cada vez melhores. Por outro lado, essas mesmas tecnologias também podem ser empregadas para violar a privacidade dos usuários na *Web*, permitindo que os mesmos possam ser rastreados e identificados [Laperdrix et al. 2016]. Um dos meios empregados é coletar características discriminatórias presentes nos dispositivos dos usuários. Dessa forma, partindo do pressuposto que cada usuário possui seu próprio dispositivo, pode-se identificar unicamente um dispositivo e, consequentemente, seu dono. Esse processo é denominado de *Web fingerprinting*.

Existem na literatura diversos métodos de *Web fingerprinting* propostos [Bursztein et al. 2016, Laperdrix et al. 2016, Ximenes et al. 2016]. No entanto, muitos deles empregam tecnologias já consolidadas, por exemplo, os objetos *screen* e *navigator* providos pelo JavaScript. Esses objetos permitem a obtenção de informações relativas

 © 2017 Sociedade Brasileira de Computação

ao navegador *Web*, *plugins* instalados no mesmo e características relativas ao dispositivo, por exemplo, dimensão de tela [Khademi et al. 2015]. Outra tecnologia utilizada é o HTML5 Canvas, que possibilita realizar desenhos e obter os *pixels* dos mesmos de forma a identificar usuários. Embora esteja perdendo o suporte, o Flash também é usado em implementações de *Web Fingerpriting*, tendo em vista que essa tecnologia é capaz de prover informações mais estritas sobre o *hardware* do dispositivo, por exemplo, fontes instaladas no sistema [Khademi et al. 2015]. Além dessas tecnologias, a análise do cabeçalho HTTP também é empregada, tendo em vista que é possível obter o *User Agent*, dentre outros dados disponibilizados pelo cabeçalho.

Neste cenário, este trabalho investiga a utilização de características e tecnologias de áudio na identificação da classe dos dispositivos. O objetivo é desenvolver um método de *Web Fingerprinting*, empregando as interfaces da *Web Audio API* – especificamente *AudioContext* e *OfflineAudioContext*, visando a identificação da classe dos dispositivos e a futura incorporação em uma solução aplicada em um cenário real. De forma sucinta, a ideia é gerar ondas periódicas, obter suas amplitudes no domínio de tempo e, então, identificar a classe dos dispositivos. A vantagem dessa abordagem é a possibilidade obter características relativas ao *hardware* do dispositivo.

No restante do artigo: a Seção 2 apresenta uma breve descrição sobre *Web Fingerprinting* e *Web Audio* API; a Seção 3 discorre sobre os trabalhos relacionados; a Seção 4 apresenta a visão geral e a implementação do método de *Web Fingerprinting* proposto; a Seção 5 apresenta os experimentos realizados para averiguar a efetividade do método proposto; a Seção 6 apresenta os resultados obtidos por meio dos experimentos; e, por fim, a Seção 7 apresenta as limitações e trabalhos futuros.

2. Web Fingerprinting e Web Audio API

Web fingerprinting é o processo no qual as características do dispositivo são extraídas para gerar uma chave identificadora (*fingerprint*) capaz de identificar, com alta probabilidade e de forma única, um usuário. Para tanto, seu funcionamento requer que o dispositivo possua capacidade de navegação por meio de navegadores *Web* (Chrome, Firefox, Edge, dentre outros) [Saraiva et al. 2014].

As características que podem ser obtidas por meio de um *Web fingerprinting* podem ser classificadas em duas categorias: *software* e *hardware*. As características relacionadas ao *software* são aquelas que dependem da configuração do navegador *Web*, tais como sistema operacional, versão e plataforma do navegador, dentre outras; e do seu ambiente de execução, como extensões, *plugins*, tipos *MIME* e fontes instaladas. O problema com essas características é a facilidade e a frequência com que são alteradas, devido a, por exemplo, atualização do navegador, instalação de novos *plugins*, novas fontes e uso de extensões de contramedida, entre outros. Em outras palavras, sua volatilidade. Já as características relacionadas ao *hardware* são aquelas que dependem dos periféricos e até mesmo dos *drivers* instalados no dispositivo. Dentre alguns exemplos, destacam-se: monitor/tela do dispositivo, placa gráfica, *drivers* e placa de áudio. Diferente das características de *software*, as características de *hardware* têm menor volatilidade.

No que diz respeito a implementação de um *Web fingerprinting*, é possível fazê-lo empregando diferentes tecnologias. As mais consolidadas incluem o uso de JavaScript, Flash, HTML5 Canvas, *cookies*, Cabeçalhos HTTP, CSS e HTML. Contudo, é possível

 © 2017 Sociedade Brasileira de Computação

utilizar tecnologias ainda em estágio de desenvolvimento ou não padronizadas. Um bom exemplo é a *Web Audio API*.

De acordo com a W3C (*World Wide Web Consortium*), a *Web Audio* API é uma API em JavaScript de alto nível para o processamento e sintetização de áudio em aplicações *Web*. Ela define um conjunto de operações de áudio realizadas por meio de nós de áudio (*AudioNodes*) em contextos de áudio. Os nós são interligados entre si e formam um grafo, permitindo que o processamento do áudio siga um determinado fluxo. Dentre dos tipos de nós disponíveis, destacam-se: (I) fontes de áudio, que permitem gerar algum som ou carregar uma música; (II) filtros de efeitos de áudio, que permitem processar o áudio e adicionar efeitos; (III) destino de áudio, que geralmente são os alto-falantes do dispositivo; e (IV) nós de análise, que provêm dados sobre o áudio que está sendo processado.

Em relação à contextos de áudio, a *Web Audio API* implementa dois tipos: *AudioContext* e *OfflineAudioContext*. O *AudioContext* é um contexto de áudio que controla a criação de nós e a execução do processamento de áudio. O *OfflineAudioContext* tem função similar ao *AudioContext*, mas ao invés de renderizar o som diretamente no dispositivo do usuário, gera o som ou o decodifica em um *AudioBuffer*. Na prática, para escutar algum som produzido a partir de um nó oriundo (instanciado) do *OfflineAudioContext*, é necessário renderiza-lo por meio do *AudioContext*. No caso de nós oriundos do *AudioContext*, tal procedimento não é necessário.

A Figura 1 ilustra um exemplo simples do funcionamento do *AudioContext* (à esquerda) e do *OfflineAudioContext* (à direita). A Tabela 1 sumariza os nós de áudio que podem ser utilizados pelo *AudioContext* e *OfflineAudioContext*.

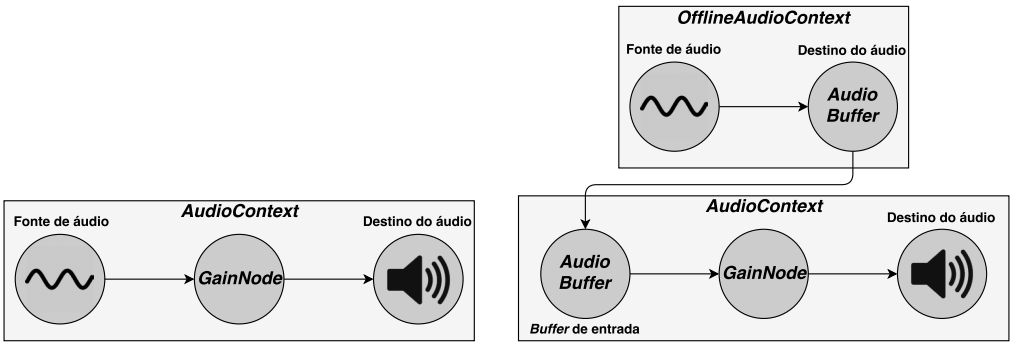


Figura 1. Exemplos de implementação do *AudioContext* (à esquerda) e do *OfflineAudioContext* (à direita).

Por também depender da placa de som do dispositivo, o *AudioContext* também está relacionado com as características de *hardware*. O processamento de áudio em diferentes máquinas e navegadores *Web* pode ser usado para obter características dos dispositivos devido a diversas combinações de *hardware* (placa de som) e *software* (*driver* do periférico) entre os mesmos, enquanto que a mesma combinação de dispositivo e navegador resultará em um *fingerprint* idêntico [Englehardt and Narayanan 2016]. Esse fenômeno pode ser atribuído à inconsistências durante o processo de fabricação dos *hardwares*, mesmo que seus modelos e número de versão sejam iguais. Essas inconsistências podem revelar, por exemplo, diferentes latências nas renderizações (no caso da

© 2017 Sociedade Brasileira de Computação

GPU) [Nakibly et al. 2015].

Tabela 1. Nós de áudio utilizados pelo *AudioContext* e *OfflineAudioContext* [MDN 2016].

Funções/Interfaces	Descrição
AudioNode	Interface que representa um módulo de processamento de áudio, por exemplo, uma fonte/destino de áudio ou módulo de processamento intermediário. O método *connect()* encontrado neste nó, permite conectar a saída de um nó *A* na entrada de um nó *B*.
OscillatorNode	Interface que representa um módulo de processamento de áudio que gera uma onda periódica (sinal) com determinada frequência em *hertz*. As ondas podem ser dos seguintes tipos: senoidal, quadrada, triangular e customizada.
GainNode	Interface que representa um módulo de processamento de áudio que possibilita a alteração do volume do áudio de entrada antes que o mesmo se propague para outros nós.
AudioDestinationNode	Interface que representa o destino final de uma fonte de som em um grafo de áudio instanciado em um dado contexto, geralmente o destino é o alto-falante do dispositivo.
AnalyserNode	Interface que representa um nó capaz de prover, em tempo real, informações de análise no domínio da frequência e tempo, para o propósito de visualização de áudio. Este nó não necessita estar conectado à um nó de saída para prover os dados. Essa interface possui métodos que provêm dados sobre o áudio, dois deles são: (I) *getByteFrequencyData(Uint8Array)*: Copia os dados de frequência do sinal (no domínio da frequência) para vetor passado por parâmetro; (II) *getByteTimeDomainData(Uint8Array)*: Copia os dados de frequência do sinal (no domínio do tempo) para o vetor passado por parâmetro.
ScriptProcessorNode	É a interface que representa um módulo de processamento de áudio, permitindo gerar, processar ou analisar áudio com o JavaScript. Esse módulo está interligado a dois *AudioBuffers*, um contendo dados de entrada e o outro cujo conteúdo são os dados de saída. Esse módulo possui um manipulador de eventos (*onaudioprocess()*), que é disparado sempre que o *AudioBuffer* de entrada está pronto para ser processado.
AudioBuffer	É a interface que representa um pequeno recurso de áudio carregado em memória, obtido a partir de um áudio decodificado ou de dados (frequências) não tratados.

3. Trabalhos Relacionados

A literatura apresenta diversos trabalhos sobre *Web fingerprinting*, focando na identificação do navegador e/ou na classe do dispositivo. Alguns desses trabalhos são apresentados a seguir.

Mowery et al. [Mowery et al. 2011] implementaram e avaliaram um método de *Web Fingerprinting*, visando a identificação do navegador através da análise do desempenho e do tempo de execução do JavaScript. Para tanto, utilizaram uma combinação de 39 diferentes *benchmarks* JavaScript, bem conhecidos e bem estabelecidos, e geraram um *fingerprint* normalizado a partir de padrões de tempo de execução. Como resultado da classificação, dos 1015 casos de teste, 810 foram classificados corretamente, permitindo uma acurácia de 79.8% na identificação do navegador.

Mowery e Shacham [Mowery and Shacham 2012] propuseram um método de *Web fingerprinting* baseado na renderização de texto e gráfico em 3D no HTML5 Canvas. A ideia é que cada navegador possui um conjunto de fontes e, em certos casos, este é customizado pelo usuário. Assim, a combinação do HTML5 Canvas com fontes é capaz de extrair características discriminatórias que podem ser usadas para implementar um *Web fingerprinting*. Como resultado, o método proposto pelos autores identificou de forma única um total de 116 dispositivos do total de 294. A amostra de *fingerprint* obtida apresenta entropia de 5,73 *bits* para o Canvas nos contextos 2D e 3D. No caso do Canvas exclusivamente no contexto 2D, foi alcançada a entropia de grau 3,05 *bits*.

Mulazzani et al. [Mulazzani et al. 2013] propuseram um método de *Web Fingerprinting* que utiliza testes no motor do JavaScript para melhorar o estado-da-arte no que tange o gerenciamento de sessão HTTP, dificultando o roubo de sessão. Os autores implementaram *scripts*, baseados no padrão ECMAScript, para testar certos aspectos dos

 © 2017 Sociedade Brasileira de Computação

navegadores e observaram aqueles que falharam em cada teste. Como resultado, os autores listaram os navegadores *Web* mais populares, bem como o número de testes em que os mesmos são reprovados e, dessa maneira, comprovaram que é possível identificar um navegador *Web* empregando testes de conformidade.

Khademi et al. [Khademi et al. 2015] realizaram uma análise de métodos de *Web Fingerprinting* (objetos do JavaScript, Flash, detecção de fontes por JavaScript e Canvas *fingerprinting*) para determinar o fluxo que os *scripts* de *Web Fingerprinting* seguem e, dessa forma, definir métricas para detectar tentativas de *fingerprinting* por parte de interessados. Os autores desenvolveram um *script* que emprega diversos atributos obtidos por meio dos métodos citados anteriormente. Como resultado, foram coletados 1.523 *fingerprints*, sendo que 90,41% deles são únicos. No entanto, o HTML5 Canvas colaborou com apenas 0,05% da unicidade, apresentando uma entropia de grau 3,21 bits.

Laperdrix et al. [Laperdrix et al. 2016] projetaram e avaliaram uma técnica de *Web fingerprinting* que emprega novos recursos do HTML5, com o objetivo de averiguar se contribuem significativamente na identificação de usuários. Dessa maneira, os autores exploraram a extração das características propostas por Eckersley [Eckersley 2010] (resolução de tela, *user agent*, lista de *plugins*, dentre outros) e adicionaram sete atributos (*WebGL vendor*, *WebGL renderer*, Canvas, dentre outros) que surgiram em decorrência da evolução do HTML5 e de suas APIs. Nos experimentos realizados foram coletados 118.000 *fingerprints*, sendo que 89,4% deles são únicos.

Bursztein et al. [Bursztein et al. 2016] desenvolveram e avaliaram uma técnica de *Web fingerprinting* para detectar e prevenir ataques automatizados oriundos de clientes maliciosos em lojas de aplicativos. O objetivo era distinguir clientes autênticos daqueles que são emulados ou automatizados. Diferentemente dos outros trabalhos encontrados na literatura, o método determina a classe do dispositivo, sendo composta pela tupla $\{$*navegador Web; SO*$\}$. Como resultado, o método proposto foi capaz de determinar com 100% de acurácia a classe de 52 milhões de dispositivos.

3.1. Discussão

Os trabalhos relacionados demonstram que *Web fingerprinting* é empregado tanto para identificar usuários quanto para identificar a classe de seus dispositivos. No que tange a identificação da classe, três métodos são utilizados: performance do JavaScript, conformidade do JavaScript e desenhos no HTML5 Canvas. Embora eficaz, o teste de performance não é eficiente, pois pode exigir recursos do navegador de forma a comprometer a experiência de navegação na *Web*, além do tempo de execução ser maior do que os de outros trabalhos encontrados na literatura. Já o teste de conformidade apresenta-se como eficaz e eficiente, mas é instável uma vez que um navegador (aprovado ou reprovado no teste) possa mudar de estado a medida que seja atualizado. O método que emprega HTML5 Canvas cumpre o objetivo, mas apenas recursos de vídeo foram explorados.

Ainda não há na literatura um método que emprega recursos de áudio para identificar classes de dispositivos.

4. Proposta e Implementação

Esta Seção apresenta a proposta do método e explica seus elementos, arquitetura e funcionamento.

© 2017 Sociedade Brasileira de Computação

4.1. Visão Geral

A Figura 2 ilustra, em alto nível, a visão geral da proposta.

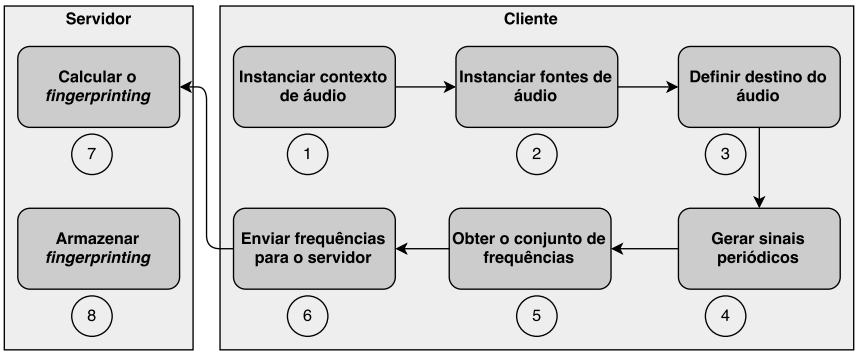


Figura 2. Visão geral de fluxo do método proposto.

O método proposto consiste em: Instanciar, no lado do cliente, um contexto de áudio **(1)**, criado a partir do *AudioContext* ou do *OfflineAudioContext*; Instanciar fontes de áudio **(2)**, que podem ser representadas por um oscilador (*OscillatorNode*) – capaz de gerar ondas periódicas (por exemplo, senoidal, quadrada e triangular) – ou por arquivos de áudio (música em mp3, por exemplo); **(3)** Definir o destino do áudio, que geralmente são os alto-falantes; **(4)** Gerar sinais periódicos – no caso do *OscillatorNode*, basta usar o método *start()* e no caso de arquivos de áudio é necessário fazer a decodificação dos mesmos; **(5)** Obter o conjunto de frequências (características) das ondas periódicas ou músicas, que pode ser obtido por métodos específicos ou pela leitura do *AudioBuffer* de saída; **(6)** Enviar as frequências para o servidor; **(7)** Calcular o *fingerprint* a partir das características extraídas, por exemplo, com o uso de funções de *hashing*; e, por fim, **(8)** Armazenar o *fingerprint* no banco de dados para análises de resultados.

Ao final do processo, as características obtidas como resultado serão enviadas para o servidor. Os passos 1 à 4 resultarão em sinais que serão processados. Parte do processamento consiste em calcular[1] a FFT (*Fast Fourier Transform*) dos mesmos, cujos resultados serão as características extraídas do dispositivo, capaz de identificar a classe do mesmo.

4.2. Implementação

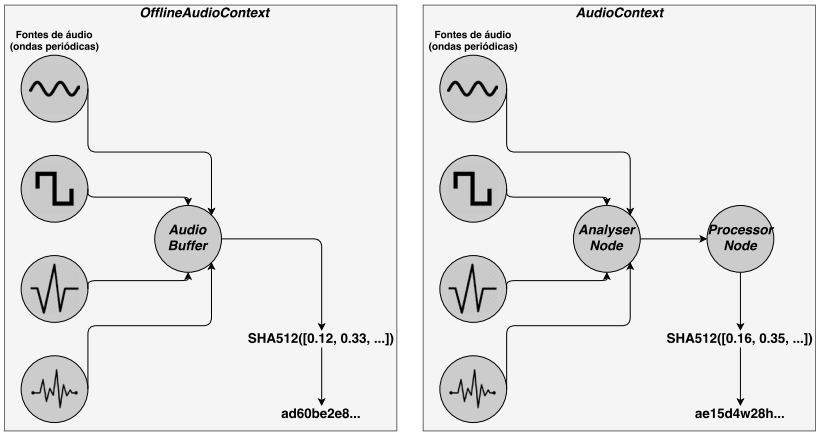
A Figura 3 ilustra os fluxos de processamento de áudio que podem ser usados para implementar o método de *Web fingerprint* proposto.

No caso do *OfflineAudioContext* (Figura 3 à esquerda), quando uma onda é gerada, sua FFT é calculada e o resultado é passado para um *AudioBuffer*, onde o sinal periódico fica armazenado. Após todos os sinais estarem em seus respectivos *AudioBuffers*, as primeiras 2048 frequências são coletadas e enviadas para o servidor, onde será calculado o *fingerprint* com o emprego da função de *hashing* SHA512.

Quando se trata do *AudioContext*, o processo é similar, mas diferente em aspectos relacionados a topologia do fluxo de áudio. Quando os sinais periódicos são gerados, os mesmos são passados para o *AnalyserNode* e para o *ScriptProcessorNode*, onde a FFT

[1] A *Web Audio* API se encarrega de fazer isso.

 © 2017 Sociedade Brasileira de Computação



(a) (b)

Figura 3. Arquitetura do método proposto a partir de dois pontos de vista: *OfflineAudioContext* (à esquerda) e *AudioContext* (à direita).

de cada sinal será calculada e os dados de análise dos sinais serão obtidos. Após este processo, um conjunto com 2048 elementos contendo os dados de análise é obtido e então enviado para o servidor, onde o *fingerprint* será calculado.

5. Experimentos

Essa Seção apresenta os experimentos que foram realizados para avaliar o método de *Web fingerprinting* proposto. Primeiramente, são descritos os experimentos preliminares realizados para avaliar qual interface (*AudioContext* ou *OfflineAudioContext*) é mais viável de se utilizar. Em seguida, é apresentado o experimento realizado com usuários voluntários.

5.1. *AudioContext* vs *OfflineAudioContext*

Uma vez que a *Web Audio API* possui duas interfaces disponíveis e, aparentemente, aptas a serem empregadas em um *Web fingerprinting*, qual delas utilizar? Com este questionamento em mente, decidiu-se realizar uma série de experimentos envolvendo os recursos disponíveis em cada uma das interfaces. Desta forma, foram definidos seis (6) cenários para experimentação, conforme a Tabela 2.

Tabela 2. Cenários dos experimentos.

Cenário	Conjunto de *AudioNodes*	Objetivo
CE1	Recurso de áudio + *AudioContext* + *Analyser*	As frequências obtidas por meio das funções do *AnalyserNode* são estáveis, considerando um áudio no formato mp3 ou ogg?
CE2	Recurso de áudio + *OfflineAudioContext*	As frequências obtidas por meio do *AudioBuffer* do *OfflineAudioContext* são estáveis, considerando um áudio no formato mp3 ou ogg?
CE3	Sinal + *OfflineAudioContext*	As frequências obtidas por meio do *AudioBuffer* do *OfflineAudioContext* são estáveis, considerando as ondas periódicas?
CE4	Sinal + *OfflineAudioContext* + *Analyser*	Depois de obter as frequências estáveis por meio do *AudioBuffer* do *OfflineAudioContext*, as mesmas vão continuar estáveis após serem reobtidas por meio das funções do *AnalyserNode*?
CE5	Sinal + *AudioContext* + *Analyser*	As frequências obtidas por meio das funções do *AnalyserNode* são estáveis, considerando as ondas periódicas?
CE6	Sinal + *ScriptProcessorNode*	As frequências obtidas por meio do *AudioBuffer* de entrada do *ScriptProcessorNode* são estáveis, considerando as ondas periódicas?

© 2017 Sociedade Brasileira de Computação

Todos os experimentos foram realizados em ambiente controlado com um número de dispositivos limitado, sendo 1 *notebook* (Linux/Ubuntu 16.04 LTS), 1 *desktop* (Windows 10), 1 *smartphone* (Android 6) e uma máquina virtual[2] (Mac OS X 10.11). O processo de experimentação ocorreu da seguinte maneira: A página do experimento[3] contendo o *script* de *fingerprinting* que emprega a *Web Audio API* foi acessada por todos os navegadores em cada um dos dispositivos. Em cada acesso, foram testados os quatro tipos de ondas periódicas (senoidal, quadrada, triangular e customizada, como ilustra a Figura 4), com intervalo de 5 segundos entre cada onda e de 25 segundos para se iniciar um novo experimento. Esse acesso (experimento) à página foi realizado 50 vezes para cada navegador. Os navegadores utilizados foram: Chrome 59, Firefox 54, Edge 15, Safari 10 e Opera 46.

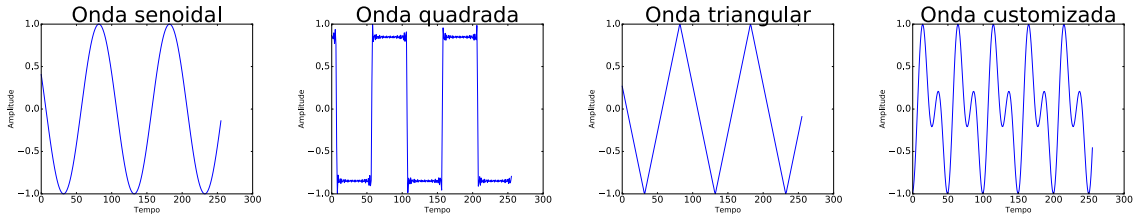


Figura 4. Ondas periódicas.

O primeiro experimento (CE1) consistiu em carregar um recurso de áudio no navegador, decodificá-lo, renderizá-lo no dispositivo e então armazenar todos os *AudioBuffers* em uma lista. Ao final, o *AudioBuffer* é recuperado e enviado para o servidor, onde será calculado o *fingerprint* com o emprego da função de *hashing* SHA512. O segundo experimento (CE2) consistiu em carregar um recurso de áudio no navegador, decodificá-lo, enviá-lo para um *AudioBuffer* e, por fim, renderizá-lo no dispositivo, fazendo com que o som fosse emitido. Embora similar ao primeiro experimento, as frequências do *AudioBuffer* foram alimentadas pelo *OfflineAudioContext* e não pelo *AudioContext*. A Figura 5 ilustra os resultado dos experimentos CE1 e CE2.

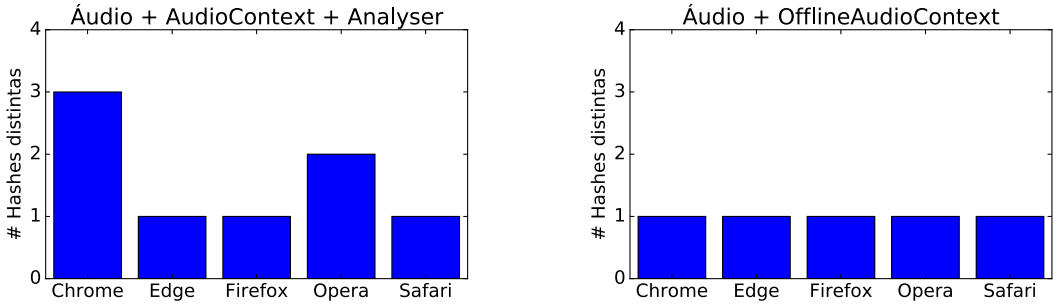


Figura 5. Experimentos com os cenários CE1 e CE2.

Percebe-se no primeiro cenário (Figura 5 à esquerda) que o conjunto de frequências (resultante da FFT) não se mantém estável durante as repetições, refletindo em *hashes* diferentes para o mesmo navegador. O navegador Chrome, por exemplo, gerou

[2]Devido a falta de disponibilidade de uma máquina física.

[3]Por ser um experimento controlado, a página ficou hospedada em um servidor local.

© 2017 Sociedade Brasileira de Computação

3 *hashes* diferentes. Tal fato, demonstra a inviabilidade do uso destes elementos (componentes) na geração de um *fingerprint*. Já o cenário CE2 sempre gerou um único *hash* para cada navegador (Figura 5 à direita). Isso serve de indicativo que o *OfflineAudioContext* pode contribuir com artefatos de *Web Fingerprinting*.

O experimento CE3 (Figura 6 à esquerda) consistiu em gerar três ondas primárias e uma customizada em um *AudioBuffer* do *OfflineAudioContext*. Depois de geradas, um conjunto com 2048 frequências de cada onda periódica foi obtido, enviado para o servidor e, por fim, o *fingerprint* foi calculado com base no conjunto de frequências obtido. O experimento CE4 (Figura 6 à direita) consistiu em renderizar, diretamente no dispositivo do usuário, as ondas periódicas. Da mesma forma que no experimento CE3, um conjunto com 2048 frequências de cada onda foi obtido e então o *fingerprinting* foi calculado.

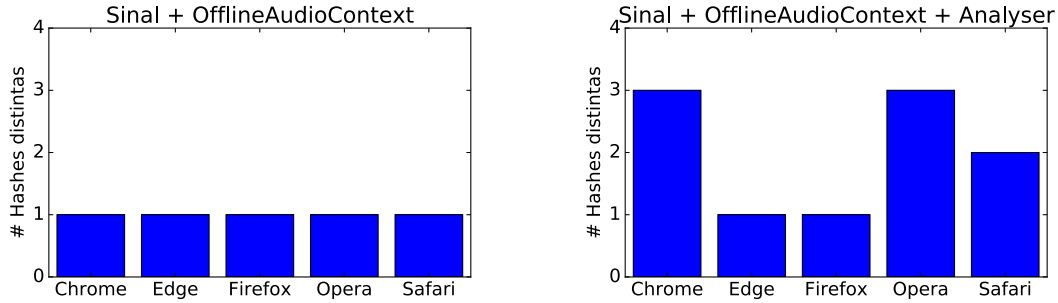


Figura 6. Experimentos com os cenários CE3 e CE4.

Percebe-se que no experimento CE3 todos os navegadores geram a mesma chave no decorrer das repetições. Tal comportamento é um indicativo de que o *OfflineAudioContext* pode ser empregado em métodos de *Web Fingerprinting*. Por outro lado, no experimento CE4, as frequências não se mantém as mesmas no correr de diversas repetições. Os navegadores Opera e Chrome, por exemplo, geraram 3 chaves distintas ao longo das 50 repetições, colocando em evidência que empregar o *AnalyserNode* para obter as frequências resulta em instabilidade, mesmo que a fonte de onde são obtidas seja estável.

O experimento CE5 consiste em renderizar, diretamente no dispositivo do usuário, três ondas primárias e uma customizada. Durante a renderização, suas frequências são obtidas a partir das funções providas pelo *AnalyserNode*. Após a extração, as ondas são enviadas para o servidor, local onde o *fingerprint* é calculado a partir das frequências extraídas. Esse experimento é similar ao CE1, mas a diferença é que no lugar de um recurso de áudio, é um sinal que está sendo renderizado. O experimentos CE6, similarmente ao CE5, consiste em renderizar três ondas primárias e uma customizada. No entanto, durante a renderização, as frequências são obtidas por meio do *AudioBuffer* de entrada do *ScriptProcessorNode*.

Na análise dos resultados, constatou-se que tanto no experimento CE5 quanto no experimento CE6 os conjuntos de frequências são voláteis, como ilustrado na Figura 7 à esquerda e direita, respectivamente. Por exemplo, na Figura 7 à esquerda, o Chrome apresentou 5 chaves distintas ao longo das 50 repetições. Na Figura 7 à direita, o Edge e o Opera apresentaram 5 chaves distintas. Dessa maneira, é possível observar que empregar o *AudioContext* e seus elementos não é recomendado para implementar métodos de *Web*

© 2017 Sociedade Brasileira de Computação

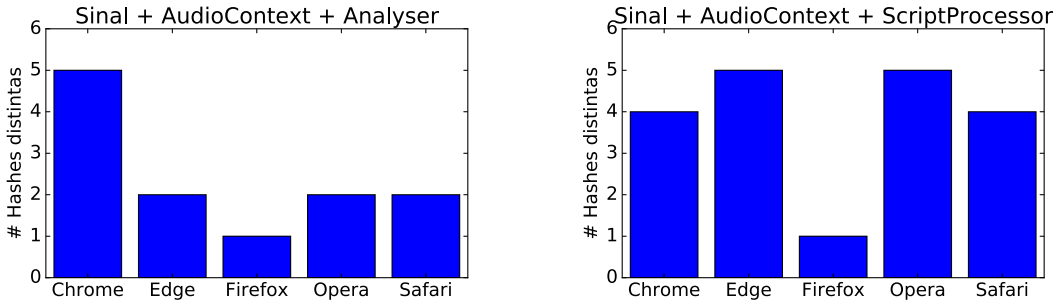


Figura 7. Experimentos com os cenários CE5 e CE6.

Fingerprinting.

Analisando os resultados obtidos por meio dos experimentos CE1 até CE6, pode-se perceber que o *AudioContext* e os nós oriundos do mesmo (por exemplo, o *AnalyserNode*) não demonstram contribuir na obtenção de *fingerprints* estáveis, provavelmente pelo fato do *AudioContext* trabalhar em tempo real, onde latências adicionais devido à processamentos podem ocasionar instabilidades [W3C 2017]. Por outro lado, o *OfflineAudioContext* mostra-se promissor e, por tanto, decidiu-se utilizar essa interface na implementação do *Web Fingerprinting* proposto neste trabalho. A investigação sobre o comportamento instável do *AudioContext* é um dos trabalhos futuros.

5.2. Experimentos com o *OfflineAudioContext*

Para comprovar a efetividade do *OfflineAudioContext*, um *script* que implementa o método proposto foi embarcado em uma página *Web*, que por sua vez foi hospedada em um servidor da Universidade, de forma que usuários pudessem participar do experimento. Em linhas gerais, quando a página é acessada, mostra-se uma mensagem explicando rapidamente o experimento e o propósito do mesmo. Além disso, um pequeno formulário deve ser preenchido com os seguintes campos: *email*, dispositivo (*Desktop/Notebook* ou *Smartphone/Tablet*, *SmartTV*, *Smartwatch*, *Game console*), e navegador (Chrome, Firefox, Opera, Safari, Internet Explorer, Edge, Chromium e outros). Esses dados informados pelos usuários tem como finalidade auxiliar na análise dos resultados. Além desses dados, o *User Agent*[4] também é coletado.

Com relação as ondas periódicas, são empregadas as mesmas descritas na Subseção 5.1, ilustradas na Figura 4. A proposta está fundamentada no fato de que navegadores e dispositivos distintos geram a mesma onda, mas com pequenas diferenças no conjunto de frequências [Englehardt and Narayanan 2016]. Essas diferenças podem ser usadas para identificar a classe dos dispositivos, que por sua vez é representada pelo conjunto $\{$*Navegador; Versão do navegador; Tipo do dispositivo*$\}$. As frequências de todas as ondas periódicas são concatenadas e então são passadas por parâmetro para a função de *hashing* SHA512, gerando a chave que caracteriza uma classe de dispositivo.

A dinâmica de funcionamento desse experimento é similar ao da Seção 5.1. A cada experimento foram geradas ondas periódicas, com intervalo de 5 segundos (com exceção da primeira). Logo depois, um subconjunto contendo 2048 amostras de suas

[4]O *User Agent* é usado apenas para fins de validação do formulário preenchido pelo usuário, ou seja, não é utilizado pelo método proposto.

 © 2017 Sociedade Brasileira de Computação

respectivas frequências foi obtido (para cada onda). Após mais 5 segundos, os conjuntos foram enviados para o servidor, onde ficaram armazenados para análises posteriores. O experimento possuiu duração aproximada de 20 segundos, para cada participante. Vale ressaltar que o intervalo de tempo para gerar as ondas, bem como o número de amostras, foram escolhidos de forma empírica e nenhum áudio foi emitido para os participantes.

6. Resultados e Discussão

A Figura 8 ilustra algumas características dos dispositivos das pessoas que participaram do experimento, realizado por 11 dias (16 à 26 de junho de 2017).

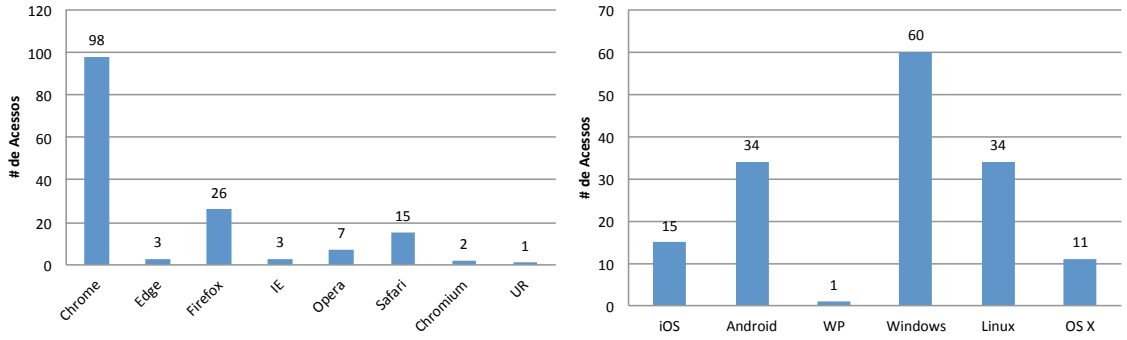


Figura 8. Características dos dispositivos usados no experimento.

Foram registrados 129 participantes, contabilizando 155 acessos. No que diz respeito aos dispositivos, a maioria dos participantes fez uso de *desktops* ou *notebooks* (106 acessos contra apenas 45 de *Smartphones* e *Tablets*). Entretanto, mais a frente essa "definição" espontânea de qual tipo de dispositivo apresentará alguns equívocos. Já sobre o navegador, o Chrome se mostrou o mais usado, seguido pelo Firefox e Safari (Figura 8 à esquerda). Navegadores menos usados como Chromium e UR[5] também foram percebidos. Outro ponto interessante é a presença dos dois navegadores da Microsoft (Internet Explorer e Edge). Por fim, em relação aos sistemas operacionais, o Windows foi a plataforma mais usada, seguido pelo Android e Linux (empatados), iOS e OS X (Figura 8 à direita). Também foi detectado um Windows Phone (WP).

Ao analisar as chaves identificadores geradas, percebeu-se um fato interessante. Houveram apenas 16 *fingerprints*, variando de acordo com o motor de renderização do navegador e com o tipo de dispositivo. Esse comportamento é descrito na Tabela 3.

A primeira observação que se faz na Tabela 3 é que os navegadores que possuem o mesmo motor de renderização – caso do Chrome, Opera e Chromium – compartilham a mesma chave (linhas de 6 à 11). Vale ressaltar que optou-se por realizar esse agrupamento por se tratar do mesmo tipo de dispositivo (*Desktop/Notebook* no caso). Também percebe-se que a chave permanece constante embora esses dispositivos apresentem versões distintas de navegador (Chrome da versão 51.0.2704.106 até 59.0.3071.109 na linha 6), sistema operacional e versão do sistema operacional. Por outro ponto de vista, o uso do *fingerprinting* baseado no *OffilineAudioContext* permite a identificação de navegadores distintos, ou seja, qual está sendo utilizado.

[5]https://www.ur-browser.com

© 2017 Sociedade Brasileira de Computação

Tabela 3. Fingerprints gerados e seus atributos.

	Chave	Navegador	Versão (Navegador)	SO	Versão (SO)	Dispositivo
1	12916ca56 ...	Chrome	58.0.3029.110	OS X	10.12.5	Destkop/Notebook
2		UR	55.1.2883.7			
3	1bd2be502 ...	Chrome	58.0.3029.83	Android	5	Smartphone/Tablet
4	412e24302 ...	Firefox	47.0	Windows	7	Destkop/Notebook
5	559f0ecae ...	Edge	15.15063	Windows	10	Destkop/Notebook
6	64e7af5ca ...	Chrome	58.0.3029.96 - 51.0.2704.79 - 55.0.2883.87 - 59.0.3071.86 - 56.0.2924.87 - 59.0.3071.104 - 58.0.3029.110 - 54.0.2840.90 - 57.0.2987.133 - 59.0.3071.109 - 58.0.3029.81 - 59.0.3071.86 - 51.0.2704.106	Linux	–	Destkop/Notebook
7			58.0.3029.110 - 58.5.3029.81	Windows	10	
8		Opera	46.0.2597.26	Linux	–	
9			45.0.2552.898	Windows	10	
10			45.0.2552.898	Windows	7	
11		Chromium	53.0.2785.143	Linux	–	
12	654fe84fe ...	Chrome	59.0.3071.86	Linux	–	Destkop/Notebook
13			58.0.3029.110 - 59.0.3071.109	Windows	7	
14			58.0.3029.110 - 59.0.3071.109 - 59.0.3071.104	Windows	10	
15			58.0.3029.110	Windows	8	
16		Opera	45.0.2552.898	Windows	10	
17		Chromium	51.0.2683.0	Windows	10	
18	6dc773aa2 ...	Opera	47.0.2628.0	OS X	10.12.4	Destkop/Notebook
19		Chrome	58.0.3029.110	OS X	10.12.4	
20	73e92b923 ...	Safari	10.1.1	OS X	10.12.5	Destkop/Notebook
21			10.1	OS X	10.12.4	Destkop/Notebook
22	749f38fac ...	Chrome	58.0.3029.83 - 51.0.2704.106 - 59.0.3071.92	Android	7.0	Smartphone/Tablet
23			58.0.3029.83 - 43.0.2357.93;	Android	6.0.1	
24			58.0.3029.83 - 43.0.2357.93	Android	5.1.1	
25			58.0.3029.83	Android	4.2.2	
26			58.0.3029.83	Android	7.1.2	
27			56.0.2924.87	Android	5.0	
28		Samsung Broswer	5.4	Android	5.0.2	
29	78a323009 ...	Firefox	53.0	Android	6.0.1	Smartphone/Tablet
30			53.0	Android	5.0.2	
31			54.0	Android	5.1.1	
32	91f4b87d3 ...	Firefox	54.0	Linux/Ubuntu	–	Destkop/Notebook
33			55.0	Linux	–	
34			53.0 e 54.0	Windows	10	
35			54.0	Windows	7	
36			54.0	OS X	10.12	
37			54.0	OS X	10.11	
38	c649ae295 ...	Chrome	58.0.3029.83	Android	6.0.1	Smartphone/Tablet
39			58.0.3029.83	Android	5.1.1	
40			58.0.3029.83	Android	4.4.2	
41		Opera	42.7.2246.114996	Android	6.0.1	
42	d1998f0cc ...	Chrome	59.0.3071.104	Linux	–	Destkop/Notebook
43	e5f4d3534 ...	Safari	10.1.1	OS X	10.12.5	Destkop/Notebook
44	eed33e7c6 ...	Firefox	44.0	Windows	7	Destkop/Notebook
45	038e78e78 ...	Internet Explorer	11	Windows	10	Desktop/Notebook
46			11	Windows	8.1	Smartphone/Tablet
47		Safari	9	iOS	9.3	Smartphone/Tablet
48			10	iOS	10.3.2	
49			11	iOS	11.0	
50		Chrome	59.0.3071.102	iOS	10.3	Smartphone/Tablet
51			59.0.3071.109	Linux	–	Desktop/Notebook

A segunda observação é que quando os navegadores possuem o mesmo motor de renderização e são da mesma versão, mas o tipo dos dispositivos em que estão executando são diferentes (por exemplo, *Smartphone* e *Desktop*), as chaves geradas são distintas. Essa distinção pode ser empregada para determinar o tipo de dispositivo que está sendo usado pelo usuário, mesmo que haja extensões de contramedida que modifiquem, por exemplo, o *User Agent*, já que o processamento do sinal depende do motor de renderização do navegador e do dispositivo onde o mesmo se encontra instalado. Esse resultado pode ser observado no navegador Chrome, versão 51.0.2704.106 nas linhas 6 e 22.

A terceira observação diz respeito a versão do navegador afetar no *fingerprint* gerado. Por exemplo, percebe-se que o Firefox 44 e 47 (linhas 44 e 4, respectivamente) possuem chaves diferentes, embora estejam instalados no mesmo tipo de dispositivo (*Desktop* ou *Notebook*) com o mesmo SO (Windows 7). Esse comportamento pode ser atribuído a

 © 2017 Sociedade Brasileira de Computação

alguma mudança no motor de renderização dos navegadores no decorrer das versões, bem como alguma nuance oriunda do *hardware* do dispositivo.

Outro resultado interessante que foi observado durante a análise dos resultados é que os navegadores Safari, Chrome e Internet Explorer (linhas 45 à 51) estão agrupados, mesmo que seus respectivos motores de renderização sejam distintos. Acontece que para esses navegadores o conjunto de frequências retornado foi nulo devido a alguma falha na rede, por falta de suporte do navegador à *Web Audio* API (algo que ocorre no Internet Explorer, console PS3 e em SmartTV com WebOS) ou por alguma proteção interna do próprio navegador. Esse comportamento pode ser atribuído ao fato da *Web Audio* API ainda não ter um padrão defino e também a limitação do método proposto.

A quarta e última observação é que o método proposto é capaz de identificar a classe dos dispositivos, ainda que informações falsas sobre os mesmos sejam passadas. Por exemplo, um Firefox que se passava por Safari foi agrupado com outros navegadores Firefox. O mesmo aconteceu com um dispositivo *Desktop/Notebook* que se passava por um *Smartphone/Tablet*, mas que foi agrupado com outros dispositivos do tipo *Desktop/-Notebook*. Esses resultados foram confirmados com a verificação do *user agent*, embora essa característica não tenha sido empregada na geração do *fingerprint*.

7. Conclusão

Nesse trabalho foi apresentado um método de *Web Fingerprinting* que utiliza recursos providos pela *Web Audio API* (que até a data de escrita deste trabalho está em *working draft* [W3C 2017]) com a finalidade de identificar a classe dos dispositivos, por meio do emprego do *OfflineAudioContext*. Mostrou-se que com o *OfflineAudioContext* é possível gerar ondas periódicas, obter suas frequências e a partir disso gerar um *fingerprint*. Isso é possível devido a pequenas diferenças entre os navegadores e suas versões, bem como os tipos de dispositivos e os sistemas operacionais dos mesmos. Essas diferenças afetam em como o sinal é gerado, o que pode ser usado para identificar a classe dos dispositivos.

Os resultados apresentados na Seção 6 evidenciam a efetividade do método de *Web Fingerprinting* proposto nesta pesquisa, pois pode-se observar que o mesmo é capaz de identificar a classe dos dispositivos com base no seguinte conjunto de características: $\{$*Navegador; Versão do navegador; Tipo do dispositivo*$\}$. Embora não seja capaz de identificar unicamente os usuários, é possível perceber nuances do navegador e do dispositivo, demonstrando que a *Web Audio API* é uma das características relevantes a ser empregada em métodos de *Web Fingerprinting*.

7.1. Limitações e Trabalhos Futuros

Embora possa ser usado para identificar as classes dos dispositivos, o método de *Web Fingerprinting* proposto neste trabalho atribui chaves diferentes para dispositivos que possuem o mesmo conjunto $\{$*Navegador; Versão do Navegador; Tipo do dispositivo*$\}$, possivelmente devido a alguma característica que os distinguem, mas que esse estudo não foi capaz de detectar. Além disso, o método falha nos dispositivos e navegadores que não suportam a *Web Audio* API.

Como trabalhos futuros, pode-se destacar: (I) Investigar o por quê de dispositivos que na teoria deveriam fazer parte da mesma classe estão em classes distintas; (II) Investigar como o *AudioContext* ou *OfflineAudioContext* pode ser usado para identificar,

© 2017 Sociedade Brasileira de Computação

de forma única, os usuários, invés de apenas determinar a classe dos dispositivos; (III) Investigar o comportamento instável do *AudioContext* em métodos de *Web fingerprinting*.

Agradecimentos

Os autores deste trabalho agradecem à CAPES por viabilizar esta pesquisa.

Referências

Bursztein, E., Malyshev, A., Pietraszek, T., and Thomas, K. (2016). Picasso: Lightweight Device Class Fingerprinting for Web Clients. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 93 –102, Austria.

Eckersley, P. (2010). How Unique is Your Web Browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, pages 1–18, Berlin, Germany. Springer-Verlag.

Englehardt, S. and Narayanan, A. (2016). Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1388–1401. ACM.

Khademi, A. F., Zulkernine, M., and Weldemariam, K. (2015). An Empirical Evaluation of Web-Based Fingerprinting. *IEEE Software*, 32(4):46–52.

Laperdrix, P., Rudametkin, W., and Baudry, B. (2016). Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In *IEEE Symposium on Security and Privacy (SP)*, pages 878–894. IEEE.

MDN (2016). Web audio: conceitos e uso. `https://developer.mozilla.org/pt-BR/docs/Web/API/`.

Mowery, K., Bogenreif, D., Yilek, S., and Shacham, H. (2011). Fingerprinting Information in JavaScript Implementations. In *Proceedings of W2SP 2011*, pages 1–11. IEEE.

Mowery, K. and Shacham, H. (2012). Pixel perfect: Fingerprinting canvas in HTML5. In *Proceedings of W2SP*, pages 1–12. IEEE.

Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E., and Wien, F. (2013). Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*.

Nakibly, G., Shelef, G., and Yudilevich, S. (2015). Hardware Fingerprinting Using HTML5. *CoRR*, abs/1503.0.

Saraiva, A., Feitosa, E., Elleres, P., and Carneiro, G. (2014). Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. In *Livro de Minicursos do XIV SBSeg*, pages 49–98, Belo Horizonte - MG. SBC.

W3C (2017). Web Audio API. `https://webaudio.github.io/web-audio-api/`.

Ximenes, P., Correia, M., Mello, P., Carvalho, F., Franklin, M., and Andrade, R. (2016). TARP Fingerprinting: Um Mecanismo de Browser Fingerprinting Baseado em HTML5 Resistente a Contramedidas. In *Anais do XVI SBSeg*, pages 100–113, Niterói - RJ, Brazil.

 © 2017 Sociedade Brasileira de Computação

ConCReCT: Um Mecanismo para Mitigação dos Efeitos de Ataques de Negação de Serviço em Internet das Coisas Industriais

Vladimir Borgiani[1], Patrick Moratori[1], Juliano Kazienko[2], Emilio Tubino[3]

[1]ICT - Instituto de Ciência de Tecnologia
Universidade Federal Fluminense (UFF)
Rio das Ostras, RJ, Brasil

[2]CTISM - Colégio Técnico Industrial de Santa Maria
Universidade Federal de Santa Maria (UFSM)
Santa Maria, RS, Brasil

[3]GCC - Graduação em Ciência da Computação
Universidade Federal do Pampa (UNIPAMPA)
Alegrete, RS, Brasil

{vborgiani, pmoratori}@id.uff.br, kazienko@redes.ufsm.br

emiliotubino@alunos.unipampa.edu.br

Abstract. *Volumetric Denial of Service (DoS) attacks are traditionally defined by a high traffic volume targeted to one or more Internet connected devices. As countermeasure, strategies based on admission control and reputation are posed as an efficient way for detecting attacks. However, the mitigation of such attacks is a challenge that demands a suitable solution. In this work, we propose the ConCReCT (congestion control by duty-cycle restriction) mechanism in order to mitigate DoS attacks in an industrial internet of things environment. The proposed mechanism enables each node to manage the traffic and control the duty-cycle parameters to pinpoint attacker nodes and mitigate DoS attacks. Simulation results indicate 100% of efficacy for networks up to 20 nodes.*

Resumo. *Ataques de Negação de Serviço Volumétrico são caracterizados por um alto volume de tráfego direcionado a um, ou mais, dispositivos conectados à Internet. Estratégias existentes, baseadas em controle de admissão e reputação, são eficazes na detecção. Porém, a minimização, ou eliminação, de seus efeitos carecem de esforços mitigatórios. Este artigo propõe um mecanismo denominado ConCReCT (Controle de Congestionamento com Restrição do Ciclo de Trabalho) cujo objetivo é permitir que cada nó de uma rede possa gerenciar o seu tráfego e controlar parâmetros do Ciclo de Trabalho (Duty-cycle) a fim de detectar e, principalmente, mitigar ataques de negação de serviço. Resultados de simulação indicam uma eficácia de 100% em redes com até 20 nós.*

1. Introdução

Redes de comunicação de dados industriais, com ou sem fio, tradicionalmente, demoram a adotar soluções inovadoras de comunicação devido à maturidade e funcionalidades

 © 2017 Sociedade Brasileira de Computação

providas por tecnologias como Aquisição de Dados e Controle Supervisório *(SCADA - Supervisory Control and Data Acquisition)*, baseada em protocolos legados como ModBus e DNP [Alcaraz and Lopez 2010]. Este cenário tem sofrido o impacto da necessidade de disponibilização dos dados para os mais variados dispositivos, incluindo dispositivos como telefones celulares e outros mais restritivos em termos de recursos computacionais assim como diversos sistemas operacionais.

A evolução das redes industriais, com a utilização do protocolo IP e abandono de padrões proprietários, permitiu o que [Evans and Annunziata 2012] chamou de revolução da Internet Industrial, ou ainda, de Indústria 4.0, quando o foco da comunicação são interações entre dispositivos e não mais entre seres humanos e dispositivos. A evolução definitiva para uma solução baseada em IPv6 é o futuro da Internet das Coisas (*Internet of Things - IoT*) e as redes de sensores sem fio são partes importantes como descrito em [Gubbi et al. 2013] e [Miorandi et al. 2012]. Essa evolução implica em maior exposição dos dispositivos às ameaças na Internet e, por isso, existe a necessidade do desenvolvimento de mecanismos de prevenção e eliminação de ameaças.

Ataques de negação de serviço com inundação de pacotes, ou volumétricos, são particularmente críticos pois dificultam que mensagens importantes alcancem o seu destino devido ao congestionamento criado na rede [Ghazali and Hassan 2011]. Já [Rughinis and Gheorghe 2010] destaca o efeito colateral do gasto energético gerado pelo constante recebimento de pacotes na interface de rede do nó, que pode provocar o colapso da rede e isolar a mesma em pequenas seções sem comunicação entre si. Em ambos os casos, o impacto em redes industriais é crítico e as técnicas apresentadas sugerem tratar o tráfego malicioso como interferência a ser ignorada. Apenas ignorar os pacotes não atende as necessidades *IoT* pois não elimina o alto volume de tráfego que causa congestionamento e diminue a taxa efetiva de entrega de pacotes da rede.

Em [Karlof 2003], são detalhados diversos ataques contra a infraestrutura de rede, incluindo o denominado *Hello flooding*. Uma contramedida simples contra esse ataque é o teste de bidirecionalidade de dados descritos em [Wallgren et al. 2013] que trata de um sistema de identidade, com criptografia, mas que nem sempre está disponível em dispositivos com recursos computacionais escassos. Em [Zargar et al. 2013], são descritos diversos ataques de negação de serviço, tanto no âmbito de aplicação quanto no nível de rede e transporte, inclusive ataques de reflexão onde o atacante não precisa nem mesmo comprometer um determinado nó, mas apenas explorar uma vulnerabilidade enviando pacotes capazes de iniciar um comportamento anormal que leve o nó alvo a gerar tráfego direcionados a outros nós ou mesmo para a Internet. Atualmente, a utilização de *firewalls* de borda apresentam dificuldade em mitigar este tipo de ataque por se tratar de um tráfego legítimo e, portanto, são necessários esforços mitigatórios no próprio dispostivo e não somente na infra-estrutura de rede [Sassani et al. 2016].

A proposta deste trabalho consiste em um mecanismo denominado ConCReCT (Controle de Congestionamento com Restrição do Ciclo de Trabalho). O objetivo de tal mecanismo consiste em estender o mecanismo *SVELTE* [Raza et al. 2013] de modo que o nó sorvedouro detecte ataques que produzam tráfego excessivo de pacotes IPv6 transmitido para a rede por todos os outros nós e tome medidas para a mitigação do ataque. Para avaliar o mecanismo, foi utilizado o sistema operacional *ContikiOS*, criado especificamente para *IoT*, e seu simulador embutido, denominado *Cooja* [Osterlind et al. 2006].

 © 2017 Sociedade Brasileira de Computação

O ConCReCT estende o mecanismo denominado *SVELTE* que, originalmente, apresenta um Sistema de Detecção de Intrusão para detecção de ataques *Sinkhole*, a fim de mitigar os efeitos de ataque de negação de serviço em ambiente industrial.

As seções subsequentes deste artigo estão organizadas da seguinte maneira: a Seção 2 discute os trabalhos relacionados; na Seção 3, o mecanismo ConCReCT é introduzido; a Seção 4 demonstra a avaliação de tal mecanismo; a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Ataques de negação de serviço baseados em inundação de pacotes são especialmente nocivos e ignorá-los não resolve o problema. Esse fluxo de pacotes, constante ou variável, afeta a capacidade da rede de entrega dos pacotes legítimos devido ao aumento da concorrência pelo acesso ao canal de radiofrequência. Diversos mecanismos de controle de congestionamento são apresentados em [Ploumis et al. 2012], mas todos baseiam-se na premissa de que o tráfego é legítimo e não um ataque de negação de serviço e, portanto, não deve ser eliminado.

Mecanismos de detecção de intrusão como os descritos em [Stavrou and Pitsillides 2014] utilizam *Blacklists* ou *Whitelists* como mecanismo para ignorar ou aceitar mensagens de determinados dispositivos vizinhos. Entretanto, isto não elimina o tráfego nocivo da rede enviado por um nó autenticado ou considerado confiável. Estes utilizam sistemas de reputação enquanto outros utilizam sistemas de identidade para verificação dos nós, porém nenhum deles permite a eliminação, mesmo que temporária, do tráfego nocivo.

Alguns mecanismos atuam na priorização de tráfego de modo a permitir que determinados pacotes sejam priorizados através da utilização de filas de prioridade [Alam et al. 2011]. Entretanto, este mecanismos possuem a desvantagem de que marcadores de qualidade de serviço podem ser manipulados pelo atacante de modo que o tráfego nocivo passe a ser interpretado como tráfego importante e legítimo.

Em [Kasinathan et al. 2013], é descrito um método distribuído onde todos os dispositivos verificam o tráfego em comparação com uma base de dados de assinaturas. Contudo, esse tipo de mecanismo requer memória para armazenamento de uma base de assinaturas, com atualizações constantes, além de um alto custo energético para monitoração de mensagens durante todo o tempo.

O mecanismo descrito em [Dharini et al. 2015] associa o gasto energético de um nó com um possível ataque de inundação de pacotes. Entretanto, um nó com muitos filhos poderia gerar uma alta taxa de falsos positivos pois o nó ficaria ativo muito mais tempo do que um nó sem filhos. Além disso, simulações no *Cooja* já demonstraram que o gasto energético durante o modo de escuta é tão alto quanto em modo de transmissão [Dunkels 2011].

Em [Kwon et al. 2013], a proposta é utilizar caminhos alternativos para escapar do congestionamento, no entanto não descreve como recuperar a integridade dos caminhos afetados. Ataques múltiplos poderiam até inviabilizar a existência de um caminho livre de congestionamento.

Em [Sachan et al. 2013], utiliza-se uma função *Hash* com o objetivo de proteger

© 2017 Sociedade Brasileira de Computação

as comunicações criando uma assinatura digital. A autenticação também não elimina a possibilidade de que um nó autenticado e confiável apresente um comportamento malicioso e passe a enviar tráfego acima do esperado.

A proposta de [Nigam et al. 2014] consiste em medir a quantidade de pacotes transmitidos por um determinado nó e o tempo de chegada entre eles no sorvedouro. Apesar desta medição ser importante na detecção de ataques conjugados, como *Sinkhole* e *Blackhole*, onde determinado tráfego é descartado silenciosamente, sua utilização para detecção de inundação de pacotes é questionável já que o intervalo de chegada entre os pacotes tende a ser muito pequeno, dificultando a detecção de um possível ataque.

Entretanto, a idéia de contabilizar os pacotes descrita em [Nigam et al. 2014] é aproveitada no ConCReCT para detecção de ataques de negação de serviço. O SVELTE permite ao sorvedouro montar uma matriz com dados sobre cada nó da rede e monitorar o estado de cada nó através de mensagens específicas. A não recepção dessas mensagens indica que o nó está sendo filtrado por um atacante que se encontra no meio do caminho dos pacotes. Entretanto, o mecanismo não funciona para ataques cujo objetivo é congestionar os canais de comunicação. Existe, ainda, o fato de que o atacante poderia encaminhar mensagens específicas, como de protocolos de roteamento, permitindo que a rede aparente ser funcional.

A literatura atual, até onde pesquisamos, carece de estudos ligados, diretamente, à mitigação de ataques volumétricos de negação de serviço através de inundação de pacotes em redes *IoT*. Criptografia, Reputação e Qualidade de Serviço não removem o tráfego de pacotes maliciosos da rede e a consequente deterioração do meio de transmissão. É preciso que o tráfego malicioso seja contido o mais próximo de sua origem possível.

3. O Mecanismo ConCReCT

Nesta seção, é apresentado o mecanismo ConCReCT (Controle de Congestionamento com Restrição do Ciclo de Trabalho). Seu objetivo consiste em atuar na detecção de ataques que elevam a quantidade de tráfego na rede devido a ataques de inundação de pacotes, permitindo a mitigação dos efeitos de ataques de negação de serviço volumétrico. Ataques de inundação de pacotes, como os baseados em reflexão, onde o atacante envia pacotes com endereços de origem forjados [Zargar et al. 2013] com o intuito de fazer com que um nó passe a gerar tráfego excessivo dentro de sua própria rede. É importante salientar que esse tipo de ataque não necessita que um nó seja comprometido.

O mecanismo ConCReCT compõe-se de três módulos. O primeiro módulo é responsável pela construção de uma matriz baseada na topologia contruída pelo *RPL (Routing Protocol for Lossy Networks)* e armazenamento de informações como: o número de filhos ou sensores e o tráfego gerado por cada dispositivo. O segundo módulo responsável pelo cálculo do tráfego de cada dispositivo dentro de um específico intervalo de tempo de modo a identificar o comportamento malicioso. Assim que detectado o comportamento, o terceiro módulo permite silenciar o nó através de mensagens enviadas pelo nó sorvedouro e interpretadas pelo código *ContikiMAC* responsável pelo controle do ciclo de trabalho de cada nó.

Dispositivos sem fio para *IoT* precisam de um controle eficiente do sistema de rádio e, mesmo sem transmitir mensagens, utilizam tanta energia quanto ao transmiti-las.

 © 2017 Sociedade Brasileira de Computação

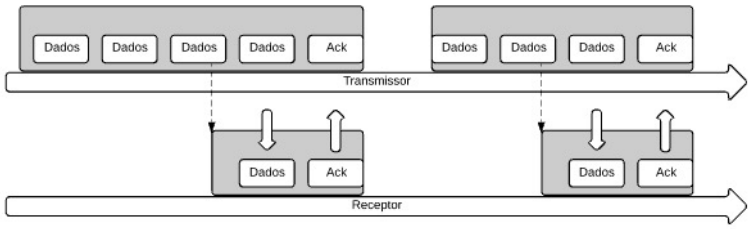


Figura 1. Pacotes de Dados e envio da Confirmação *(ACK)*.

Como o dispositivo precisa estar ativo para detectar transmissões de vizinhos, o controle do ciclo de trabalho é fundamental já que pode provocar um aumento desnecessário no número de mensagens enviadas [Dunkels 2011].

O ConCReCT se utiliza de um mecanismo, existente em diversos sistemas embarcados para *IoT*, de controle de ciclo de trabalho (*Duty-Cycle*), que regula o tempo de atividade do sistema de radiofrequência através de intervalos de funcionamento baseados na detecção da Indicação de Potência do Sinal Recebido *(RSSI - Receive Signal Strenght Indication)* [Dunkels 2011].

A Figura 1 ilustra a janela de ativação do sistema de rádio e a detecção de pacotes de dados sendo recebidos. O nó transmissor envia o mesmo pacote, diversas vezes, até receber uma mensagem de confirmação da camada de enlace *ACK (Acknowledgment)* do nó vizinho. O receptor, após detectar uma transmissão perdida, continua ativo para receber o próximo pacote repetido (identificado pelos retângulos brancos dentro do retângulo cinza) e somente então pode enviar uma confirmação (*ack*) de volta. Dessa forma, podem ser necessários vários quadros da camada de enlace para a entrega de um pacote IPv6 entre dois nós ter sucesso.

Mecanismos de controle do ciclo de trabalho foram concebidos para melhorar a eficiência energética de nós sensores sem fio. Entretanto, o ConCReCT objetiva utilizar o ciclo de trabalho como contramedida para silenciar nós maliciosos, impedindo-os de gerar uma quantidade de pacotes incompatível com suas funções primárias.

O ConCReCT permite uma ação precisa, de forma que um nó considerado malicioso seja silenciado, ou seja, não gere mais nenhum pacote mas continue a encaminhar os pacotes proveniente de seus filhos. Isso permite que o verdadeiro ofensor seja penalizado e que seus filhos não precisem reorganizar a topologia de forma a encontrar um novo pai, aumentando a tolerância a falhas da rede.

O nó sorvedouro monta uma matriz contendo informações sobre os nós da rede como suas taxas de transmissão, o número de filhos ou sensores de cada nó e um indicador de penalidade. Um nó com 10 filhos seria o equivalente a um nó com 10 sensores em termos de tráfego, considerando uma rede homogênea onde os sensores enviam a mesma quantidade de mensagens por minuto. As informações são colhidas através das mensagens provenientes de cada nó. Essa matriz é atualizada em intervalos regulares com base em informações enviadas por cada nó da rede e, em períodos específicos, seus valores de tráfego comparados ao Tráfego Esperado em um nó N ($\mathbf{TE_n}$) para aquele nó específico.

Para determinar um valor médio de $\mathbf{TE_n}$, foram realizadas simulações no módulo *Cooja* onde o nó número nove possui diferentes quantidades de filhos, como demonstra

© 2017 Sociedade Brasileira de Computação

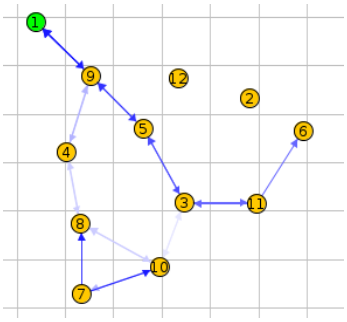


Figura 2. Nó 9 com 10 filhos.

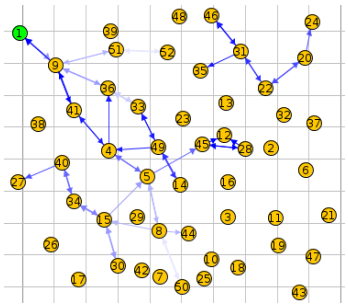


Figura 3. Nó 9 com 50 filhos.

a Tabela 1. Foram realizadas 3 simulações , cada uma com 1 hora de duração, com um nó com 10, 20, 30, 40 e 50 filhos. A tabela sumariza o resultado médio das simulações utilizando o código *RPL-UDP* nativo do *ContikiOS* que envia mensagens de *Hello* direcionadas ao sorvedouro a cada 60 segundos.

O Tráfego Medido no nó N ($\mathbf{TM_n}$) é calculado, no nó sorvedouro 1, através da contabilização das mensagens de dados recebidas no mesmo.

As Figuras 2 e 3 demonstram os extremos dos cinco cenários de simulações (10 e 50 filhos) onde o nó número 9 é o elo de ligação entre seus filhos e o nó sorvedouro 1.

As mensagens de *Hello* são mensagens com texto simples e, no código utilizado pelo ConCReCT, simulam as mensagens que deveriam ser enviadas por sensores para comunicar eventos como alterações em temperatura, pressão, presença de gás sulfídrico ou sensores de presença para áreas de risco. Essas mensagens são enviadas em intervalos regulares e conhecidos, permitindo a previsibilidade do tráfego na rede. Eventuais tráfegos em rajada podem existir e, por esse motivo, a detecção exige que o comportamento malicioso aconteça três vezes consecutivas para ser considerado um ataque.

O Algoritmo 1 ilustra o funcionamento do código *RPL-UDP* servidor, executado no nó sorvedouro e existente no simulador *Cooja*, modificado para permitir a identificação e mitigação do ataque volumétrico. Estas modificações introduzem a construção da matriz e contabilização de tráfego e número de filhos de cada nó da rede. A função *Controla* é responsável por calcular o tráfego esperado em função do número de filhos do nó N. Caso o tráfego medido seja maior do que o tráfego esperado, um acumulador é incrementado. Caso esse contador assuma o valor maior que o limite definido, o indicador de penalidade

Tabela 1. Simulação de mensagens enviadas.

Quantidade de Filhos	# 1	# 2	# 3	Média de mensagens enviadas	Desvio padrão	Confiança 95%
10	29	28	27	28	1	1,132
20	39	40	42	41	1,528	1,729
30	49	49	53	51	2,309	2,613
40	68	70	71	70	1,527	1,729
50	83	80	79	81	2,082	2,356

© 2017 Sociedade Brasileira de Computação

Algoritmo 1 Servidor

```
1:  função CONTROLA(No)
2:      TE_n = Formula(Numero de filhos)
3:      se TM_n > TE_n então
4:          aviso=aviso+1
5:          se aviso > LIM então
6:              No.penalidade = 1
7:          fim se
8:      fim se
9:  fim função
10: função MITIGA
11:     Controle = Controla(No.filhos)
12:     se Controle == 1 então
13:         Responde("wait")
14:     senão
15:         Responde("OK")
16:     fim se
17: fim função
18: função CICLO
19:     Ciclo = (periodo * relogio)
20:     enquanto 1 faça
21:         se Fim(Ciclo) então
22:             ZeraAvisos(No)
23:         fim se
24:     fim enquanto
25: fim função
```

é ativado. A função *Mitiga* recebe o retorno de *Controla* e envia **wait**, para silenciar o nó em caso de ataque detectado ou **ok** caso o limite não tenha sido atingido. A função *Ciclo* é responsável por zerar o acumulador e o contador de penalidade.

O Algoritmo 2 ilustra o funcionamento do código *RPL-UDP* cliente, executado nos nós regulares de rede, modificado de forma a permitir o escalonamento da geração de *Hellos* de uma mensagem por minuto para uma mensagem por segundo, configurando assim um ataque de inundação de pacotes a ser detectado pelo algoritmo servidor. A função *Controlado* verifica se a mensagem **wait** foi recebida e aumenta o período de silêncio para a variável *periodom* para 300 segundos.. A função *Envia pacote* permite que os dados dos filhos continuem a ser enviados apesar do nó continuar sem poder transmitir seus próprios pacotes. A função *Ciclo* é responsável por reativar o envio de pacotes do nó após o fim do tempo de penalidade.

Para cada uma das cinco opções de quantidade de filhos do nó nove, foram realizadas três simulações, sendo extraída a média aritmética entre elas. A Figura 4 ilustra as médias, seu desvio padrão com confiança de 95% e, também, a curva correspondente à equação que calcula o $\mathbf{TE_n}$ obtida através da utilização da técnica de ajustes de curva aos dados com o módulo *Curve Fitting* do software *Matlab* [The MathWorks Inc. 2015], gerando a seguinte Equação 1:

© 2017 Sociedade Brasileira de Computação

Algoritmo 2 Cliente

```
1:  função CONTROLADO
2:      se mensagem == ”wait” então
3:          periodo = periodom
4:      fim se
5:  fim função
6:  função ENVIA PACOTE
7:      Envia (BUF, Nfilhos, ”MSG”)
8:  fim função
9:  função CICLO
10:     enquanto 1 faça
11:         Controlado()
12:         se Fim(Ciclo) então
13:             Envia pacote()
14:             Ciclo = (periodo * relogio)
15:         fim se
16:     fim enquanto
17: fim função
```

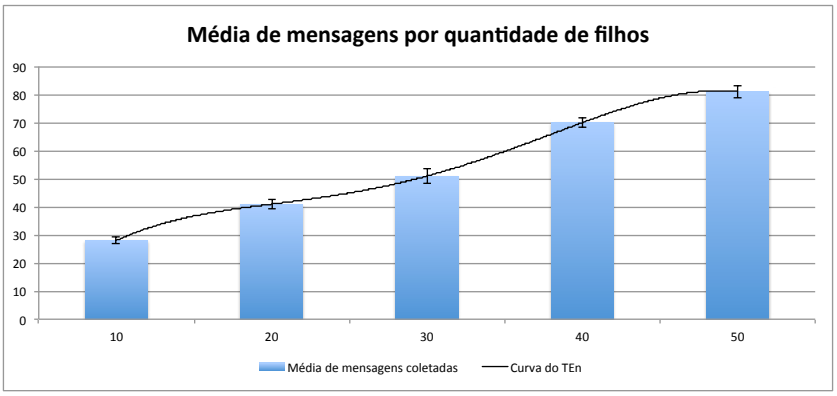


Figura 4. Média de mensagens, desvio padrão e confiança de 95%.

$$TE_n = 0,295 + 2,94 * (nfN) - 0,056 * (nfN)^2 + 0,0006 * (nfN)^3 \quad (1)$$

A Equação 1, portanto, permite prever o tráfego de um sensor de acordo com **nfN**, que representa o número de filhos ou sensores do nó N.

A detecção do comportamento malicioso é realizada através da comparação entre o $\mathbf{TM_n}$, medido em intervalos de sessenta segundos, e $\mathbf{TE_n}$, como demonstrado na Inequação 2. O nó somente é considerado malicioso se a comparação retornar verdadeiro em três intervalos consecutivos, ou seja, quando o contador de penalidade assumir valor maior do que 3. O objetivo é prevenir que uma rajada de mensagens não seja confundida com um ataque real.

$$TM_n > TE_n \quad (2)$$

© 2017 Sociedade Brasileira de Computação

Caso o nó N seja considerado malicioso, o nó sorvedouro aplica uma penalidade ao nó colocando-o em modo dormente durante um intervalo de trezentos segundos através do envio de uma mensagem de espera **(wait)**. Este intervalo é suficiente para que o protocolo *RPL* mude a topologia de rede, se necessário. Durante este intervalo, o nó malicioso continua encaminhando o tráfego dos nós filhos, conservando, assim, a integridade da rede. Ao fim do intervalo, o contador de penalidade é reajustado para zero e o processo de monitoração recomeça.

4. Experimentos e Análise dos Resultados

Nesta seção o ConCReCT é avaliado através de simulação conforme as seguintes métricas: taxa de detecção e mitigação do nó malicioso, além da variação (delta) do tempo entre detecção e mitigação. Em uma rede industrial, como uma plataforma de petróleo, o número de sensores é conhecido e controlado, assim como a taxa de mensagens que os sensores enviam, resultando em uma rede determinística. Eventuais tráfegos em rajada, como alterações súbitas de temperatura que podem gerar mais mensagens que o esperado, são tratadas através da detecção que exige que a mesma ocorra em três intervalos consecutivos.

O *ContikiOS* incorpora funcionalidades IPv6, RPL, 6LowPAN e 802.15.4, que o posicionam como um ecossistema de *IoT* e não somente uma rede de sensores sem fio processando uma pilha de protocolos específica [Raza et al. 2013].

O modelo de ataque baseia-se em um atacante que pretende causar um comportamento malicioso apenas enviando pacotes legítimos ao nó de destino, como em um ataque de reflexão utilizando o protocolo de hora da rede (NTP) [Czyz et al. 2014]. O objetivo é fazer com que o nó passe a gerar uma quantidade de tráfego maior do que o normal na tentativa de responder à solicitação do atacante. O fato de ser uma rede industrial permite saber exatamente quanto tráfego, informação e periodicidade, cada nó deveria enviar em intervalos específicos e uma alteração significativa no tráfego revela um possível ataque.

O ambiente simulado baseia-se na arquitetura de sensor *tmote sky* [Moteiv Corporation 2006], de propriedade da Mote IV Corporation, de potência ultra-baixa, compatível com o padrão 802.15.4, baseado no difundido microcontrolador MSP430 e com recursos de memória RAM (10k) e armazenamento flash (48k). O ambiente de simulação *Cooja* permitiu simular até 20 sensores na criação e manutenção da matriz de controle devido ao tamanho da matriz no nó sorvedouro, que excede o tamanho de memória disponível em tal arquitetura. Entretanto, esse número, combinado com a divisão das redes por função (temperatura, pressão e vazão de gás), onde existiriam diversos sorvedouros, é suficiente para cobrir um ambiente típico de uma plataforma de petróleo, permitindo a extensão da rede sem aumentar a competição pelo canal de rádio. Ambientes industriais se beneficiam da homogeneidade dos dispositivos de rede por facilitar a manutenção e criar um ambiente preditivo.

A utilização de uma taxa de sucesso na entrega de pacotes de 100% permite que, mesmo com atrasos, todas as mensagens alcancem o nó sorvedouro e, portanto, aumentando a eficiência do método em detectar e silenciar o nó ofensor. Uma alta taxa de entrega de pacotes é possível graças a limitação de nós em redes operando em diferentes frequências, resultando em menor disputa pelo meio físico.

Os cenários avaliados correspondem a uma área de dez mil metros quadrados (100

© 2017 Sociedade Brasileira de Computação

Tabela 2. Parâmetros de Simulação no Cooja.

Parâmetro	Valor
Área	100 x 100 m
Nós	10, 15 e 20
Topologia	Grade
Atacantes simultâneos	1, 3 e 5
Alcance de transmissão	45 m
Alcance de interferência	65 m
Atraso de inicialização de cada nó	1 milissegundo
Tempo de simulação	Até ocorrer a detecção
Frequência de *Hellos*	1/minuto
Frequência de *Hellos* durante ataque	1/segundo

x 100 metros) de forma a simular o tamanho médio de uma plataforma de exploração de Óleo e Gás, representando um ambiente industrial com conectividade da Internet das Coisas. A Tabela 2 apresenta todos os parâmetros utilizados nas simulações.

Os nós foram dispostos de maneira linear, formando uma grade. A disposição em grade foi utilizada por permitir uma cobertura de 100% da área com o menor número possível de nós.

A Mobilidade do sensor também é importante, permitindo sua realocação no espaço da plataforma e sendo mais uma razão para utilização de sensores sem fio em ambientes *IoT*. Para evitar congestionamento e aumentar a largura de banda da rede, uma implementação real de Internet das Coisas nesse ambiente poderia possuir mais de um sorvedouro e utilizar diferentes canais de radiofrequência, formando camadas de rede sobrepostas. Esta sobreposição também permitiria que mais de um sensor seja responsável pela mesma informação, aumentando as chances de entrega da informação.

O nó 1 é o sorvedouro e utiliza o código udp-server.c modificado, sendo o destino de todos os pacotes de *Hello* enviados pelo nós clientes, que utilizam o código udp-client.c modificado. Todos os nós clientes possuem uma função para iniciar o ataque durante a simulação. A taxa de 1 pacote por segundo para definir o ataque volumétrico é suficiente para diferenciar nós maliciosos de outros nós que propagam mensagens de *Hello* e do protocolo *RPL*. Para melhor compreensão e reprodutibilidade dos resultados, tais códigos estão disponíveis em https://github.com/vspnet/ConCReCT. Os códigos originais do *ContikiOS* podem ser encontrados em https://github.com/contiki-os/contiki/tree/master/examples/ipv6/rpl-udp.

Foram realizadas 20 simulações para cada cenário, com 10, 15 e 20 nós, todas com duração de 2 horas permitindo iniciar, detectar e mitigar os ataques ativados. Foram iniciados ataques com 1, 3 e 5 nós simultâneos em cada cenário, totalizando 9 cenários simulados como demonstra a Tabela 3.

Foram colhidos dados relativos aos tempos, em milissegundos, do início do ataque, da detecção do ataque pelo nó sorvedouro e da mitigação do ataque com o silenciamento efetivo do nó através de mensagem **wait** enviada pelo sorvedouro em direção ao nó malicioso.

 © 2017 Sociedade Brasileira de Computação

Tabela 3. Cenários de simulação.

Cenário	Nós da rede	Nós maliciosos
1	10	1
2	10	3
3	10	5
4	15	1
5	15	3
6	15	5
7	20	1
8	20	3
9	20	5

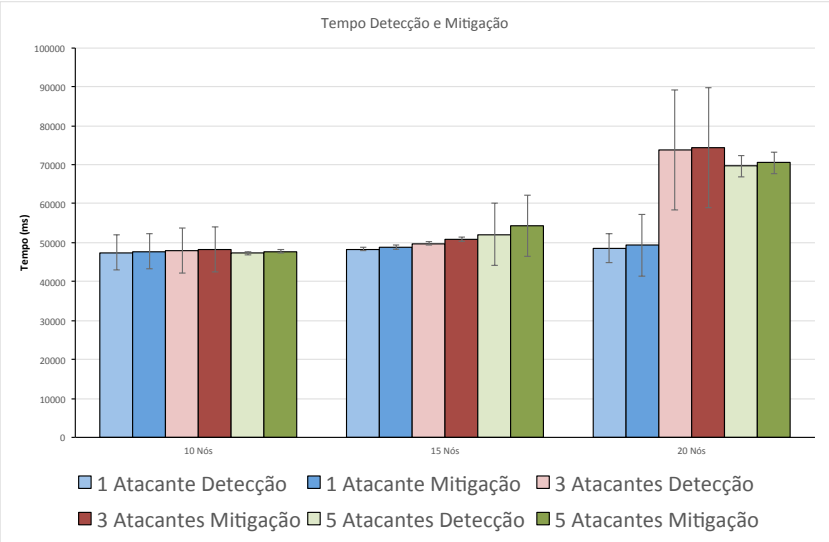


Figura 5. Tempos de Detecção e Mitigação.

A Figura 5 apresenta os tempos, em milissegundos e com confiança de 95%, da detecção e da mitigação do ataque. O tempo de detecção é contabilizado desde o início do ataque até que o sorvedouro detecte o comportamento malicioso. O tempo de mitigação é contabilizado desde a detecção do ataque até que o nó malicioso receba o pacote **wait** e pare de gerar tráfego.

Observa-se que, em cenários com maior densidade de nós e atacantes, os tempos apresentam aumento significativo. Esse aumento é provocado pela disputa pelo acesso ao meio de transmissão e também pode ser influenciado pelo tempo de ciclo de trabalho de cada nó, causando congestionamento e, consequentemente, perda de pacotes.

A Figura 6 apresenta as variações, chamadas neste trabalho de *Delta*, do tempo entre a detecção e a efetiva mitigação do ataque, também com confiança de 95%. É possível verificar um aumento significativo do *Delta* entre detecção e mitigação no cenário com 15 nós (3 e 5 atacantes). Acredita-se que esse aumento pode ser explicado pelo início do ataque coincidir com o momento em que o protocolo de roteamento *RPL* iniciou a troca de informações entre todos os nós, ocasionando um congestionamento momentâneo. Caso a mensagem **wait** seja perdida, o sorvedouro continua a computação da quantidade de

© 2017 Sociedade Brasileira de Computação

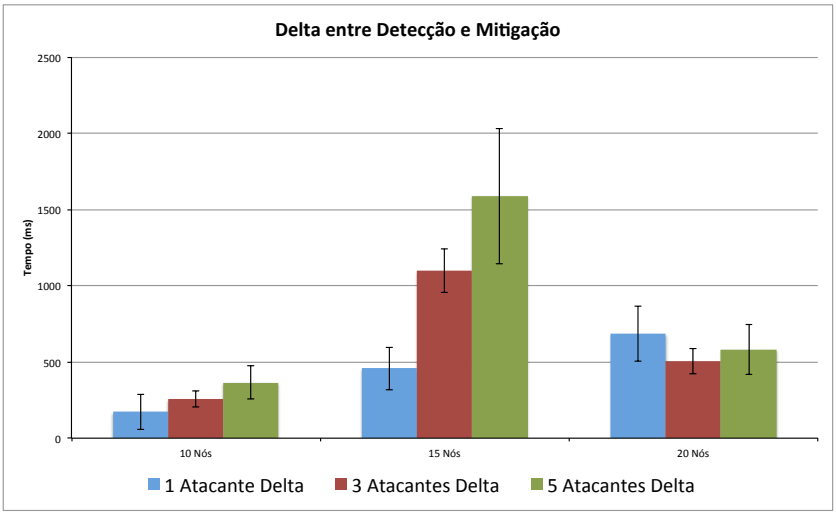


Figura 6. Variação do tempo entre Detecção e Mitigação.

mensagens e, quando novamente detectado, outra mensagem **wait** é enviada. A perda da mensagem **wait** pode atrasar a detecção e mitigação, mas o mecanismo continuará enviando a mensagem até o nó ser silenciado, demonstrando a robustez do processo mesmo diante de uma rede com menor taxa de sucesso de entrega de pacotes.

Os resultados demonstram que o ConCReCT consegue uma taxa de detecção e mitigação de 100% em ambientes limitados a 20 nós e com até 5 ataques simultâneos. Entretanto, essa taxa pode diminuir em função do atraso, ou mesmo da perda, das mensagens enviadas por cada nó ao servidor ou da mensagem **wait** enviada pelo sorvedouro ao nó malicioso. O atraso faz com que o servidor credite, parcialmente, tráfego medido em um intervalo *X* ao próximo intervalo *X+1*, ocasionando um atraso na detecção. A perda de mensagens, caso ocorra, pode causar danos maiores porque aumenta a chance de que as mensagens não cheguem ao servidor e, portanto, não sejam utilizadas na detecção do ataque. Mensagens perdidas podem, inclusive, interferir no cálculo de $\mathbf{TE_n}$ já que o servidor pode não saber o número correto de filhos de determinado nó e, portanto, não poderia estimar corretamente o tráfego esperado no nó.

É possível perceber o aumento no tempo de detecção e, consequentemente, no tempo de mitigação causado pelo atraso na entrega dos pacotes ao nó sorvedouro, principalmente quando mensagens do protocolo *RPL* competem pelo acesso ao meio físico para transmissão. Esse fato é comprovado, em algumas das simulações, pois os únicos tráfegos presentes em todas as simulações são mensagens *RPL* e *Hello* e é possível identificar o momento em que ambos ocorrem simultaneamente.

Os atrasos medidos na detecção e mitigação não comprometem a eficácia do mecanismo, dentro dos cenários testados, pois ambos ficam abaixo de 2 segundos.

5. Conclusões e Trabalhos Futuros

Ataques de negação de serviço por inundação de pacotes têm sido a principal causa de indisponibilidade de serviços na Internet e grande parte dos ataques direcionados a dispositivos pertencentes a categoria Internet das Coisas. Essas plataformas de ataque são comprometidas por técnicas de reflexão de tráfego, onde o atacante nem mesmo precisa

 © 2017 Sociedade Brasileira de Computação

ter acesso ao nó e comprometer o seu código. O ConCReCT se mostra eficaz em uma rede industrial controlada, onde dispositivos heterogêneos apresentam um comportamento homogêneo e preditivo, apresentando uma taxa de 100% de detecção e mitigação em todos os cenários simulados. É de suma importância que cada nó possa ter sua quantidade de tráfego controlada de maneira a evitar gargalos nas comunicações de eventos importantes. O fato do nó malicioso poder ser silenciado e ainda ser capaz de encaminhar o tráfego de nós não maliciosos demonstra tolerância à falhas, na medida em que os nós filhos não precisam convergir para outro nó pai.

Como trabalhos futuros, podemos destacar a necessidade de estender o ConCReCT para recuperar o nó após a detecção, e mitigação temporária, do ataque. Uma possibilidade poderia ser a geração de uma nova identidade para o nó, impedindo que o atacante continue a enviar o tráfego que provoca o comportamento malicioso. Outro trabalho consiste em permitir o armazenamento do histórico do comportamento da rede, em um servidor externo, para aprimoramento da função que calcula $\mathbf{TE_n}$. Isso permitiria ao ConCReCT se auto-adaptar, permitindo sua utilização em redes com perfil de tráfego heterogêneos, como cidades inteligentes. Outra importante evolução futura consiste na habilidade de monitoração dentro de cada nó permitindo que o ConCReCT possa identificar ataques não direcionados ao nó sorvedouro, distribuindo o impacto do custo computacional entre os nós da rede e ainda proporcionando um sistema de reputação entre os nós da rede.

Agradecimentos

Trabalho desenvolvido no âmbito do projeto de pesquisa PROMISE - Protocolos e Mecanismos para a Internet das Coisas Segura e Eficiente, n. 045491/UFSM.

Referências

Alam, K. M., Kamruzzaman, J., Karmakar, G., Murshed, M., and Azad, A. K. M. (2011). QoS support in event detection in WSN through optimal k-coverage. *Procedia Computer Science*, 4:499–507.

Alcaraz, C. and Lopez, J. (2010). A Security Analysis for Wireless Sensor Mesh Networks in Highly Critical Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(4):419–428.

Czyz, J., Kallitsis, M., Gharaibeh, M., Papadopoulos, C., Bailey, M., and Karir, M. (2014). Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks. In *Conference on Internet Measurement Conference - IMC 2014*, pages 435–448, New York, USA. ACM Press.

Dharini, N., Balakrishnan, R., and Renold, A. P. (2015). Distributed detection of flooding and gray hole attacks in Wireless Sensor Network. *International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, (May):178–184.

Dunkels, A. (2011). The ContikiMAC Radio Duty Cycling Protocol. *SICS Technical Report T2011:13 , ISSN 1100-3154*, pages 1–11.

Evans, P. C. and Annunziata, M. (2012). Industrial Internet: Pushing the Boundaries of Minds and Machines. *General Electric*, page 37.

Ghazali, K. W. M. and Hassan, R. (2011). Flooding distributed denial of service attacks-A review. *Journal of Computer Science*, 7(8):1218–1223.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

© 2017 Sociedade Brasileira de Computação

Karlof, C. (2003). Secure routing in wireless sensor networks: attacks and countermeasures. *Ad Hoc Networks*, 1(2-3):293–315.

Kasinathan, P., Pastrone, C., Spirito, M. A., and Vinkovits, M. (2013). Denial-of-Service detection in 6LoWPAN based Internet of Things. *International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 600–607.

Kwon, K., Ha, M., Kim, S. H., and Kim, D. (2013). TAMR: Traffic-aware multipath routing for fault tolerance in 6LoWPAN. *IEEE Global Telecommunications Conference - GLOBECOM*, pages 109–114.

Miorandi, D., Sicari, S., De Pellegrini, F., and Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516.

Moteiv Corporation (2006). Tmote Sky Datasheet. Disponivel em http://www.eecs.harvard.edu/˜konrad/projects/shimmer/references/tmote-sky-datasheet.pdf. Acessado em 17 de Junho de 2017.

Nigam, V., Jain, S., and Burse, K. (2014). Profile based scheme against DDoS attack in WSN. *2014 4th International Conference on Communication Systems and Network Technologies, CSNT 2014*, pages 112–116.

Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006). Cross-Level Sensor Network Simulation with COOJA. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648. IEEE.

Ploumis, S. E., Sgora, A., Kandris, D., and Vergados, D. D. (2012). Congestion Avoidance in Wireless Sensor Networks: A Survey. *2012 16th Panhellenic Conference on Informatics*, pages 234–239.

Raza, S., Wallgren, L., and Voigt, T. (2013). SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks*, 11(8):2661–2674.

Rughinis, R. and Gheorghe, L. (2010). Storm Control Mechanism in Wireless Sensor Networks. *9th Roedunet Ieee International Conference*, pages 430–435.

Sachan, R. S., Wazid, M., Singh, D. P., and Goudar, R. H. (2013). A cluster based intrusion detection and prevention technique for misdirection attack inside WSN. *International Conference on Communication and Signal Processing, ICCSP 2013*, pages 795–801.

Sassani, B. A., Abarro, C., Pitton, I., Young, C., and Mehdipour, F. (2016). Analysis of NTP DRDoS attacks' performance effects and mitigation techniques. pages 421–427.

Stavrou, E. and Pitsillides, A. (2014). Recovering from the selective forwarding attack in WSNs - Enhancing the recovery benefits of blacklisting and rerouting using directional antennas. *IWCMC 2014 - 10th International Wireless Communications and Mobile Computing Conference*, pages 299–303.

The MathWorks Inc. (2015). MATLAB Version R2015b. Disponivel em https://www.mathworks.com/. Acessado em 10 de Junho de 2017.

Wallgren, L., Raza, S., and Voigt, T. (2013). Routing Attacks and Countermeasures in the RPL-Based Internet of Things. *International Journal of Distributed Sensor Networks*, 9(8):794326.

Zargar, S. T., Joshi, J., Tipper, D., and Member, S. (2013). A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys & Tutorials*, 15(4):2046–2069.

 © 2017 Sociedade Brasileira de Computação

Uma Avaliação de Desempenho de Segurança Definida por Software através de Cadeias de Funções de Rede*

Igor J. Sanz[1], Igor D. Alvarenga[1], Martin Andreoni[13], Leopoldo A. F. Mauricio[1], Diogo M. F. Mattos[1], Marcelo G. Rubinstein[2] e Otto Carlos M. B. Duarte[1]

[1]Universidade Federal do Rio de Janeiro – GTA/UFRJ/PEE-COPPE – Brasil
[2]Universidade do Estado do Rio de Janeiro – FEN/DETEL/PEL – Brasil
[3]Université Pierre et Marie Curie/Sorbonne Universités – LIP6/CNR – França

Abstract. *Network Function Virtualization enables the provisioning and composition of an on-demand network function chain appropriated tailored to meet requirements of an application or service. This feature is necessary to provide security for enterprise networks and critical infrastructures, which depend on the processing in the core of the network. This paper proposes and develops virtual network security functions and evaluates their performance on the Open Platform for Network Function Virtualization (OPNFV). A prototype using the software defined networking technology and compliant with the Network Service Header (NSH) protocol has been designed. Our prototype shows a smart chaining of an intrusion and prevention detection system with a firewall that presents high flexibility without compromising the end-to-end delay.*

Resumo. *A Virtualização de Funções de Rede (Network Function Virtualization – NFV) possibilita o provisionamento e a composição sob demanda de uma cadeia de função de rede criada sob medida para atender requisitos de uma aplicação ou serviço. Esta funcionalidade é necessária para prover segurança de redes empresariais e infraestruturas críticas que dependem de processamento no núcleo da rede. Este artigo propõe e desenvolve funções virtuais de segurança de rede e avalia seus desempenhos na plataforma aberta para funções virtuais de rede, denominada OPNFV. Um protótipo usando a tecnologia de redes definidas por software e compatível com o protocolo Network Service Header foi projetado e desenvolvido. O protótipo realiza o encadeamento inteligente de um sistema de detecção e prevenção de intrusão com um firewall e apresenta alta flexibilidade sem comprometer o atraso fim-a-fim.*

1. Introdução

A tecnologia de Virtualização de Funções de Rede (*Network Function Virtualization* – NFV) tem como objetivos principais reduzir os gastos com *hardware* e com a implantação de novos serviços, reduzir a complexidade de operação e gerenciamento e melhorar a eficiência e o desempenho da rede através da otimização do consumo de recursos [Pattaranantakul et al. 2016]. Nesse novo modelo as funções de rede, tais como, tradução de endereços de rede (*Network Address Translation* – NAT), *firewall*, sistema de detecção e prevenção de intrusão (*Intrusion Detection and Prevention System* – IDPS)

*Este trabalho foi realizado com recursos do INCT INTERNET DO FUTURO, do CNPQ, da CAPES e da FAPERJ.

 © 2017 Sociedade Brasileira de Computação

e balanceador de carga, deixam de ser executadas em dispositivos de *hardware* dedicados para serem executadas em *softwares* que rodam em computadores de uso geral. Assim, os serviços de rede podem ser controlados de forma centralizada, migrados e implantados dinamicamente na rede para atender sob medida as necessidades de cada aplicação. Devido a essa flexibilidade, é possível definir zonas de segurança, redefinir a direção do tráfego para isolar elementos de rede comprometidos e impedir propagação de *malware* [Pattaranantakul et al. 2016, Reynaud et al. 2016]. O encadeamento de funções de serviço (*Service Function Chaining* – SFC) é o habilitador para o gerenciamento flexível do tráfego de um serviço ou aplicação [Medhat et al. 2017].

A tecnologia de Virtualização de Funções de Rede possui enormes desafios, pois requer uma execução eficiente do encadeamento das funções virtuais de rede (*Virtual Network Functions* – VNFs), além da correta ordenação das funções e a localização da função virtual na máquina física apropriada [Quinn and Nadeau 2015, Medhat et al. 2017]. No que se refere às funções de segurança, uma alta latência ou uma ordem incorreta destas funções pode resultar na não aplicação de uma política adequada sobre os pacotes, ocasionando vulnerabilidades e incidentes de segurança. Portanto, é importante identificar os principais gargalos de desempenho das plataformas de NFV, assim como das ferramentas de encadeamento de funções [Luizelli et al. 2017]. Além disso, a correta definição dos requisitos de recursos é necessária para evitar o sobre ou o subprovisionamento de recursos [Andreoni Lopez et al. 2016].

Este artigo provê o encadeamento de funções de serviço padronizado pelo *Internet Engineering Task Force* (IETF) [Halpern and Pignataro 2015], sobre uma arquitetura de virtualização de funções de rede descrita pelo *European Telecommunications Standards Institute* (ETSI) [ETSI 2014] e implementada através da *Open Platform for Network Function Virtualization* (OPNFV). O protótipo de segurança definida por *software* desenvolvido atende às especificações do protocolo de cabeçalho de serviço de rede (*Network Service Header* – NSH) e se serve da tecnologia de redes definidas por software (SDN) para rapidamente bloquear ataques. Foram desenvolvidos como funções virtuais de segurança de rede um sistema de detecção e prevenção de intrusão baseado em aprendizado de máquina com processamento em nuvem e um *firewall* configurável através de uma interface *RESTFull*. O protótipo implementado pelo encadeamento de funções virtuais de rede usando a OPNFV é pioneiro no Brasil e os resultados da avaliação de desempenho ressaltam importantes aspectos que se mostram os principais gargalos a serem vencidos. O desempenho da versão atual da plataforma mostra que o encadeamento de diversas funções de rede implica um forte decaimento da banda e um aumento linear da latência fim-a-fim, independentemente da topologia de realização física das funções, o que impõe restrições de serviço.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 aborda a virtualização de funções de rede e o estado da arte do encadeamento de funções de rede. A Seção 4 descreve o protótipo proposto e as funções virtuais de segurança. A Seção 5 expõe os detalhes da implementação e os resultados da avaliação de desempenho. Por fim, a Seção 6 conclui o artigo.

 © 2017 Sociedade Brasileira de Computação

2. Trabalhos Relacionados

A arquitetura de encadeamento, assim como a implementação das funções de serviço e o encapsulamento da cadeia de funções de serviço (SFC) variam entre diversas propostas. A proposta FlowTags é de um encapsulamento de SFC para que os sistemas intermediários sejam capazes de introduzir rótulos no cabeçalho dos pacotes e passar informações de contexto para os próximos sistemas intermediários em sequência, possibilitando a aplicação de políticas sobre o tráfego [Fayazbakhsh et al. 2013]. Já a proposta StEERING direciona os pacotes entre os sistemas intermediários sem adicionar rótulos [Zhang et al. 2013]. Para isso, a proposta utiliza múltiplas tabelas presentes no protocolo *OpenFlow* 1.0 para criar regras de encaminhamento hierárquicas que, a cada etapa do processamento, adicionam metadados ao tratamento do pacote e definem o próximo destino do encadeamento. A ferramenta ESCAPE se baseia na arquitetura de virtualização de funções de rede padronizada pelo ETSI para o desenvolvimento de protótipos de funções virtuais de rede. A ferramenta utiliza o NETCONF para o gerenciamento das funções e o controlador de rede SDN POX para fazer o encadeamento das funções [Csoma et al. 2014]. Paralelamente, a proposta Cloud4NFV apresenta uma arquitetura de virtualização de funções de redes baseada em quatro planos: infraestrutura, gerenciamento de infraestrutura virtual, orquestração e serviço [Callegati et al. 2015]. A ESCAPE e a Cloud4NFV são as propostas que mais se aproximam do padrão de arquitetura previsto pelo ETSI, mas ainda não empregam a definição de encadeamento de funções de rede do IETF [Halpern and Pignataro 2015].

Quanto ao desempenho do encadeamento de funções, Kulkarni *et al.* argumentam que o encapsulamento SFC com o cabeçalho NSH pode ter seu desempenho aprimorado utilizando uma nova semântica de identificação do SFP (*Service Function Path*) [Kulkarni et al. 2017]. Essa semântica se baseia em utilizar campos do cabeçalho NSH para definir uma sequência lógica entre tipos de VNFs, sem especificidade, e assim o controlador SDN pode decidir dinamicamente o melhor encadeamento disponível entre VNFs do mesmo tipo. A proposta NetBricks, por sua vez, desenvolve um encaminhamento de pacotes em funções virtuais de rede com zero cópia de memória. Para isso, usa-se memória compartilhada com passagens de referência pela área de memória [Panda et al. 2016]. Essa proposta traz consideráveis ganhos de desempenho das funções virtuais em termos de banda e latência. O desempenho do encadeamento de funções virtuais ainda é avaliado analiticamente e considerado nas restrições de problemas de otimização, como o da localização das VNFs na infraestrutura física. Luizelli *et al.* formalizam um modelo para encadeamento de funções usando programação linear inteira e propõem uma heurística que torna o atraso fim-a-fim da SFC comparável ao de sistemas intermediários físicos com uma alocação eficiente de recursos [Luizelli et al. 2015]. Lopez *et al.*, por sua vez, argumentam que a localização das funções virtuais sobre a infraestrutura física segue uma relação de compromisso entre aceitar um maior número de VNFs e o atraso do encadeamento em nós físicos mais distantes [Andreoni Lopez et al. 2016].

Outros trabalhos de avaliação de desempenho existentes na literatura também se relacionam com o encadeamento de funções de rede. Emmerich *et al.* avaliam o desempenho de comutadores virtuais e concluem que otimizações em configurações no núcleo do sistema operacional e o uso de CPU dedicada para a placa de rede são essenciais para aumentar o desempenho dos comutadores virtuais [Emmerich et al. 2014]. Callegati *et al.* avaliam o desempenho da virtualização de rede e dos principais componen-

© 2017 Sociedade Brasileira de Computação

tes do sistema operacional de nuvem *OpenStack* [Callegati et al. 2014] e Bonafiglia *et al.* comparam o desempenho de diferentes tecnologias de virtualização de funções de rede [Bonafiglia et al. 2015]. Bonafiglia *et al.* consideram configurações com comutadores virtuais e com comutadores com suporte a DPDK (*Data Plane Development Kit*), que transfere as operações de rede em *software* do ambiente virtual diretamente para serem executadas em *hardware* na interface física. Os autores comparam ainda o desempenho do encadeamento de funções virtuais de rede executadas em contêineres versus uma execução com virtualização completa. Os resultados mostram que ao dedicar núcleos de processamento para máquinas virtuais e usar comutadores com suporte a DPDK, o desempenho alcançado por máquinas virtuais é superior ao desempenho dos contêineres. No entanto, o encadeamento efetuado por Bonafiglia *et al.* não segue as normas da ETSI para NFV, nem da IETF para SFC, ou qualquer encapsulamento SFC.

Este artigo, diferentemente de todos os trabalhos citados acima, avalia o desempenho de funções virtuais de segurança encadeadas conforme a arquitetura de encadeamento de funções de rede proposta pelo IETF [Halpern and Pignataro 2015], usando o protocolo de cabeçalho de serviço de rede (NSH) e a plataforma aberta OPNFV. A plataforma OPNFV é compatível com arquitetura de virtualização de rede do ETSI [Rosa et al. 2014] e possui o controlador de redes definidas por software *OpenDayLight*. Portanto, toda função virtual implementada é NSH-consciente, permitindo a passagem de contexto entre funções. Além disso, este artigo avalia o desempenho de um protótipo de segurança definida por *software* composto pelo encadeamento de funções virtuais de segurança.

3. A Virtualização e o Encadeamento de Funções de Rede

A tecnologia de virtualização das funções de rede é a aplicação dos conceitos de computação em nuvem no domínio de redes de operadoras de telecomunicações [Medhat et al. 2017]. Para alcançar desempenho similar aos requisitos de sistemas intermediários em *hardware*, avanços recentes em computação em nuvem, como múltiplos hipervisores, virtualização assistida por *hardware*, sistemas operacionais de nuvem e comutadores por *software* eficientes, têm contribuído na implementação em *software* de funções virtuais de rede. Por sua vez a tecnologia de redes definidas por *software* (SDN) desacopla o plano de controle do plano de dados subjacente e consolida funções de controle em um controlador logicamente centralizado [Mattos and Duarte 2016]. O NFV e o SDN são tecnologias complementares, visto que, o gerenciamento das funções de rede se beneficia do controle SDN logicamente centralizado para configurar o plano de dados e encadear funções de rede coerentes. A Figura 1 mostra como a infraestrutura de virtualização de funções de rede (*NFV Infrastructure* – NFVI), em acordo com a arquitetura de gerência e orquestração das funções virtuais de rede (*Network Function Virtualization MANagement and Orchestration* – NFV-MANO) [ETSI 2014], pode compor microsserviços fim-a-fim sob medida para cada aplicação. A NFVI fornece as abstrações de processamento, armazenamento e acesso à rede às funções virtuais. Além disso, o controle do encaminhamento de pacotes e a consequente abstração da infraestrutura em um grafo de encadeamento de funções virtuais podem ser realizados de forma flexível pelo controle SDN [Han et al. 2015].

Funções de serviço e funções de rede são definidos como sinônimos neste trabalho, e o encadeamento coerente destas funções para cumprir um objetivo é denominado um serviço de rede. A ideia básica do encadeamento de funções de rede é oferecer funcio-

 © 2017 Sociedade Brasileira de Computação

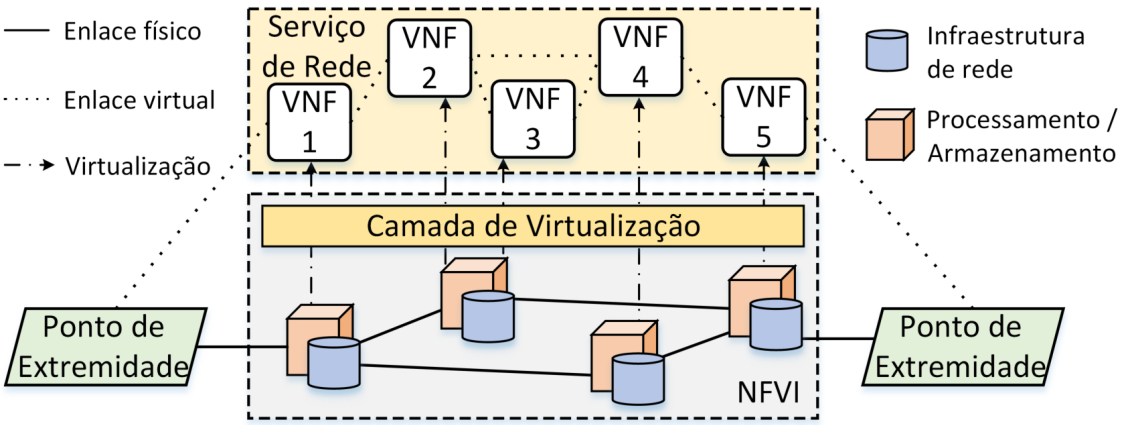


Figura 1. Infraestrutura de virtualização de funções de rede que possibilita a criação de enlaces virtuais para microsserviços fim-a-fim através do encadeamento de funções virtuais de rede (VNF).

nalidades através do encadeamento de funções virtuais que executam funções específicas da camada rede, tais como detecção de intrusão, *firewall*, *proxy* etc. As funções virtuais de rede substituem os inúmeros elementos intermediário de *hardware* que existem hoje (*middleboxes*). A arquitetura básica do encadeamento de funções de rede (*Service Function Chaining* – SFC) conforme a RFC 7665 é apresentada na Figura 2. Nesta arquitetura, o caminho virtual de um serviço, que envolve uma sequência ordenada de VNFs, é definido como caminho de funções de serviço (*Service Function Path* – SFP). Para que o tráfego de um determinado serviço percorra o SFP desejado, um classificador de fluxos é configurado com regras específicas determinando qual SFP cada fluxo deve percorrer. Dessa forma, tráfegos de diferentes microsserviços podem percorrer simultaneamente as mesmas VNFs, mas seguindo caminhos lógicos distintos e isolados em função do identificador de caminho de cada fluxo. As VNFs também podem estar hospedadas em nós físicos diferentes. Por isso, o encaminhador de funções de serviço (*Service Function Forwarder* – SFF) é um elemento necessário em cada nó da NFVI que fornece os enlaces virtuais para suas VNFs hóspedes. Por sua vez, as VNFs podem ser conscientes ou não do encapsulamento da SFC. As VNFs que não são conscientes do encapsulamento podem operar normalmente contanto que sejam precedidas por um elemento que faça o desencapsulamento dos pacotes. Este elemento é chamado de *Proxy* SFC. Na implementação deste artigo, os elementos da arquitetura SFC são construídos baseados em regras através do comutador virtual *Open vSwitch* usando o protocolo *OpenFlow*. Essas regras são gerenciadas pelo controlador SDN *OpenDaylight* aliado a um gerenciador e orquestrador de VNFs denominado *Tacker*.

Para orientar o encaminhamento dinâmico de fluxos em diferentes caminhos virtuais de serviço, como por exemplo para balancear carga ou para conseguir um controle granular e preciso de políticas, é necessário fazer um encapsulamento para marcar os pacotes. A plataforma OPNFV especifica o protocolo de cabeçalho de serviço de rede (*Network Service Header* – NSH) para o encapsulamento da SFC [Quinn and Elzur 2017]. O NSH fornece visibilidade do caminho virtual do serviço de ponta-a-ponta sobre o protocolo de transporte UDP e permite classificar e reclassificar o fluxo em cada função de serviço. Este direcionamento do tráfego é feito a partir de dois campos principais do protocolo. O primeiro, responsável por identificar unicamente cada caminho SFP, é o identificador de

© 2017 Sociedade Brasileira de Computação

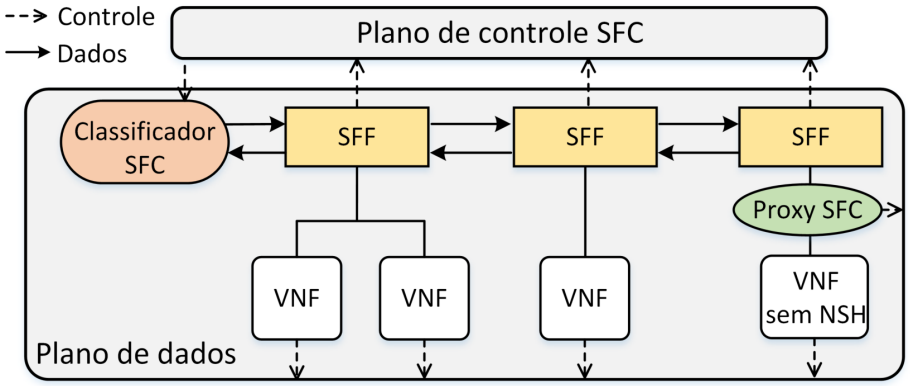


Figura 2. Elementos básicos da arquitetura de Encadeamento de Funções de Rede conforme a RFC 7665.

caminho de serviço (*Service Path Identifier* – SPI), enquanto o segundo, que identifica a posição na cadeia, é o índice de função de serviço (*Service Index* – SI). Este índice é um iterador de 8 bits que decresce uma unidade de 255 a cada salto nas funções de serviço, até atingir um valor referente ao tamanho final da cadeia. Outra vantagem do encadeamento com o NSH é o intercâmbio de metadados entre as VNFs, que é possível devido à existência de campos de contexto. Este aspecto é positivo para os casos de uso de internet móvel ou de provimento de acesso para usuários fixos domésticos em que o contexto pode conter o identificador do assinante ou do inquilino, de forma a permitir políticas baseadas em contratos [Kulkarni et al. 2017].

4. O Protótipo de Segurança Definida por *Software*

Um dos principais usos para a virtualização de funções de rede corresponde à garantia da aplicação correta, coerente e consistente de políticas ao tráfego de rede [Zhang et al. 2013]. No protótipo implementado, as duas principais funções de segurança são o sistema de detecção e prevenção de intrusão e o *firewall*. O primeiro é responsável por analisar os pacotes e gerar alarmes, enquanto o segundo implementa regras de filtragem de pacotes para gerar perímetros de segurança na rede. O encadeamento inteligente destas funções virtuais, doravante denominado como segurança definida por *software*, fornece flexibilidade para instanciar mecanismos complexos de segurança em tempo real e em qualquer ponto da rede. A Figura 3 mostra a arquitetura do protótipo, cuja a ideia central é associar o processamento provido por uma nuvem de processamento por fluxo *Spark* com a flexibilidade de encadeamento das funções de segurança e ainda possibilitar uma ação de resposta em tempo real ao tráfego malicioso detectado. Sua principal vantagem é poder escalar a tarefa de processamento dos fluxos com o espelhamento e o tratamento dos fluxos em nuvem, portanto, de forma independente da infraestrutura de virtualização de funções de rede.

O elemento principal da arquitetura é o sistema de detecção e prevenção de intrusão que utiliza uma tecnologia de processamento por fluxo para fazer a análise do tráfego em tempo real [Andreoni Lopez et al. 2017]. Os pacotes são capturados, através do espelhamento de tráfego, pelo módulo do IDPS que atua diretamente na cadeia. Estes pacotes são abstraídos em fluxos, que são definidos como uma sequência de pacotes com a mesma quíntupla IP de origem, IP de destino, porta de origem, porta de destino e protocolo, durante uma janela de tempo. Ao todo, 46 características de fluxo são extraí-

 © 2017 Sociedade Brasileira de Computação

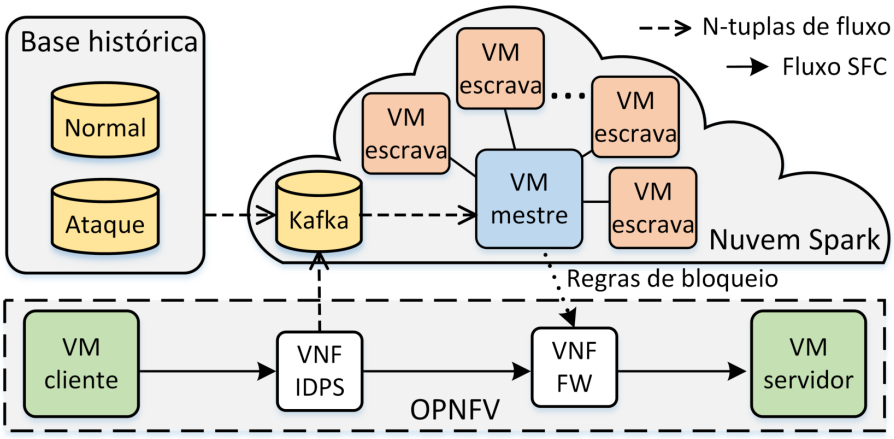


Figura 3. Arquitetura do protótipo de Segurança Definida por *Software* através do encadeamento de um sistema de detecção e prevenção de intrusão e de um *firewall* virtuais.

das e publicadas em um serviço produtor/consumidor de mensagens *Apache Kafka*. Este serviço opera como um sistema de fila e manipulação de fluxo de dados a baixa latência, onde as características da fila são consumidas pelo módulo de classificação. O módulo de classificação, por sua vez, é instanciado em uma nuvem dedicada para a classificação e possui um núcleo principal de processamento, o *Apache Spark*. O *Spark* é implementado em um aglomerado de máquinas no modelo mestre/escravo, onde os escravos possuem a capacidade de expansão e redução de recursos, tornando o sistema escalável.

O IDPS implementado utiliza a arquitetura lambda [Marz and Warren 2013], composta por três camadas: a camada de processamento de fluxo de dados, a camada de processamento em lotes e a camada de serviço. A camada de processamento de fluxo de dados trata o fluxo de dados em tempo real. A camada de processamento em lotes analisa grande quantidade de dados armazenados através de técnicas de computação distribuída como *map-reduce*. Finalmente, a camada de serviço agrega as informações obtidas das duas camadas anteriores para criar as saídas dos dados analisados. O IDPS classifica os fluxos em maliciosos ou benignos através de algoritmos de aprendizado de máquina (*Machine Learning* – ML) baseados em árvores de decisão. Uma base histórica com dados de tráfegos marcados como ataques ou normais é utilizada treinar o algoritmo ML de maneira *offline*. Os parâmetros calculados no processamento *offline* com os dados históricos servem para ajustar o modelo de processamento em tempo real. Assim, o sistema possui uma característica adaptativa, pois os parâmetros podem ser atualizados, se ajustando a novos padrões de uso. Após o treino, o IDPS permite uma classificação do tráfego em tempo real e, ao detectar fluxos maliciosos, é capaz de enviar regras de bloqueio para o *firewall* implementado.

A VNF *firewall* implementada nesse trabalho possui um módulo capaz de encapsular e desencapsular o pacote NSH baseado na aplicação *Python* de código aberto *vxlan_tool*[1]. Esta aplicação é estendida para conter um dicionário *Python* que armazena regras de bloqueio baseadas nas quíntuplas do fluxo. Antes do encaminhamento do pacote para a próxima VNF da cadeia, a quíntupla do pacote é verificada quanto ao conjunto de regras existentes. Uma interface *RESTful* é implementada ainda para possibilitar entradas

[1]Disponível em https://github.com/opendaylight/sfc/blob/master/sfc-test/nsh-tools/vxlan_tool.py.

© 2017 Sociedade Brasileira de Computação

e remoções de regras no dicionário. Assim, a adição ou remoção de regras de bloqueio no *firewall* é feita por chamadas REST em tempo real pelo IDPS.

5. Implementação e Resultados

Um protótipo de Segurança Definida por *Software* construído através do encadeamento das funções IDPS e *firewall* é implementado conforme a Figura 3 e seu desempenho e eficiência são avaliados para diferentes topologias e configurações.

A plataforma utilizada para implementação dos testes e do protótipo é a plataforma aberta *Open Platform for Network Function Virtualization* (OPNFV) na versão Danube 2.0[2]. Esta plataforma implementa a arquitetura de referência NFV-MANO usando como base o sistema operacional de nuvem *OpenStack*[3] A plataforma OPNFV é construída através do instalador *Fuel* conforme no cenário que fornece a arquitetura de Encadeamento de Funções de Rede referenciada na RFC 7665. Este cenário utiliza ainda o controlador SDN *OpenDaylight*, responsável pelo gerenciamento da camada de enlace, o gerenciador e orquestrador de VNFs *Tacker* e o comutador virtual *Open vSwitch* com suporte ao protocolo NSH. As versões utilizadas desses softwares são ramificações que foram adaptadas para integração à plataforma OPNFV e que estão disponíveis publicamente na versão Danube 2.0 da plataforma. Deve ser ressaltado que essa plataforma ainda se encontra em fase de desenvolvimento, com documentação precária, e muito erros tiveram que ser depurados e corrigidos pela equipe. Este protótipo de cadeia de funções de rede usando NSH é com certeza uma implementação pioneira no Brasil.

As máquinas usadas para criação do ambiente de testes consistem em um nó controlador Intel 8-Core i7-4770 CPU, 3,40 GHz com 32 GB de memória, e três nós de processamento, Intel Xeon CPU X5570 2,93 GHz com 96 GB de memória (nó 1), Intel 8-Core i7-6700 CPU, 3.40 GHz com 64 GB de memória (nó 2) e Intel 8-Core i7-2600 CPU, 3,40 GHz com 32 GB de memória (nó 3). Todas as máquinas são interligadas através de um comutador topo de *rack* por interfaces de rede de 1 Gb/s que englobam as cinco redes VLANs necessárias para a instalação da nuvem OPNFV: pública, privada, de gerenciamento, de armazenamento e de *Preboot Execution Environment*.

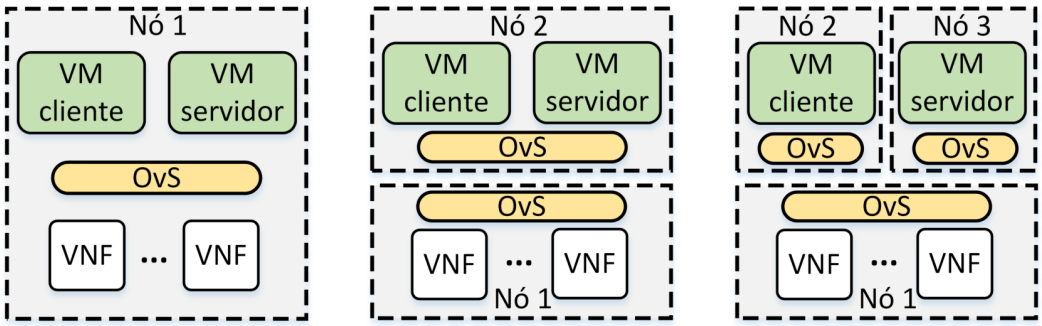
A avaliação de desempenho do encadeamento de funções virtuais de segurança é baseada na arquitetura da RFC 7665 usando o protocolo NSH, como mostrado na Figura 2. Para isso, são analisados diversos casos de uso que impactam o desempenho do encadeamento das funções virtuais de segurança. As medidas estudadas neste trabalho incluem a posição das máquinas virtuais do cliente e do servidor e a cadeia de VNFs instanciados na nuvem, a sobrecarga que cada função de rede introduz na cadeia e a quantidade de núcleos virtuais de processamento demandados pela VNF. Além disso, é avaliado o comportamento na cadeia de uma VNF *firewall* ao adicionar um número alto de regras de bloqueio. Por fim, o protótipo implementado que encadeia a função IDPS e a função *firewall* é avaliado quanto à acurácia de classificação *online* na cadeia em relação a classificação *offline*, fora da cadeia, e à eficiência na detecção em tempo real e no bloqueio automático de fluxos maliciosos.

As topologias escolhidas para avaliar o encadeamento são ilustradas na Figura 4. A primeira topologia, ilustrada na Figura 4(a), utiliza apenas um nó da nuvem, uma má-

[2]Disponível em https://www.opnfv.org.

[3]Disponível em https://www.openstack.org.

 © 2017 Sociedade Brasileira de Computação



(a) Topologia 1. (b) Topologia 2. (c) Topologia 3.

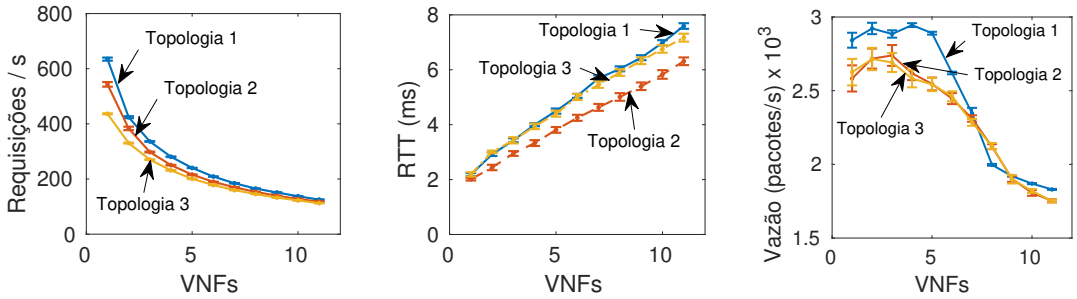
Figura 4. Topologias utilizadas para avaliar o desempenho do encadeamento de funções virtuais de rede: a) cliente, servidor e cadeia de VNFs no mesmo nó; b) cliente e servidor em um nó separado do nó da cadeia; c) cliente, servidor e cadeia em três nós separados.

quina física, para instanciar a cadeia de funções virtuais e as VMs do pontos de extremidade. Dessa forma, as máquinas virtuais da extremidade da cadeia e a própria cadeia de VNFs competem por recursos de rede do mesmo nó, ao mesmo tempo que não há saltos entre os enlaces físicos dos nós. A Figura 4(b) define a Topologia 2, em que as VMs da extremidade são instanciadas em um mesmo nó, mesma máquina física, e a cadeia de VNFs se encontra em outro nó, uma outra máquina física. A terceira topologia, mostrada na Figura 4(c), a VM servidor, a VM cliente e a cadeia de VNFs se encontram em três nós distintos. Dessa forma, as Topologias 2 e 3 demandam o uso de mais comutadores virtuais, uma vez que cada nó possui um comutador virtual *Open vSwitch* local para controlar seus recursos de rede.

A Figura 5 mostra os resultados de impacto no desempenho obtidos em cada topologia em função do número de VNFs da cadeia, considerando um intervalo de confiança de 95%. Vale ressaltar que as VNFs usadas neste cenário são funções de rede extremamente simples que apenas encaminham os pacotes recebidos para a próxima VNF da cadeia. Este encaminhamento é feito desencapsulando o pacote do protocolo NSH, em que o campo *Network Service Index* do cabeçalho é decrescido de 1 e, assim, o pacote é retornado ao *Open vSwitch* que o envia à próxima função virtual. Assim, o processamento e o atraso é praticamente somente devido ao desencapsulamento e reencapsulamento com o protocolo NSH. A aplicação responsável por estas operações é uma ferramenta escrita em *Python* (*vxlan_tool*) para testes com o cabeçalho NSH. A Figura 5(a) compara as três topologias em relação a taxa de requisições HTTP feitas a partir de uma VM cliente para uma VM servidor que atravessa a cadeia. Nota-se que a Topologia 1 fornece a melhor taxa de requisições HTTP para cadeias curtas de VNFs. No entanto, essa diferença se torna desprezível quando o comprimento da cadeia ultrapassa 8 VNFs. Isso demonstra que, para cadeias curtas, a sobrecarga introduzida por separar o cliente, servidor e as VNFs em diferentes nós é o fator limitante do desempenho. No entanto, cadeias longas introduzem uma sobrecarga que ultrapassa este fator. Já a Figura 5(b) mostra que o tempo de ida e volta em todas as topologias cresce linearmente com o aumento do tamanho da cadeia. A Topologia 2, entretanto, apresentou um aumento de latência significantemente menor que as outras duas. Isto pode ser explicado em função de que na Topologia 2 as VMs cliente e servidor estão no mesmo nó físico, o que diminui o tempo da volta do pacote, e ainda, este nó não compartilha recursos com outras VNFs. Pode-se concluir então que o au-

© 2017 Sociedade Brasileira de Computação

mento de saltos entre enlace físicos dos nós, assim como o compartilhamento de recursos de um mesmo nó, são fatores que comprometem a latência. Deste modo, a Topologia 2 encontra-se ao meio termo no compromisso desses fatores. O gráfico da Figura 5(c), por sua vez, mostra a vazão máxima medida em pacotes por segundo para cada topologia ao aumentar o tamanho da cadeia. O tamanho de pacote usado nos experimentos deste trabalho é 1334 bytes, que representa o tamanho máximo do pacote *Ethernet* enviado pelo cliente, para que ao ser encapsulado com NSH não haja fragmentação IP. A Topologia 1 apresentou uma melhor vazão em relação às demais para o encadeamento de poucas VNFs por evitar saltos pelos enlaces físicos entre os nós. Percebe-se, no entanto, que o aumento de VNFs em todas as topologias gera competição de recursos do nó que hospeda a cadeia, o que compromete consideravelmente a vazão.



(a) Taxa máx. de req. HTTP em função do comprimento da cadeia.

(b) Tempo de ida e volta em função do comprimento da cadeia.

(c) Vazão máxima em função do comprimento da cadeia.

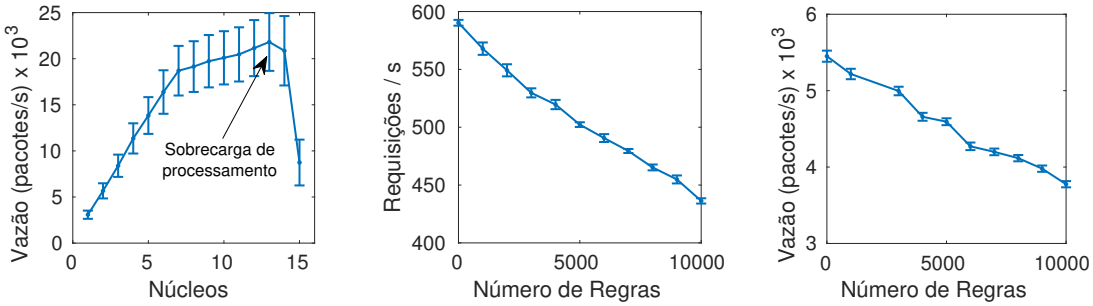
Figura 5. Impacto no desempenho de cadeias de funções de rede ao aumentar o número de VNFs encadeadas avaliado pelas métricas: a) taxa de requisições HTTP suportada; b) tempo de ida e volta; e c) vazão máxima da cadeia.

O maior fator limitante da vazão nestes resultados é a ferramenta *vxlan_tool* que desencapsula os pacotes NSH. Essa aplicação, por padrão, opera de forma sequencial em apenas um núcleo de processamento, mas foi estendida para operar paralelamente em vários núcleos. O efeito desta mudança é observado ao analisar a Figura 6(a), que mostra o aumento da vazão de uma VNF em uma cadeia de tamanho unitário ao alocar-se mais núcleos virtuais dedicados do hipervisor. Dessa forma, a VNF passa a reter mais poder de processamento e consegue efetuar mais operações com pacotes por segundo até atingir o limite de processamento do hipervisor. A plataforma OPNFV, no entanto, já prevê em suas próximas versões que esta função de desencapsulamento seja implementada no núcleo do sistema operacional para ganhos em desempenho.

Para avaliar o desempenho de diferentes funções virtuais de segurança, são implementadas as VNFs *firewall* e IDPS descritas na Seção 4 em cadeias individuais, considerando a Topologia 1. As Figuras 6(b) e 6(c) caracterizam as taxas máximas de requisições HTTP e a vazão máxima da cadeia da VNF *firewall* ao aumentar o número de regras de bloqueio. Observa-se uma redução linear tanto da vazão máxima quanto do número máximo de requisições com o aumento da quantidade de regras impostas. No entanto, essa redução impacta o desempenho em apenas 1% para um uso considerável de 500 regras no *firewall*. O aumento do número de regras também não interferiu significativamente na latência do pacote que atravessa o *firewall*, que manteve-se a uma média próxima de 2 ms.

A segunda função de segurança avaliada é o IDPS, que nesta implementação é

 © 2017 Sociedade Brasileira de Computação



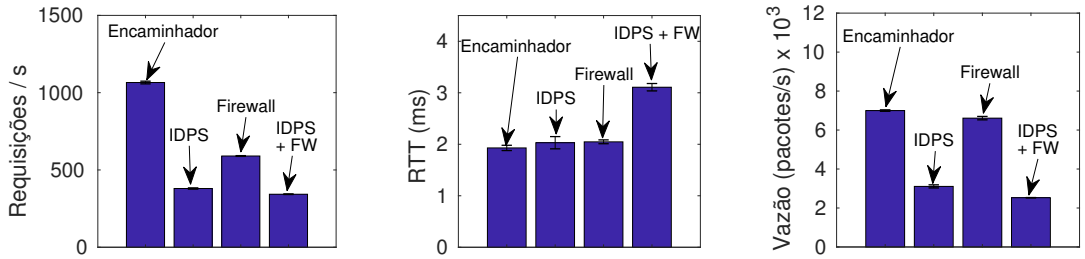
(a) Vazão máxima em função do n^o núcleos virtuais atribuídos.

(b) Taxa máx. de requis. HTTP em função do n^o regras do *FW*.

(c) Vazão máxima em função do n^o de regras do *FW*.

Figura 6. (a) Acréscimo da vazão ao aumentar os núcleos dedicados do hipervisor para a VNF. Impacto no desempenho da (b) taxa máxima de requisições HTTP e da (c) vazão máxima em função do número de regras da VNF *firewall*.

composto por dois módulos descritos na Seção 4. O primeiro módulo atua diretamente na cadeia e realiza duas funções básicas no tráfego, extrair características dos fluxos em janelas de dois segundos e, ao mesmo tempo, encaminhar os pacotes para a próxima VNF. Além disso, a VNF que implementa o IDPS publica as características extraídas no sistema de fila e manipulação de fluxo de dados *Kafka*, para serem lidos pelo segundo módulo de processamento que opera fora da cadeia.



(a) Taxa máxima de requisições HTTP em cada cadeia.

(b) Tempo de ida e volta médio em cada cadeia.

(c) Vazão máxima suportada em cada cadeia.

Figura 7. Impacto no desempenho introduzido por cada função virtual de segurança e pelo encadeamento das duas funções em relação à: (a) taxa máxima de requisições HTTP; (b) tempo de ida e volta do pacote; e (c) vazão máxima.

A Figura 7 mostra o desempenho de cadeias simples com diferentes funções virtuais de rede e ainda do encadeamento composto por duas funções virtuais de segurança. Em todas as cadeias é utilizada a Topologia 1 como referência, fornecendo apenas um núcleo virtual para cada VNF e sem nenhuma regra introduzida no *firewall* na etapa de caracterização. Pode-se observar que o desempenho da VNF que somente encaminha pacotes é superior ao das outras funções virtuais e por isso é usado como patamar inicial de referência. A Figura 7(a) e a Figura 7(c) mostram resultados similares em relação à taxa máxima de requisições HTTP e à vazão suportada de cada cadeia. A VNF *firewall* apresentou melhores resultados que a VNF IDPS nestas duas métricas, por isso, o encadeamento do IDPS com o *firewall* tem seu desempenho limitado pelo IDPS, com uma pequena sobrecarga devido ao encadeamento de duas VNFs. No entanto, a Figura 7(b) mostra que a sobrecarga em latência introduzida por cada função virtual de segurança é bem pequena, mantendo-se a um tempo similar ao patamar inicial. Já o encadeamento

© 2017 Sociedade Brasileira de Computação

composto pelas duas funções de segurança aumenta em 50% no tempo de ida e volta do pacote, o que já era previsto a partir da Figura 5(b). No entanto, este valor é 1,33 vezes menor do caso em que as funções *firewall* e IDPS atuassem no tráfego separadamente.

Por fim, o protótipo é testado quanto à classificação em tempo real de um conjunto de dados de usuários de uma operadora de telecomunicações brasileira. O algoritmo de aprendizado de máquina implementado é primeiro treinado por uma fração do tráfego que foi rotulada como ataque ou normal pelo IDS Suricata[4]. O restante do tráfego é injetado a partir do cliente em direção ao servidor e atravessa as duas funções virtuais de segurança. A Tabela 1 mostra os resultados da classificação[5] comparando os fluxos que atingiram o servidor com os que foram bloqueados no meio do caminho pelo *firewall*. Os resultados mostram uma acurácia de 72,7% na classificação e que 0,02% dos fluxos maliciosos foram bloqueados antes de serem iniciados no servidor. No entanto, como o *firewall* proposto é reativo, há uma redução real de 15% do volume em bytes do tráfego malicioso total que chega ao servidor. Tal fato é resultado do tempo necessário para análise do fluxo até que seja caracterizado um comportamento malicioso. Fluxos curtos (camundongos) de ataque têm duração menor que a janela de análise e, portanto, não são possíveis de serem bloqueados em tempo real. Isto mostra que o protótipo foi eficaz contra ataques com grande volumes de dados, como os de negação de serviço.

Tabela 1. Matriz de confusão da classificação e bloqueio de fluxos em tempo real. Os fluxos que atingem o servidor são representados como Normal, enquanto os bloqueados no meio do caminho como Ataque.

	VP	**FP**	**VN**	**FN**
Ataque	430	2277	4412795	1658973
Normal	4412795	1658973	430	2277

6. Conclusão

Este trabalho apresentou uma avaliação de desempenho do encadeamento de funções virtuais de segurança conforme as normas da arquitetura de referência NFV-MANO da ETSI, da arquitetura de encadeamento de funções de rede da RFC 7665 e do protocolo *Network Service Header*. O ambiente de testes utilizado para criação das cadeias foi a plataforma de código aberto OPNFV, que usa como base o sistema operacional de nuvens *OpenStack*. Esse artigo propôs ainda duas funções virtuais de segurança em rede: um sistema de detecção e prevenção de intrusão baseado no processamento de dados na nuvem e um *firewall* configurável por uma interface RESTful. Foi avaliado o impacto no desempenho em termos de vazão, tempo de ida e volta e taxa de requisições HTTP causado por diferentes topologias de implementação da cadeia, núcleos virtuais fornecidos às VNFs, configurações das VNFs e sobrecarga acrescentada por cada função virtual. Ressalta-se que tanto os saltos entre enlaces físicos dos nós, quanto a competição de recursos em um mesmo nó, são fatores que impactam o desempenho do encadeamento destas funções. Ainda, a vazão está diretamente condicionada à quantidade de núcleos fornecidos à função virtual, que define a quantidade de pacotes que cada VNF é capaz de processar. A implementação realizada na plataforma OPNFV usando encadeamento de funções de rede (SFC) com o protocolo de cabeçalho de serviço de rede (NSH) é seguramente pioneira no Brasil. A versão atual do protocolo NSH executado pela ferramenta *vxlan_tool*

[4]Disponível em https://suricata-ids.org.

[5]VP: Verdadeiros Positivos; FP: Falsos Positivos; VN: Verdadeiros Negativos; FN: Falsos Negativos.

 © 2017 Sociedade Brasileira de Computação

em *Python* é certamente o principal fator limitante da plataforma. Esta limitação deve ser eliminada nas próximas versões com a sua implementação em *kernel*.

Como trabalhos futuros, pretende-se avaliar o desempenho do encadeamento de funções em novas topologias, como alternar a localização de cada VNF da cadeia e ainda implementar e avaliar novas funções virtuais de segurança, como funções que criptografam e descriptografam o tráfego.

Referências

Andreoni Lopez, M., Lobato, A. G. P., Mattos, D. M. F., Alvarenga, I. D., Duarte, O. C. M. B., and Pujolle, G. (2017). Um Algoritmo Não Supervisionado e Rápido para Seleção de Características em Classificação de Tráfego. In *SBRC'2017*, Belém/PA.

Andreoni Lopez, M., Mattos, D. M. F., and Duarte, O. C. M. B. (2016). Evaluating allocation heuristics for an efficient virtual network function chaining. In *7th International Conference on the Network of the Future (NoF)*, pages 1–5.

Bonafiglia, R., Cerrato, I., Ciaccia, F., Nemirovsky, M., and Risso, F. (2015). Assessing the performance of virtualization technologies for NFV: A preliminary benchmarking. In *2015 Fourth European Workshop on Software Defined Networks*, pages 67–72.

Callegati, F., Cerroni, W., Contoli, C., and Santandrea, G. (2014). Performance of network virtualization in cloud computing infrastructures: The openstack case. In *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 132–137.

Callegati, F., Cerroni, W., Contoli, C., and Santandrea, G. (2015). Dynamic chaining of virtual network functions in cloud-based edge networks. In *1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5.

Csoma, A., Sonkoly, B., Csikor, L., Németh, F., Gulyas, A., Tavernier, W., and Sahhaf, S. (2014). ESCAPE: Extensible service chain prototyping environment using Mininet, Click, NETCONF and POX. In *ACM Conference on SIGCOMM*, SIGCOMM '14, pages 125–126, New York, NY, USA. ACM.

Emmerich, P., Raumer, D., Wohlfart, F., and Carle, G. (2014). Performance characteristics of virtual switching. In *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 120–125.

ETSI (2014). ETSI GS NFV-MAN 001: Network functions virtualisation; management and orchestration. Technical report.

Fayazbakhsh, S. K., Sekar, V., Yu, M., and Mogul, J. C. (2013). Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *II ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 19–24, New York, NY, USA. ACM.

Halpern, J. and Pignataro, C. (2015). Service Function Chaining (SFC) architecture. RFC 7665, RFC Editor. `http://www.rfc-editor.org/rfc/rfc7665.txt`.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

© 2017 Sociedade Brasileira de Computação

Kulkarni, S., Arumaithurai, M., Ramakrishnan, K. K., and Fu, X. (2017). Neo-NSH: Towards scalable and efficient dynamic service function chaining of elastic network functions. In *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 308–312.

Luizelli, M. C., Bays, L. R., Buriol, L. S., Barcellos, M. P., and Gaspary, L. P. (2015). Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 98–106.

Luizelli, M. C., Raz, D., Sa'ar, Y., and Yallouz, J. (2017). The actual cost of software switching for nfv chaining. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 335–343.

Marz, N. and Warren, J. (2013). *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

Mattos, D. M. F. and Duarte, O. C. M. B. (2016). Authflow: authentication and access control mechanism for software defined networking. *Annals of Telecommunications*, 71(11):607–615.

Medhat, A. M., Taleb, T., Elmangoush, A., Carella, G. A., Covaci, S., and Magedanz, T. (2017). Service function chaining in next generation networks: State of the art and research challenges. *IEEE Communications Magazine*, 55(2):216–223.

Panda, A., Han, S., Jang, K., Walls, M., Ratnasamy, S., and Shenker, S. (2016). Netbricks: Taking the v out of NFV. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, pages 203–216. USENIX Association.

Pattaranantakul, M., He, R., Meddahi, A., and Zhang, Z. (2016). SecMANO: Towards network functions virtualization (nfv) based security management and orchestration. In *IEEE Trustcom/BigDataSE/ISPA*, pages 598–605.

Quinn, P. and Elzur, U. (2017). Network service header. Internet-Draft draft-ietf-sfc-nsh-12, IETF Secretariat. `http://www.ietf.org/internet-drafts/draft-ietf-sfc-nsh-12.txt`.

Quinn, P. and Nadeau, T. (2015). Problem statement for service function chaining. RFC 7498, RFC Editor. `http://www.rfc-editor.org/rfc/rfc7498.txt`.

Reynaud, F., Aguessy, F. X., Bettan, O., Bouet, M., and Conan, V. (2016). Attacks against network functions virtualization and software-defined networking: State-of-the-art. In *IEEE NetSoft Conference and Workshops (NetSoft)*, pages 471–476.

Rosa, R., Siqueira, M., Barea, E., Marcondes, C., and Rothenberg, C. (2014). Network function virtualization: Perspectivas, realidades e desafios. In *Minicursos do XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2014*.

Zhang, Y., Beheshti, N., Beliveau, L., Lefebvre, G., Manghirmalani, R., Mishra, R., Patneyt, R., Shirazipour, M., Subrahmaniam, R., Truchan, C., and Tatipamula, M. (2013). Steering: A software-defined networking for inline service chaining. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10.

 © 2017 Sociedade Brasileira de Computação

Análise de Séries Temporais Multivariadas para Detecção de Intrusão em Nuvens Computacionais

Mayk Fernando Choji[1], **Paulo Lício de Geus**[1]

[1]Instituto de Computação
Universidade Estadual de Campinas (UNICAMP)
Campinas, SP – Brasil

{mayk,paulo}@lasca.ic.unicamp.br

Abstract. *Despite the numerous advantages offered by cloud computing services, security aspects constitute a critical issue when considering to adopt this kind of service. Novel intrusion detection methods have been proposed in order to mitigate attacks toward the several layers from cloud architectures. In this paper, we present an intrusion detection system for IaaS clouds. Our approach relies on the analysis of performance metrics gathered by virtual machine introspection techniques, which allows one to monitor clients directly from the host machine. Anomaly detection is performed by treating the collected data as multivariate time series.*

Resumo. *Apesar das inúmeras vantagens oferecidas por serviços de computação em nuvem, aspectos de segurança constituem um dos pontos críticos ao se considerar adotar este tipo de serviço. Novos métodos de detecção de intrusão têm sido propostos para mitigar ataques direcionados às várias camadas da arquitetura de nuvem. Neste trabalho, apresentamos um sistema de detecção de intrusão para nuvens do modelo IaaS. A abordagem baseia-se na análise de métricas de desempenho obtidas por meio de técnicas de introspecção de máquina virtual, permitindo a monitoramento de clientes a partir da máquina hospedeira. Anomalias são identificadas tratando-se os dados coletados como séries temporais multivariadas.*

1. Introdução

Embora vários estudos tenham sido realizados sobre técnicas de detecção de intrusão [Liao et al. 2013, Zhu and Zheng 2008], a chegada de novos paradigmas como a computação em nuvem traz consigo novos desafios na área de segurança da informação. A dinamicidade intrínseca dessas arquiteturas, somada a fatores de privacidade e confidencialidade de dados entre cliente e provedor de serviço, por exemplo, faz com que sistemas de segurança tradicionais tenham que ser repensados para atender às novas demandas.

Neste trabalho, utilizamos métricas de desempenho coletadas de máquinas virtuais (VMs) de forma não-invasiva para detectar anomalias de segurança utilizando um algoritmo baseado em percurso aleatório sobre grafo. Nosso objetivo é apresentar um sistema de detecção de intrusão capaz de identificar anomalias em máquinas virtuais utilizando somente dados coletados a partir do *hypervisor* e que não tenha a necessidade de submeter o sistema a testes de ataque definidos *a priori*. Esta abordagem visa garantir a

 © 2017 Sociedade Brasileira de Computação

integridade do sistema e a continuidade dos serviços contratados por clientes de nuvens computacionais.

O conjunto de métricas coletadas ao longo do tempo é tratado como uma série temporal multivariada, e a indução de um grafo pela matriz *kernel* desses dados possibilita descrever cada observação em termos de sua conectividade, ou semelhança, com as demais observações. Partindo do pressuposto de que ataques causem perturbações no desempenho da máquina virtual, observamos que essas ocorrências mapeiam-se como nós com baixa conectividade no grafo, destacando as anomalias. O intervalo de coleta de dados, métricas utilizadas e níveis de segurança são controlados pela arquitetura de gerenciamento de níveis de acordo de serviço (SLA) introduzida em [Ferreira 2013].

O restante deste artigo está organizado da seguinte forma. Na Seção 2 descrevemos a arquitetura de nuvem e a técnica utilizada para detecção de anomalias. Além disso, descrevemos o método para geração de tráfego *web* adotado nos testes. Detalhes sobre os sistemas computacionais e os parâmetros do sistema de detecção de intrusão (IDS) são descritos na Seção 3. Na Seção 4 mostramos os resultados obtidos pelo método implementado ao analisarmos ataques de *port scan* e ataques distribuídos de negação de serviço (*DDoS*). Uma breve discussão sobre trabalhos anteriores é feita na Seção 5, e a Seção 6 conclui este trabalho e indica pesquisas futuras.

2. Metodologia

Nesta Seção, apresentamos a metodologia utilizada neste trabalho, incluindo a organização de máquinas físicas e virtuais, e as principais ferramentas utilizadas.

2.1. Arquitetura

Conforme esquematizado na Figura 1, a arquitetura lógica do ambiente utilizado neste trabalho pode ser decomposta em três componentes principais. O primeiro é formado pelos nós que compõem a nuvem de infraestrutura como serviço (*IaaS*). Aqui, utilizamos a plataforma *OpenStack (release Juno)* com um nó rodando os serviços de controle e um nó rodando os serviços de virtualização (e.g. *nova-compute*), que é onde são alocadas as máquinas virtuais dos clientes da nuvem.

O segundo componente representa a arquitetura de monitoramento de níveis de acordo de serviço (*Security-SLA*) apresentada em [Ferreira 2013]. Nesta arquitetura encontra-se um elemento que atua como gerenciador dos componentes de monitoramento (*sla-manager*), um nó que oferece os serviços de gerenciamento por meio de mensagens SOAP (*sla-server*), os agentes de monitoramento em si, e uma base de dados onde são armazenadas informações sobre clientes, SLAs, métricas coletadas, entre outras.

O terceiro e último componente é formado pelas máquinas responsáveis por gerar tráfego em direção a uma VM, conforme detalhado na Seção 2.2, e executar os ataques contra ela. Uma *botnet* baseada no protocolo IRC foi desenvolvida para facilitar a sincronização da geração de tráfego e dos ataques a partir de um ponto central, e também facilitar a escalabilidade dos testes. Os nós pertencentes a este componente rodam uma aplicação cliente (*bot*) que se conecta a um servidor IRC, o qual atua como um servidor de comando e controle (C&C) da *botnet*. Os comandos para os *bots* devem ser passados por um usuário específico a partir de qualquer computador conectado a este servidor IRC.

 © 2017 Sociedade Brasileira de Computação

Dado o aspecto modular desta arquitetura, o IDS pode ser inserido em qualquer região, desde que tenha acesso ao banco de dados que armazena os dados coletados das VMs a serem analisadas. Por este motivo ele não foi atrelado a nenhum dos três componentes descritos anteriormente. Uma vantagem direta desta flexibilidade é que este IDS torna-se menos vulnerável a ataques externos, se comparado com aplicações que obrigatoriamente devem ser instaladas no sistema analisado—e eventualmente comprometido.

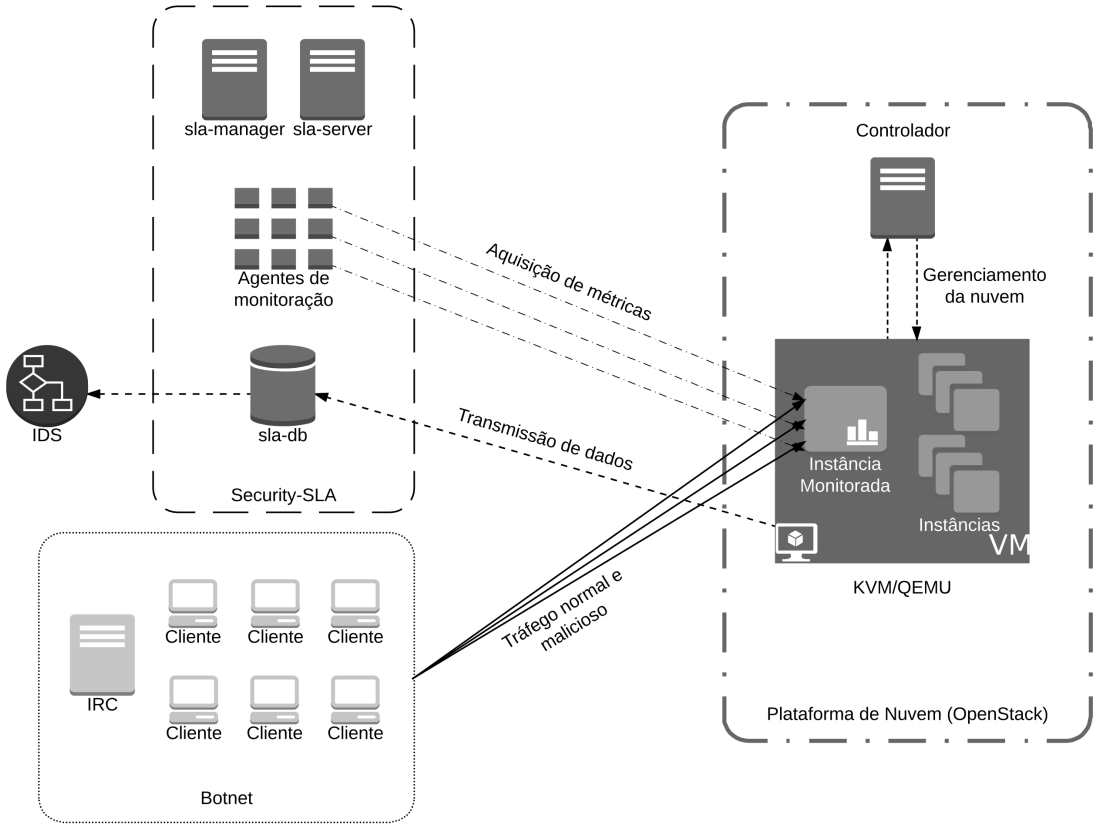


Figura 1. Arquitetura lógica do ambiente

2.2. Geração de Tráfego

A análise do IDS proposto em um cenário de estudo realístico é uma das principais preocupações deste trabalho. Mais especificamente, desejamos analisar o impacto de ataques contra um servidor *web* em termos de variação nas medidas de desempenho. Neste aspecto, é importante que a máquina virtual analisada apresente características realísticas. No caso de requisições HTTP, portanto, devemos nos preocupar com fatores como tamanho dos arquivos transmitidos, quantidade de arquivos transmitido por sessão, intervalo entre sessões etc.

Diversos estudos abordam o problema de geração de tráfego, como [Megyesi et al. 2015, Vishwanath and Vahdat 2009, Avallone et al. 2004, Mah 1997]. Em nossos testes, adotamos a ferramenta *SURGE* (abreviação para *Scalable URL Reference Generator*) [Barford and Crovella 1998] para gerar carga de trabalho sobre a máquina virtual alvo. Esta ferramenta é capaz de gerar tráfego HTTP/1.0 e HTTP/1.1

© 2017 Sociedade Brasileira de Computação

seguindo modelos estatísticos, levando o servidor *web* a operar como se estivesse recebendo requisições de usuários reais.

2.3. Agentes e Métricas de Desempenho

Todos as métricas utilizadas para monitorar a máquina alvo são coletadas por agentes externos à ela. Neste trabalho, apoiamo-nos em agentes baseados em técnicas de introspecção de máquina virtual (VMI), comunicação com *hypervisor* e sistema operacional (SO) do hospedeiro. Graças a essa abordagem, nenhum *software* precisa ser instalado nos sistemas convidados (*guest*), garantindo que o IDS possa continuar sua execução normal mesmo que um *guest* seja comprometido.

Estendendo o trabalho de [Ferreira 2013], os agentes foram refatorados para melhoria de desempenho, suportar outras plataformas e coletar novas métricas—rdbytes_persec, wrbytes_persec, tcp_reset_in_persec, tcp_reset_out_persec e tcp_currestab_persec. Os agentes baseados em VMI utilizam agora a versão 0.11 da biblioteca *libVMI* e são compatíveis com a plataforma de nuvem *OpenStack*. Apesar da implementação atual dos agentes CPU e IO suportar apenas os *hypervisors Xen* e *KVM/QEMU*, outros modelos podem ser facilmente adicionados à arquitetura. Na Tabela 1 estão relacionados os nomes dos agentes e a técnica de coleta de dados utilizada por cada um, e na Tabela 2 constam as métricas utilizadas neste trabalho.

Tabela 1. Método de coleta utilizado por cada agente

Agente	Método
CPU	Hypervisor
IO	Hypervisor
Memory	VMI
MIB	VMI
NetStatistics	SO
Netfilter	SO
Process	VMI

2.4. Sistema de Detecção de Intrusão

O conjunto das medidas de desempenho coletadas de uma VM ao longo do tempo pode ser visto como uma série temporal multivariada. Assim, utilizamos o algoritmo baseado em grafo para detecção de anomalias em séries temporais multivariadas proposto por [Cheng et al. 2008], descrito brevemente a seguir.

Seja $X = x_1, x_2, \ldots, x_T$ uma série temporal constituída de uma sequência de medidas para uma variável X nos intervalos de tempo $1, 2, \ldots, T$. Uma série temporal multivariada $\mathcal{D} = \{X_i\}_{i=1}^{p}$ é uma coleção de séries temporais correspondentes a medidas de p variáveis coletadas no mesmo intervalo de tempo [Cheng et al. 2008].

Uma matriz de similaridade—ou matriz *kernel*—K define o grau de similaridade entre cada par de observações de uma série temporal. A similaridade entre dois pontos i e j quaisquer é dada pela Equação 1—função de base radial (RBF) Gaussiana:

$$K(i,j) = \exp[-\frac{\sum_{k=1}^{p}(x_{ik} - x_{jk})^2}{2\sigma^2}], \quad (1)$$

 © 2017 Sociedade Brasileira de Computação

Tabela 2. Descrição das métricas de desempenho

Agente	Métrica	Descrição
CPU	utilization	Uso de CPU
IO	rdbytes_persec	Média de bytes lidos por segundo
	rdreq_persec	Média de requisições de leitura por segundo
	wrbytes_persec	Média de bytes escritos por segundo
	wrreq_persec	Média de requisições de escrita por segundo
Memory	pfmajor_psec	Média de *major page fault* por segundo
	pfminor_pesec	Média de *minor page fault* por segundo
	phys_perc_usage	Percentagem de memória utilizada
MIB	tcp_currestab_persec	Média de conexões TCP em estado ESTABLISHED ou CLOSE_WAIT por segundo
NetStatistics	traffic_in	Média de tráfego de entrada por segundo
	traffic_out	Média de tráfego de saída por segundo
Netfilter	tcp_reset_in_persec	Média de pacotes TCP de entrada com *flag* RST por segundo
	tcp_reset_out_persec	Média de pacotes TCP de saída com *flag* RST por segundo
Process	avg_wrksetsize	Média de *working set size*
	processes	Número total de processos
	threads	Número total de *threads*

onde σ é um parâmetro livre chamado largura de banda do *kernel*.

[Cheng et al. 2008] propõem tratamentos distintos dependendo se a série temporal multivariada possui uma variável alvo ou não. No nosso problema, todas as métricas possuem igual importância. Neste caso, o algoritmo considera cada uma das p variáveis como alvo em uma iteração e constrói duas matrizes *kernel*: uma utilizando apenas a variável p, chamada KY_i, e outra utilizando as $p-1$ variáveis restantes (preditores), chamada K_i. Uma vez construídas, é realizado um alinhamento de *kernel* para se obter uma matriz kernel ajustada $\widehat{K_i}$ que maximiza a correlação entre K_i e KY_i. A matriz *kernel* alinhada final $\widehat{K_\alpha}$ é obtida pelo produto Hadamard das matrizes individuais $\widehat{K_i}$. Esta matriz pode ser transformada em uma representação de grafo valorado, $\mathcal{G} = (V, E)$, onde V é o conjunto de nós—neste caso, cada observação de métricas em um instante de tempo—, e $E \subseteq V \times V$ é o conjunto de arestas.

A partir de $\widehat{K_\alpha}$, obtemos a matriz de transição S utilizando a Equação 2:

$$S(i,j) = \frac{\widehat{K_\alpha(i,j)}}{\sum_{j=1}^{n} \widehat{K_\alpha(i,j)}}, \tag{2}$$

onde i e j são os índices da linha e coluna, respectivamente, e n é a ordem da matriz $\widehat{K_\alpha}$. Cada (i,j)-ésimo elemento desta matriz representa a probabilidade de transição do nó i para o nó j.

Por fim, um passeio aleatório é executado sobre o grafo visando encontrar o grau de conectividade c de cada nó. O valor para cada nó é calculado aplicando-se a Equação 3

iterativamente até que alguma condição de parada seja atingida. Nossa implementação utiliza como critério de parada um valor de tolerância $\epsilon = 10^{-4}$ e um número máximo de iterações $m = 1000$.

$$c = \frac{d}{n} + (1 - d)Sc, \tag{3}$$

onde d é o fator de amortecimento. Em cada iteração do passeio aleatório, existe uma probabilidade d de se permanecer no nó atual, contra $1 - d$ de se visitar algum nó seguindo a matriz de transição S. A ideia principal por trás deste algoritmo, conforme descrito em [Cheng et al. 2008], é que nós anômalos serão altamente dissimilares em relação aos demais. Esses nós, portanto, serão pouco visitados durante a execução do passeio aleatório e, consequentemente, terão um baixo valor de conectividade. No contexto deste trabalho, assumimos que as perturbações causadas em uma máquina virtual por conta de atividades maliciosas resultem em nós com baixa conectividade, ou seja, anomalias.

3. Experimentos

Nesta Seção, apresentamos detalhes sobre os testes realizados, dividido por etapas.

3.1. Máquina Virtual

Criamos uma VM na nuvem privada rodando o sistema operacional *Linux Ubuntu* 10.04 LTS, com 20GB de disco, 2GB de RAM, 1GB de *swap* e 2 vCPU. Esta instância possui 2 endereços IP: um interno, visível apenas dentro da nuvem, e um externo, visível na rede local—a mesma do sistema hospedeiro e diversas outras máquinas do laboratório. Esta máquina roda os serviços *Apache* v2.2.14, *Exim* v4.77 e *MySQL Server* v5.1.73.

3.2. Geração de Tráfego

Conforme descrito na Seção 2.2, utilizamos a ferramenta *SURGE* para gerar tráfego HTTP em direção à VM analisada. Além de se valer de métodos estatísticos para controlar os tempos e as quantidades das requisições, esta ferramenta gera arquivos texto com tamanhos variáveis, também seguindo um modelo estatístico configurável, para serem hospedados no servidor *web* sob teste. A Tabela 3 mostra alguma das propriedades tratadas pela ferramenta.

Tabela 3. Algumas propriedades do tráfego HTTP simulado. A coluna *Valor* mostra um valor pré-determinado ou a distribuição de probabilidade utilizada

Propriedade	Valor
Total de arquivos	2000
Total de requisições para o arquivo mais popular	20000
Tamanho dos arquivos	Pareto e Log-Normal
Número de arquivos por objeto	Pareto
Tempo entre requisições	Pareto

Três classificações de tráfego foram consideradas neste trabalho: i) tráfego normal; ii) tráfego de sobrecarga; e iii) tráfego de ataque. O tráfego normal foi gerado utilizando-se duas máquinas. O programa *SURGE* foi executado em cada uma utilizando 2 processos, cada um com 25 *threads*, durante 5000 segundos. Este foi o tempo que encontramos como necessário para que todas, ou quase todas, as requisições estipuladas pela

 © 2017 Sociedade Brasileira de Computação

ferramenta pudessem ser realizadas. Os dois clientes mantiveram-se executando o programa durante toda a duração do teste, ou seja, a VM sob estudo sempre esteve recebendo requisições HTTP. Uma terceira máquina foi utilizada para gerar tráfego de sobrecarga durante 1 hora, representando uma situação onde o servidor *web* fosse mais acessado. A mesma configuração do *SURGE* apresentada anteriormente foi utilizada neste caso. O tráfego de ataque é descrito na Seção 3.3. As configurações dos sistemas utilizados para geração de tráfego estão descritas na Tabela 4. O tráfego normal foi gerado pelos clientes #1 e #2, enquanto o tráfego de sobrecarga foi gerado pelo cliente #3.

Tabela 4. Configuração dos sistemas utilizados na geração de tráfego

Cliente	Número de processadores	RAM (GB)	Sistema Operacional
#1	2	2	Linux Ubuntu 14.04 64-bit
#2	4	4	Linux Ubuntu 14.04 64-bit
#3	6	4	Linux Ubuntu 14.04 64-bit

3.3. Ataques

Nesta etapa, executamos ataques de *port scan* e de negação de serviço (*DDoS*) contra a VM hospedada na nuvem, ao longo de um dia. Primeiro, simulamos um atacante realizando uma varredura em 65535 portas TCP. Para verificar a variação dos distúrbios causados na VM e a sensibilidade do IDS, executamos tanto uma abordagem agressiva (alta quantidade de portas por segundo) quanto uma mais segura (poucas portas por segundo). Em relação aos ataques de *DDoS*, utilizamos as máquinas relacionadas na Tabela 4 para executar ataques na camada de aplicação, com auxílio das ferramentas *hulk* e *GoldenEye*. A Tabela 5 contém os horários aproximados de início e fim de cada ataque.

Tabela 5. Tempos de início e fim de cada ataque

Ataque	Variação	Início	Fim
Port Scan	Rápido	01:15:40	01:16:30
	Lento	02:41:09	02:46:00
DDoS	Camada de Aplicação	17:24:30	17:50:30

3.4. Pré-processamento dos Dados

Visto que nosso sistema utiliza métricas com diferentes unidades e magnitudes, é necessário realizar um pré-processamento dos dados antes de eles serem enviados ao IDS. Nosso objetivo com isso é impedir que uma métrica com grande intervalo de valores afete as medidas de distâncias das observações. Assim, para cada variável, subtraímos a média e dividimos pelo desvio padrão, padronizando-as. Após a conclusão desta etapa, todas as variáveis possuem média igual a 0 e variância igual a 1.

3.5. Sistema de Detecção de Intrusão

Neste trabalho, investigamos o impacto da escolha de dois parâmetros ligados ao IDS: i) tamanho de subsequência (δ); e ii) largura de banda do *kernel* (σ). O parâmetro δ representa a quantidade de observações do conjunto de dados que são agrupadas em

© 2017 Sociedade Brasileira de Computação

um subconjunto. Visto de outra forma, seu objetivo é permitir que se possa comparar subsequências de valores ao invés de observações atômicas entre si. Com isso, o algoritmo pode identificar um sequência anômala em uma série temporal multivariada. Detalhes sobre o cálculo da matriz *kernel* para subsequências podem ser encontrados em [Cheng et al. 2008]. O segundo parâmetro tem influência na separação dos dados e sua escolha e técnicas de otimização ainda são objetos de estudo [Benoudjit et al. 2002, Wu and Wang 2009, Li et al. 2014]. Para verificar a consequência da escolha dessas propriedades, executamos o algoritmo repetidas vezes, combinando os valores de $\delta = 1, 2, 6, 10$ e $\sigma = 2^{-15}, 2^{-13}, 2^{-11}, \ldots, 2^{3}$.

4. Análise

Iniciamos nossa análise inspecionando as características das métricas e as relações entre elas. A Figura 2 representa a matriz de dispersão gerada a partir da análise de 5 métricas utilizadas durante os testes. Algumas variáveis apresentam uma relação clara entre si, como *traffic_in* e *traffic_out*, o que era esperado uma vez que o tráfego gerado nos testes é do tipo requisição-resposta (HTTP). Um ponto importante de se confirmar essa relação é que validamos a corretude dos agentes de coleta.

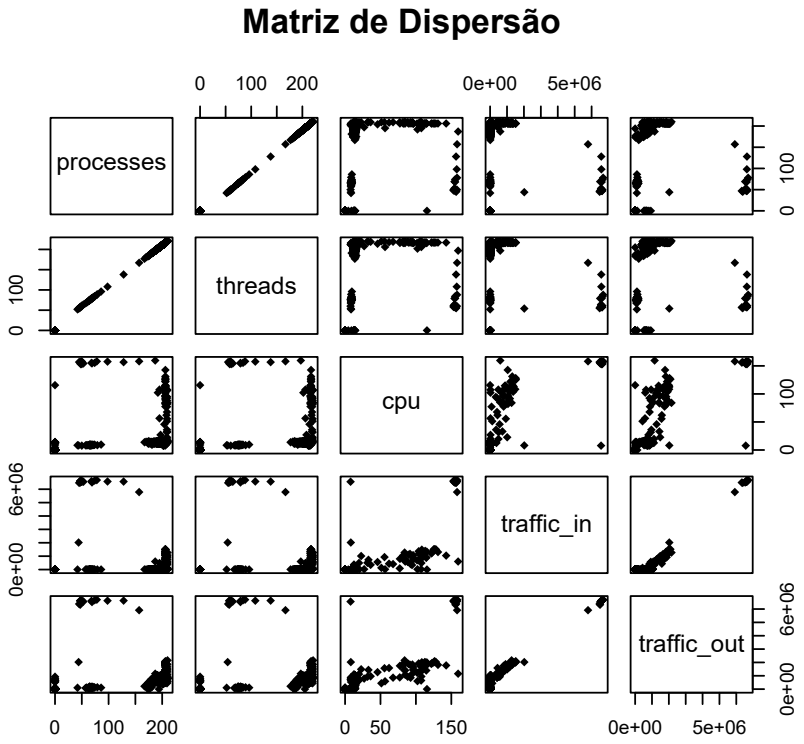


Figura 2. Matriz de dispersão das métricas *processes*, *threads*, *cpu*, *traffic_in* e *traffic_out*

A seguir, analisamos o efeito de cada ataque no desempenho da máquina virtual. Em todos os casos, analisamos intervalos de 1 hora. Após o cálculo de conectividade e padronização dos resultados, consideramos nós anômalos aqueles com valor menor ou igual a -3 (*threshold*). Este valor de *threshold* foi definido empiricamente e mostrou-se

 © 2017 Sociedade Brasileira de Computação

satisfatório para os experimentos realizados. Métodos para a escolha do melhor valor e a comparação deste com outros cenários de teste são objetos de trabalhos futuros.

4.1. Identificação de Port Scan

Na Figura 3 mostramos o perfil das métricas utilizadas ao longo do período analisado. A análise dessas métricas pelo algoritmo implementado resulta em um vetor contendo o *score* (classificação) padronizado de anomalia para cada observação ou sequência da série temporal multivariada. Esses valores indicam o quanto uma observação ou sequência está distante da média em termos de desvio padrão. Na Figura 4 mostramos o *score* resultante da execução do IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$. O resultado indica que o *port scan* agressivo gerou distúrbios significativos na VM, identificado no instante $i = 01 : 16 : 10$. Acreditamos que a outra anomalia encontrada na análise seja resultado da inicialização dos serviços da máquina virtual, pois coincide com o tempo de *boot*. Utilizando os mesmos parâmetros, também foi possível identificar o ataque de *stealth port scan* (variação mais lenta), identificado na Figura 5 no instante $i = 02 : 42 : 40$. Analisando o gráfico de perfil deste caso, identificamos distúrbios relevantes nos instantes $i = 02 : 00 : 10$ e $i = 02 : 20 : 10$, porém não identificamos a causa.

4.2. Identificação de DDoS

Nesta etapa, repetimos o procedimento descrito na Seção 4.1 no intuito de identificar ataques de *DDoS* na camada de aplicação. Os testes para $\delta = 6$ e $\delta = 10$ não encontraram nenhuma anomalia com o *threshold* de -3 utilizado anteriormente. Para $\delta = 2$, o algoritmo encontrou uma anomalia, porém não no instante referente ao ataque. Melhores resultados foram obtidos novamente com os parâmetros $\delta = 1$ e $\sigma = 2^{-5}$. Para esta configuração, o IDS identificou corretamente os distúrbios causados pelo ataque de negação de serviço e também outras duas anomalias nos instantes $i = 16 : 55 : 43$ e $i = 17 : 04 : 13$. O *score* de anomalia está apresentados na Figura 7. Para $\sigma = 2^{-15}, 2^{-13}, \ldots, 2^{-7}$ o resultado é semelhante, apenas a anomalia no instante $i = 16 : 55 : 43$ não é encontrada.

4.3. Discussão

A análise dos resultados mostra que a escolha dos parâmetros *tamanho de subsequência* e *largura de banda do kernel* tem grande impacto sobre a detecção de anomalias. Para evitar que regiões adjacentes em um intervalo de anomalia apresentem-se como anomalias isoladas, após o cálculo do *score* de anomalia realizamos uma agregação de nós anômalos. Isto é feito agrupando-se nós referentes a instantes de tempo consecutivos em uma única notificação de anomalia, enfatizando o início e o término da sequência anômala.

5. Trabalhos Relacionados

Em [Avritzer et al. 2010], os autores descrevem um *framework* para geração de assinaturas de desempenho de máquinas *Windows* reais e um algoritmo para detectar ataques de segurança automaticamente, considerando alguns testes comuns de segurança como *Buffer Overflow* e ataques de negação de serviço. As assinaturas são definidas analisando-se dados de sistema gerados pelo *Microsoft Windows Management Instrumentation API* (WMI)—uso de CPU, memória, rede e entrada/saída (I/O). Cada experimento (teste de ataque) é conduzido em duas etapas: uma etapa de uso regular do sistema para coletar o perfil de desempenho base, e uma etapa de teste de segurança para coletar o perfil do

© 2017 Sociedade Brasileira de Computação

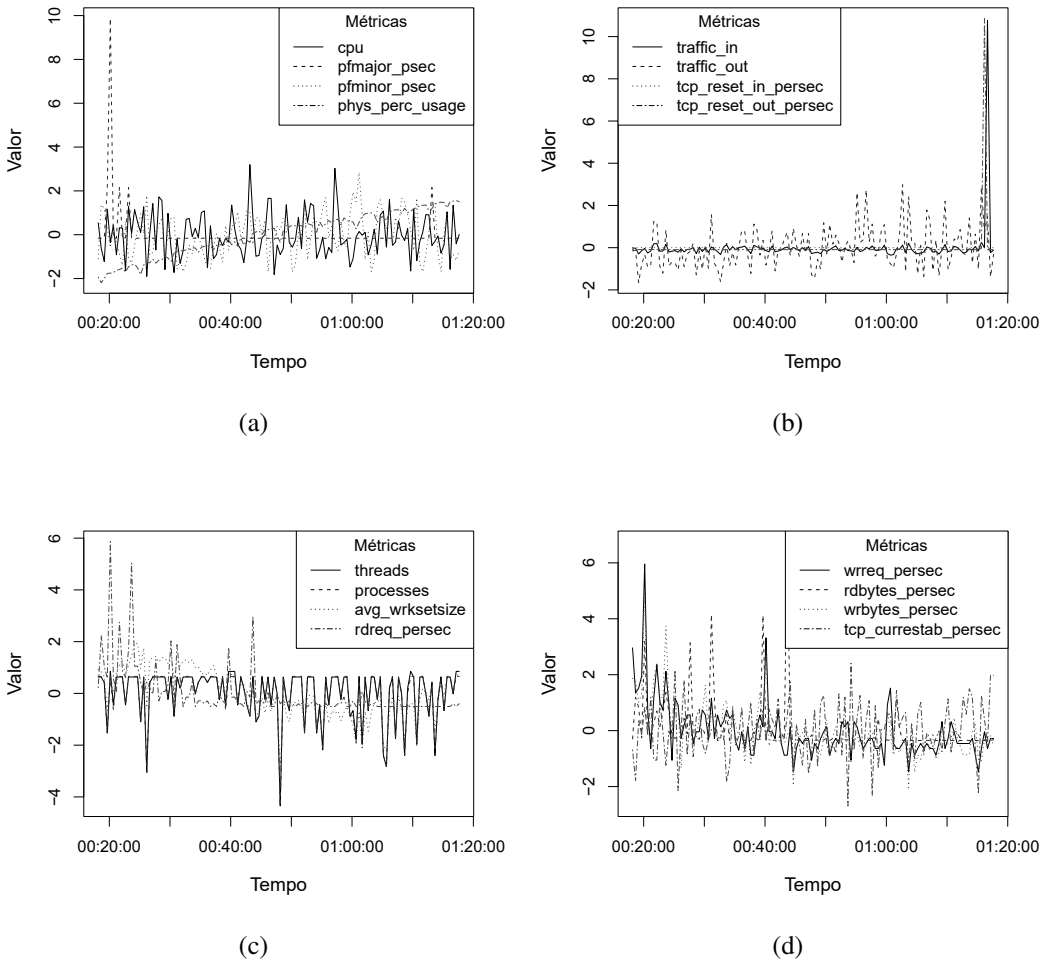


Figura 3. Perfil das métricas de desempenho durante um ataque de *port scan* agressivo

ataque. O principal problema desta abordagem é a incapacidade de se detectar novos ataques. Além disso, qualquer evento que resulte em uma alteração do perfil base impacta negativamente na capacidade do sistema identificar as ameaças mapeadas anteriormente. Neste trabalho, utilizamos uma abordagem que elimina a necessidade de se definir um perfil base *a priori* e que é capaz de detectar ataques desconhecidos. Uma extensão deste trabalho deve permitir que a anomalia identificada seja classificada em tipos específicos de ataques, aumentando a qualidade da informação apresentada para os clientes da nuvem.

Seguindo a metodologia de [Avritzer et al. 2010], [Ferreira 2013] apresenta um sistema de monitoramento de segurança baseado em acordos de níveis de serviço para nuvens de infraestrutura. No entanto, ao invés de dados coletados a partir de máquinas *Windows* reais, o autor aplica a técnica no monitoramento de máquinas virtuais rodando o sistema operacional *Linux* em um ambiente de nuvem no modelo IaaS. As métricas utilizadas para geração e análise de assinaturas são provenientes de introspecção de máquina virtual e dados coletados a partir do monitor de máquinas virtuais, ao passo que a análise das assinaturas é feita utilizando-se técnicas de aprendizado de máquina. Embora os re-

 © 2017 Sociedade Brasileira de Computação

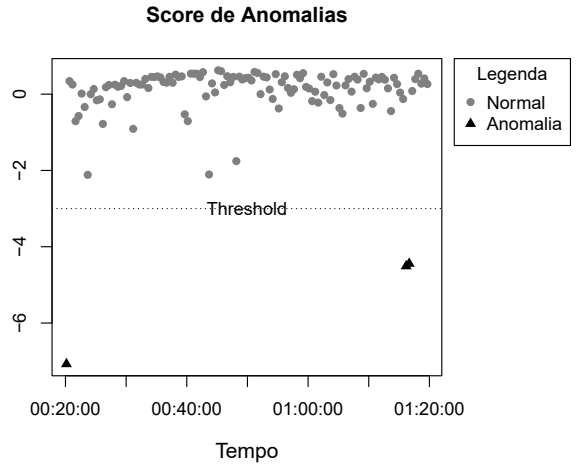


Figura 4. Score de anomalias gerado pelo IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$

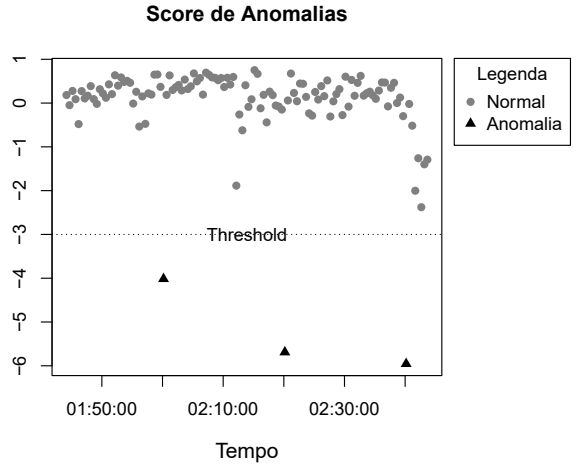


Figura 5. Score de anomalias gerado pelo IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$

sultados mostrem a factibilidade de se utilizar métricas de desempenho para detecção de ataques em um ambiente de nuvem, esta abordagem também requer que um perfil base seja definido *a priori*. As propriedades do tráfego de rede também não são levadas em consideração, visto que o perfil base é definido por uma carga constante na VM analisada, sem sinais de variância no intervalo entre sessões ou na quantidade de dados transferidos. Este ponto é tratado neste trabalho por meio da ferramenta *SURGE*, descrita na Seção 2.2. Os resultados da análise sobre um tráfego com mais variações reforça a expectativa de que a análise de métricas de desempenho para problemas de detecção de intrusão possa ser empregada em ambientes reais.

[Nikolai and Wang 2014] utilizam métricas coletadas a partir do *hypervisor* para identificar ataques de negação de serviço em máquinas virtuais rodando um servidor *web*. Embora os autores mostrem que é possível detectar ataques utilizando apenas 5 métricas oferecidas pelo *hypervisor* (pacotes transmitidos, pacotes recebidos, requisições de leitura e de escrita em disco, e uso de CPU), apenas uma variação do valor das métricas

© 2017 Sociedade Brasileira de Computação

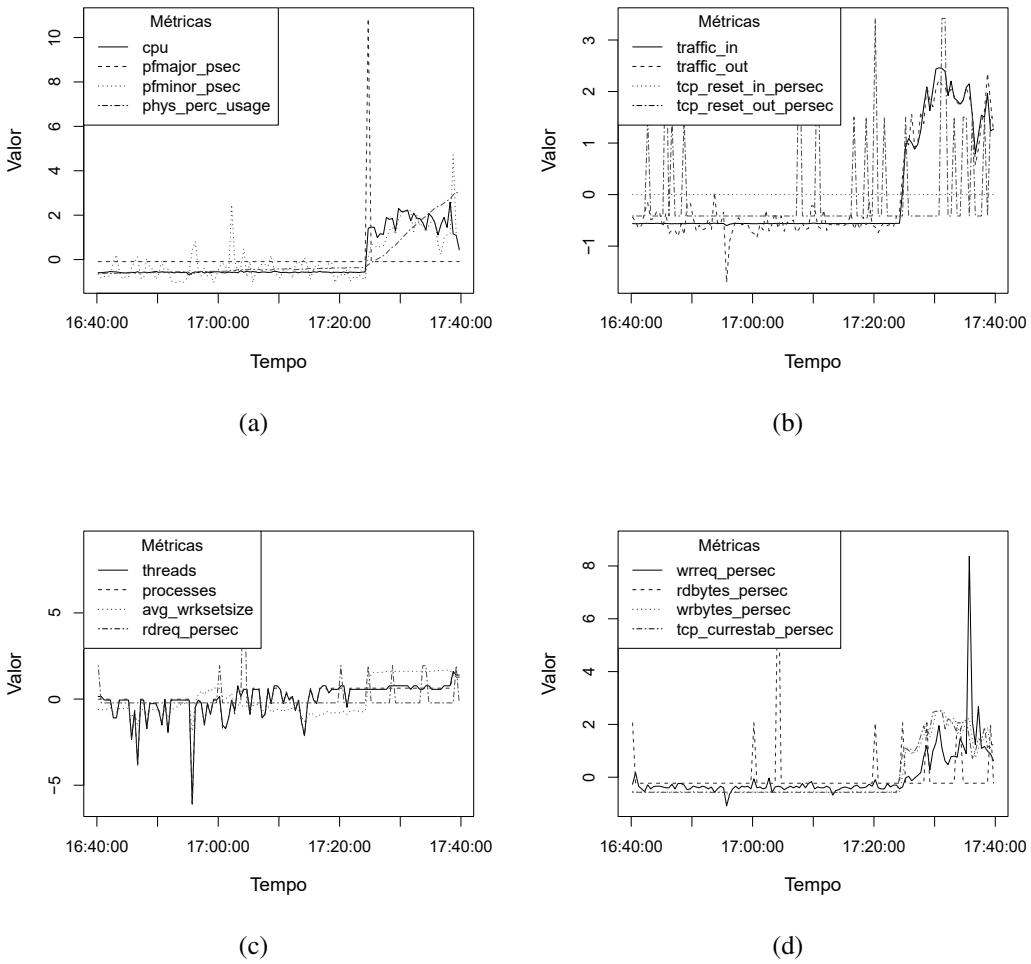


Figura 6. Perfil das métricas de desempenho durante um ataque de *DDoS* na camada de aplicação

em relação à média é utilizada para detectar intrusão baseada em assinaturas geradas manualmente. Além disso, a carga aplicada sobre a máquina testada também não considera aspectos realísticos de um servidor *web*, como intervalo entre sessões e número de requisições por sessão. Em nosso trabalho, utilizamos um total de 16 métricas de desempenho, e o algoritmo empregado determina a relação das métricas entre si para identificar possíveis intervalos de anomalia.

Detalhes sobre outras abordagens para sistemas de detecção de intrusão em nuvens computacionais são apresentadas em [Modi et al. 2013] e [Patel et al. 2013], notando-se que o último enfatiza a necessidade de se desenvolver sistemas de detecção e prevenção de intrusão projetados especificamente para nuvens, ao invés do uso de ferramentas tradicionais.

6. Conclusão e Trabalhos Futuros

Neste trabalho, apresentamos um sistema de detecção de intrusão baseado em anomalias para identificação de ataques contra máquinas virtuais hospedadas em uma nuvem com-

 © 2017 Sociedade Brasileira de Computação

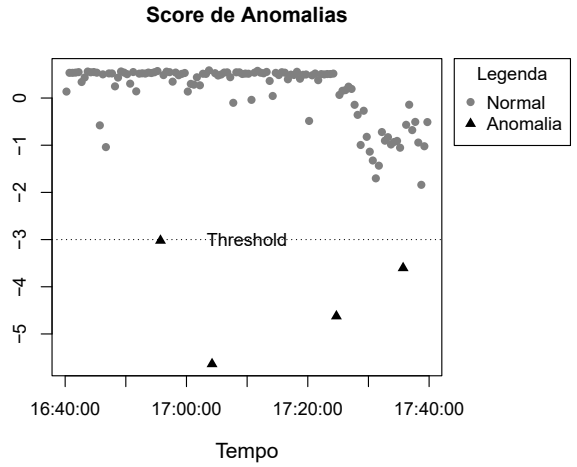


Figura 7. Score de anomalias gerado pelo IDS com parâmetros $\delta = 1$ e $\sigma = 2^{-5}$

putacional. Mostramos que o método baseado em passeio aleatório apresenta resultados satisfatórios quando escolhidos os parâmetros corretos. Ainda, tratando as métricas de desempenho coletadas como uma série temporal multivariada, nosso sistema mostrou-se capaz de detectar ataques de *port scan* e *DDoS*. Por se tratar de um método baseado em anomalias, a arquitetura é capaz de detectar ataques desconhecidos e outros distúrbios pontuais.

Embora seja possível realizar a detecção de anomalias comparando-se subsequências da série temporal, experimentos mostraram que esta abordagem não apresenta resultados satisfatórios. Um dos motivos é que, ao contrário do estudo feito por [Cheng et al. 2008], os dados de desempenho de VMs não apresentam ciclos bem definidos, além de serem altamente dependentes do fluxo de dados e *softwares* instalados.

Como trabalho futuro, pretendemos aplicar este método de detecção para avaliar uma VM rodando serviços reais, disponíveis publicamente na *Internet*, e integra-lo a um classificador baseado em assinaturas, visando mostrar para os clientes com SLA definidos não só os distúrbios mas também o tipo de ataque que cada instância sofreu. Finalmente, outros tipos de ataques e sistemas operacionais do *guest* precisam ser estudados para verificar a abrangência do método e sua robustez a ruídos, além de permitir uma análise em tempo real.

Referências

Avallone, S., Guadagno, S., Emma, D., Pescape, A., and Ventre, G. (2004). D-ITG distributed internet traffic generator. In *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*, pages 316–317.

Avritzer, A., Tanikella, R., James, K., Cole, R. G., and Weyuker, E. (2010). Monitoring for security intrusion using performance signatures. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, WOSP/SIPEW '10, pages 93–104, New York, NY, USA. ACM.

Barford, P. and Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGME-*

TRICS Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '98/PERFORMANCE '98, pages 151–160, New York, NY, USA. ACM.

Benoudjit, N., Archambeau, C., Lendasse, A., Lee, J. A., Verleysen, M., et al. (2002). Width optimization of the gaussian kernels in radial basis function networks. In *ESANN*, volume 2, pages 425–432.

Cheng, H., Tan, P. N., Potter, C., and Klooster, S. (2008). A robust graph-based algorithm for detection and characterization of anomalies in noisy multivariate time series. In *2008 IEEE International Conference on Data Mining Workshops*, pages 349–358.

Ferreira, A. S. (2013). Uma arquitetura para monitoramento de segurança baseada em acordos de níveis de serviço para nuvens de infraestrutura. Master's thesis, Universidade Estadual de Campinas, Instituto de Computação, Campinas, SP, Brasil.

Li, J.-B., Wang, Y.-H., Chu, S.-C., and Roddick, J. F. (2014). Kernel self-optimization learning for kernel-based feature extraction and recognition. *Information Sciences*, 257:70–80.

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., and Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24.

Mah, B. A. (1997). An empirical model of http network traffic. In *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 2, pages 592–600.

Megyesi, P., Szabó, G., and Molnár, S. (2015). User behavior based traffic emulator: A framework for generating test data for dpi tools. *Computer Networks*, 92:41–54.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., and Rajarajan, M. (2013). A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57.

Nikolai, J. and Wang, Y. (2014). Hypervisor-based cloud intrusion detection system. In *Computing, Networking and Communications (ICNC), 2014 International Conference on*, pages 989–993.

Patel, A., Taghavi, M., Bakhtiyari, K., and Júnior, J. C. (2013). An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of Network and Computer Applications*, 36(1):25–41.

Vishwanath, K. V. and Vahdat, A. (2009). Swing: Realistic and responsive network traffic generation. *IEEE/ACM Trans. Netw.*, 17(3):712–725.

Wu, K.-P. and Wang, S.-D. (2009). Choosing the kernel parameters for support vector machines by the inter-cluster distance in the feature space. *Pattern Recognition*, 42(5):710–717.

Zhu, Y. and Zheng, Y. (2008). Research on intrusion detection system based on pattern recognition. In *Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on*, volume 1, pages 609–612.

 © 2017 Sociedade Brasileira de Computação

Esquema de Acordo de Chaves de Conferência Baseado em um Problema de Funções Quadráticas de Duas Variáveis

Luis Antonio B. Kowada[1], **Raphael C. S. Machado**[2,3]

[1]Instituto de Computação – Universidade Federal Fluminense
Niterói – RJ – Brazil

[2]Inmetro, Instituto Nacional de Metrologia, Qualidade e Tecnologia
Rio de Janeiro – RJ – Brazil.

[3]Programa de Pós-Graduação em Ciência da Computação – CEFET/RJ
Rio de Janeiro – RJ – Brazil.

luis@ic.uff.br, rcmachado@inmetro.gov.br

Abstract. *Conference key establishment is the process to determine a common shared key between three or more participants. If all participants influence key generated by this process, we have a* conference key agreement scheme. *In the present work, we propose a conference key agreement scheme with some significant advantages over the existing schemes, of which we highlight the following:*

1. *agreement of conference keys in a single round and with linear quantity of messages published;*
2. *combination of two conference keys in a single round and with constant number of published messages;*
3. *renewing conference keys in a single round and with only one published message.*

The proposed scheme is resilient against classical attack scenarios, but is vulnerable to attack by an attacker capable of calculating the discrete logarithm and is therefore vulnerable to attacks with a quantum computer.

Resumo. *Estabelecimento de chave de conferência é o processo de determinação de uma chave comum compartilhada entre três ou mais participantes. Se neste processo, todos os participantes influenciam a chave a ser gerada, temos um* esquema de acordo de chaves de conferência. No presente trabalho, propomos um esquema de acordo de chaves de conferência que apresenta vantagens significativas sobre os atuais esquemas existentes, como:

1. *acordo de chaves de conferência em uma única rodada e com quantidade linear de mensagens publicadas;*
2. *combinação de duas chaves de conferência em uma única rodada e com número constante de mensagens publicadas;*
3. *renovação de chaves de conferência em uma única rodada e com única mensagem publicada.*

O esquema proposto é resiliente face a cenários de ataques clássicos, mas é vulnerável a ataques executados por um atacante capaz de calcular o logaritmo discreto, sendo, portanto, vulnerável a ataques usando Computação Quântica.

 © 2017 Sociedade Brasileira de Computação

1. Introdução

Esquemas de estabelecimento de chaves são uma primitiva fundamental em criptografia, pois permitem que um conjunto de participantes estabeleça uma chave criptográfica compartilhada [Barker et al. 2013] que poderá ser posteriormente utilizada, por exemplo, para fins de confidencialidade — com o uso de uma cifra — ou de autenticação — por exemplo, com o uso de um código de autenticação de mensagens. Esquemas de estabelecimento de chaves estão na origem dos modernos métodos de Criptografia: o primeiro método de criptografia assimétrica é exatamente um esquema de estabelecimento de chaves, a saber, o método de Diffie e Hellman [Diffie and Hellman 1976], desenvolvido em 1976, e que baseia-se na suposta dificuldade de se calcular o logaritmo discreto para permitir que dois (ou mais) participantes estabeleçam um segredo comum a partir de informações trocadas de maneira pública entre eles.

Desde que o método de Diffie e Hellman foi proposto, diversas variantes do método têm sido propostas, tais como o método ElGammal [ElGammal b], os protocolos MTI [Matsumoto et al.] e o protocolo STS [ElGammal a, Diffie et al. 1992]. O estabelecimento de uma chave compartilhada entre três ou mais participantes, denominado *conference keying* ou *chave de conferência* [Lee et al. 2004], tem ganhado cada vez mais atenção à medida em que comunicações em grupo ganham importância na Internet, seja devido aos protocolos multicast, seja devido à proliferação de aplicativos de mensagem instantânea envolvendo grupos de usuários. Observam-se diversos estudos a respeito de como estender o método de Diffie e Hellman para cenários multi-participantes (por exemplo, [Ateniese et al. 2000, Chunsheng 2015a, Chunsheng 2015b] ou a respeito de novas construções que suportem "nativamente" o estabelecimento de chaves de conferência em uma única rodada [Joux 2000, Boneh and Silverberg 2002].

No presente trabalho, propomos um esquema de acordo de chaves análogo ao método de Diffie e Hellman, porém, baseado na dificuldade de se encontrar as raízes inteiras de uma equação Diofantina com duas variáveis e com grau máximo 2. O esquema proposto apresenta vantagens estabelecimento de chaves de conferência, as quais enumeramos algumas a seguir:

1. Acordo de chaves de conferência em única rodada. Trata-se da principal característica do esquema proposto. Conforme discutimos na seção seguinte, desde o trabalho de Joux [Joux 2000], que construiu um esquema de acordo de chaves em rodada única para três participantes, a comunidade de criptografia busca por esquemas de acordo de chaves de conferência cujo número de rodada seja independente do número de participantes.
2. Combinação eficiente de chaves de conferência. O esquema proposto é eficiente no processo de criação de múltiplas chaves de conferência para múltiplos conjuntos de usuários, permitindo combinar a chave de dois grupos C_1 e C_2 de usuários para criar uma chave para $C_1 \cup C_2$, sem comprometer o sigilo das informações previamente encriptadas por meio das chaves originais.
3. Renovação automática de chaves de conferência. O esquema proposto permite a modificação dinâmica das chaves de conferência com base na publicação de uma única mensagem (em modo broadcast) — ou, mesmo, sem que haja necessidade de transmissão de mensagens, assumindo que os participantes estão sincronizados.

As vantagens acima enumeradas podem ser alcançadas porque, ao contrário do

 © 2017 Sociedade Brasileira de Computação

método de Diffie e Hellman, o esquema proposto não necessita que "chaves temporárias" sejam trafegadas ao longo do processo de acordo de chave de conferência. Ao contrário, cada participante precisa divulgar apenas uma única informação, a qual é utilizada para construir chaves de conferência, mas que não constituem chaves, por si só. Essa propriedade permite a imediata definição de chaves de conferência para qualquer subconjunto de participantes, bastando que cada participante deste subconjunto realize uma única computação sobre as informações divulgadas pelos participantes do subconjunto. Esses cenários são melhor explicados ao longo do texto.

O presente documento está organizado da seguinte forma. Na Seção 2 apresentamos os conceitos e terminologia necessários para a compreensão do trabalho. A Seção 3 descreve a nossa proposta de esquema de acordo de chaves, enquanto a Seção 4 descreve os aspectos de segurança do esquema proposto. Na Seção 5, estudamos a questão de como estabelecer chaves de conferência e discutimos as vantagens do esquema proposto sobre os esquemas baseados no método de Diffie e Hellman. A Seção 6 contém nossas considerações finais e aponta para os próximos desenvolvimentos a serem realizados sobre o esquema de acordo de chaves proposto.

2. Referencial Teórico

2.1. Esquemas de Acordo de Chaves

Esquemas de estabelecimento de chaves são algoritmos que permitem que uma chave criptográfica seja estabelecida entre um grupo de participantes [Barker et al. 2013]. Esquemas de estabelecimento de chaves classificam-se em *esquemas de transporte de chaves*, em que ocorre o envio de uma chave criptográfica entre os participantes, e *esquemas de acordo de chaves*, em que a chave criptográfica não é enviada entre os participantes.

O presente trabalho propõe um esquema de estabelecimento de chaves que se classifica como um esquema de acordo de chaves. Em um esquema de acordo de chaves, os participantes trocam informações a partir das quais os participantes são capazes de computar uma chave criptográfica secreta comum. Um esquema de acordo de chaves é, portanto, um protocolo no qual dois ou mais participantes podem acordar uma chave de tal maneira que ambos possam influenciar na chave final construída. Ao contrário de outros protocolos nos quais uma das partes é responsável por gerar e transmitir a chave, esquemas de acordo de chaves eliminam os riscos de que estão associados à capacidade de uma das partes "forçar" uma chave [Diffie et al. 1992].

O clássico método de Diffie e Hellman [Diffie and Hellman 1976] é um esquema de acordo de chaves baseado na dificuldade de se calcular o logaritmo discreto.

2.2. Método de Diffie-Hellman

O método de Diffie e Hellman é uma referência básica para esquemas de acordo de chave, frente ao qual o esquema proposto será comparado. Por uma questão de completude, apresentaremos o funcionamento básico do método.

Como explicamos anteriormente, o método de Diffie e Hellman permite que dois participantes gerem uma chave secreta comum por meio da qual eles poderão trocar informações através de um canal inseguro. Sejam Alice e Bob tais participantes. Primeiro, eles acordam acerca de dois números primos g e p, onde p é grande e g é uma

© 2017 Sociedade Brasileira de Computação

raiz primitiva módulo p. Os números g e p não precisam ser mantidos em segredo de outros usuários. Agora Alice escolhe um grande número aleatório a e Bob também escolhe um grande número b. Alice então calcula $A = g^a \bmod p$, que ela envia para Bob, e Bob calcula $B = g^b \bmod p$, o qual Bob transmite para Alice. Finalmente, Alice e Bob calculam sua chave compartilhada $K = g^{(ab)} \bmod p$, que Alice calcula como $K = B^a \bmod p = (g^b)^a \bmod p$ e Bob calcula como $K = A^b \bmod p = (g^a)^b \bmod p$. Onde $x = y \bmod z$, significa que x é igual ao resto da divisão inteira de y por z.

Alice e Bob agora podem usar sua chave compartilhada K para trocar informações cifradas. Para que um potencial espião obtivesse a chave K, Eva precisaria obter $K = g^{(ab)} \bmod p$ conhecendo apenas g, p, $A = g^a \bmod p$ e $B = g^b \bmod p$, o que demandaria calcular a a partir de A, g e p (resp. b a partir de B, g e p), ou seja, calcular o logaritmo discreto, o que é computacionalmente inviável para grandes valores de p.

Veremos, a seguir, que, embora engenhoso, o método de Diffie e Hellman não estende-se de maneira simples ao cenário em que se deseja estabelecer chaves de conferência — ou seja, chaves compartilhadas entre membros de um conjunto com três ou mais participantes.

2.3. Chave de Conferência

Acordo de chaves de conferência refere-se à capacidade de se estabelecer uma chave criptográfica compartilhada entre três ou mais participantes de uma comunicação.

O clássico método de Diffie e Hellman permite que se estabeleça chaves de conferência. Porém, por uma característica construtiva, para se estabelecer uma chave compartilhada entre os membros de um grupo com n participantes, é necessário que se transmita entre tais n participantes informação que poderia ser utilizada como chave criptográfica de $n' < n$ participantes. Por este motivo, o método de Diffie e Hellman acaba não sendo eficiente quando de deseja estabelecer múltiplas chaves criptográficas em diversos subgrupos de um grupo de n pessoas.

Considere o caso em que se deseje utilizar o método de Diffie e Hellman para estabelecer uma chave de conferência compartilhada por três participantes, A, B e C. Basicamente, para que usuário C possa calcular chave de conferência G^{abc}, é necessário que A envie g^a a B, que calcula g^{ab} e envia a C — que só então será capaz de calcular g^{abc}. O processo deverá ser repetido para que A e B sejam capazes de calcular g^{abc}, cada um. O processo acima apresenta três problemas, os quais descrevemos a seguir. O primeiro problema é o fato de que as chaves "parciais" são transmitidas para que se possa calcular a chave final. Desta forma, a chave compartilhada apenas entre A e B — a saber, g^{ab}, não pode ser utilizada para transmitir informações confidenciais entre A e B, pois ela também é conhecida por C. O segundo problema é a necessidade de transmissão de uma grande quantidade de mensagens, já que cada participante somente é capaz de calcular a chave final se conhecer a chave parcial construída por todos os outros participantes. O terceiro participante é a impossibilidade de se realizar uma rápida modificação dinâmica da chave de conferência — o que somente pode ser realizada por meio de uma completa reexecução do protocolo.

No caso geral, considere uso do método de Diffie e Hellman para estabelecer uma chave de conferência compartilhada por participantes de um grupo de tamanho n. Neste caso, desejar-se-á que cada participante chegue à chave $k_{s_1,\ldots,s_n} := g^{s_1 \cdot \cdots \cdot s_n}$. Para que

 © 2017 Sociedade Brasileira de Computação

o participante i seja capaz de construir $k_{s_1,...,s_n}$ tendo conhecendo apenas à sua própria informação privada s_n, é necessário que ele tenha acesso à chave "parcial" $k_{s_1,...,s_{n-1}} := g^{s_1 \cdot ... \cdot s_{n-1}}$. De maneira recursiva, a construção de $k_{s_1,...,s_{n-1}}$ demanda a "intervenção" de cada um dos participantes $s_1, ..., s_{n-1}$, o que leva a um número de rodadas igual a $O(n)$ até a definição da chave de conferência.

A busca por esquemas mais eficientes para o acordo de chaves de conferência tem despertado bastante atenção na comunidade de Criptografia. Em 2000, Joux [Joux 2000] demonstrou como utilizar emparelhamentos de Weil e Tate sobre curvas elípticas para construir um esquema com única rodada para acordo de chaves entre três participantes. O trabalho de Joux desencadeou uma série de novas pesquisas acerca de métodos para o acordo de chaves de conferência. Por um lado, uma série de trabalhos surgiram com o objetivo de aplicar o método de Joux à construção de protocolos mais eficientes para o acordo de chaves de conferência [Barua et al. 2003, Al-Riyami and Paterson 2003]. Por outro lado, uma série de estudos foi realizado com o objetivo de construir esquemas de acordo de chaves em um única rodada. Em 2002, Boneh e Silverberg [Boneh and Silverberg 2002] propõem diversas aplicações de formas multilineares à criptografia, incluindo a construção de esquemas de acordo de chaves de conferência em rodada única. Em 2013, Garg, Gentry e Halevi [Garg et al. 2013] descrevem construções baseadas em reticulados com propriedades que aproximam aquelas desejáveis para aplicações criptográficas, e apresentam diversos exemplos de aplicações, inclusive a esquemas de acordo de chaves de conferência. O método seria logo quebrado, em 2016, por Hu e Jia [Hu and Jia 2016], evidenciando a necessidade e o interesse pela definição de novos métodos para a construção de esquemas de acordo de chave em rodada única.

2.4. Equações Diofantinas

Muitos problemas de Teoria dos Números podem ser formulados como questões a respeito da solução de *equações Diofantinas*, ou seja, equações da forma $p(x_1, ..., x_n) = 0$ para um polinômio de várias variáveis e coeficientes inteiros. A importância das equações Diofantinas foi reconhecida por Hilbert ao formular suas famosa lista de problemas [Hilbert 1902] e relacionar o décimo problema como a busca por um algoritmo para determinar as raízes destas equações. Hoje, sabe-se que o problema não possui solução, ou seja, é indecidível no caso geral [Matijasevič 1970, Robinson 1972]. Mesmo para equações com duas variáveis, o problema é surpreendentemente difícil. Por exemplo, o teorema de Thue [Thue 1902], de 1909, enuncia que, para qualquer polinômio $f(x, y)$ irredutível sobre os racionais e homogêneo de grau 3, a equação $f(x, y) - m = 0$ possui no máximo um número finito de soluções inteiras. O argumento de Thue não permite construir um algoritmo que encontre tais soluções. Em 1967, Baker obteve um algoritmo não-determinístico de tempo exponencial para o problema — concluindo-se que, o problema, embora computável, é, aparentemente, intratável. Em 1978, Manders e Addleman [Manders and Adleman 1978] mostram que, para equações da forma $\alpha x_a^2 + \beta x_b = \gamma$, o problema de decidir se existe solução com x_a e x_b assumindo valores naturais é NP-completo.

3. Esquema de Acordo de Chaves Baseado em Funções Quadráticas

O esquema proposto no presente trabalho baseia-se na construção de uma chave comum a Alice e Bob a partir da troca de mensagens que contém o resultado de uma função

© 2017 Sociedade Brasileira de Computação

quadrática aplicada a valores aleatórios sorteados por Alice (e enviados a Bob) e por Bob (e enviados a Alice). Veremos que escolhendo valores apropriados para os parâmetros da função quadrática, é possível garantir que Alice e Bob chegarão à mesma chave.

Os parâmetros de domínio para o esquema proposto são os parâmetros α e β da função quadrática, um parâmetro de dimensão δ, e uma base y. Esses parâmetros de domínio podem ser gerados por um dos participantes ou por uma terceira parte confiável (Trent). Por questões de clareza, assumimos, no presente trabalho, que uma terceira parte confiável é responsável pela geração dos parâmetros de domínio.

O algoritmo a seguir descreve o funcionamento geral do esquema proposto. Usamos φ para representar a função Totiente de Euler, escrevemos $p|q$ significando que $q = 0 \bmod p$ e escrevemos $p \nmid q$ significando que $q \neq 0 \bmod p$.

Algoritmo 1: Procedimento para combinação de uma chave comum entre duas partes.

- Trent:
  1. Escolha um número natural δ
  2. Escolha um número natural β tal que $\varphi(\delta) \nmid \beta$
  3. Escolha α, tal que $\varphi(\delta)|\alpha$.
  4. Escolha $y < \delta$ tal que $\text{mdc}(y, \delta) = 1$.
  5. Publique y, α, β e δ.
- Alice:
  1. Escolha $x_a < \delta$
  2. Escolha $x_b < \varphi(\delta)$ tal que $\varphi(\delta) \nmid x_b\beta$
  3. Calcule $\gamma = \alpha x_a^2 + \beta x_b$.
  4. Envie para Bob (publique) γ e δ.
- Bob:
  1. Escolha $x'_a < \delta$
  2. Escolha x'_b tal que $\varphi(\delta) \nmid x'_b\gamma$
  3. Calcule $\gamma' = \alpha {x'_a}^2 + \beta x'_b$.
  4. Envie para Alice (publique) γ'.
- Alice:
  1. Calcule $k_{ab} = y^{\gamma' \cdot x_b} \bmod \delta$.
- Bob:
  1. Calcule $k_{ba} = y^{\gamma \cdot x'_b} \bmod \delta$.

3.1. Corretude do algoritmo

Mostramos, a seguir, que o Algoritmo 1 retorna uma chave comum entre Alice e Bob.
Teorema 1. *As chaves criptográficas k_{ab} e k_{ba} são idênticas. Adicionalmente, $k_{ab} \neq y$.*

Demonstração.

$$\begin{aligned} k_{ab} &= y^{\gamma' x_b} \bmod \delta \\ &= y^{(\alpha {x'_a}^2 + \beta x'_b) \cdot x_b} \bmod \delta \\ &= y^{\alpha {x'_a}^2 x_b + \beta x'_b x_b} \bmod \delta \\ &= y^{\alpha {x'_a}^2 x_b} \cdot y^{\beta x'_b x_b} \bmod \delta \\ &= (y^{{x'_a}^2 x_b})^{\alpha} \cdot y^{\beta x'_b x_b} \bmod \delta \end{aligned}$$

 © 2017 Sociedade Brasileira de Computação

Considerando que mdc$(y, \delta) = 1$ e $\varphi(\delta)|\alpha$, o Teorema de Euler implica $(y^{{x'_a}^2 x_b})^\alpha \equiv 1 \bmod \delta$. Logo, $k_{ab} \equiv y^{\beta x'_b x_b} \bmod \delta$.

De forma análoga, podemos mostrar que $k_{ba} = y^{\gamma \cdot x'_b} \bmod \delta \equiv y^{\beta x_b x'_b} \bmod \delta$.

Portanto, $k_{ab} = k_{ba}$.

□

3.2. Escolha dos parâmetros de domínio

Como se sabe, a segurança de um esquema criptográficos assimétricos depende de forma decisiva dos critérios utilizados para a escolha dos parâmetros de domínio [Barker et al. 2013]. Isso porque as restrições impostas aos parâmetros de domínio é que irão definir se os "problemas computacionais" associado aos cenários de ataque são, de fato, intratáveis e, inviabilizarão a realização bem-sucedida de ataques tais como a recuperação da chave.

Sabe-se [Manders and Adleman 1978] que determinar se existem x_a e x_b naturais que satisfaçam a equação $\gamma = \alpha x_a^2 + \beta x_b$, dados α, β e γ naturais, é um problema NP-completo [Manders and Adleman 1978]. Em relação ao esquema proposto no presente trabalho, a segurança do procedimento depende do padrão utilizado para escolher β e δ. Em relação à α, podemos assumir que $\alpha = \varphi(\delta)$.

Apresentamos, a seguir, dois possíveis "padrões" a serem seguidos no que diz respeito à escolha de β e δ.

Padrão 1: β e δ como produtos de primos

- Escolha duas sequências de primos distintos $p_1, p_2, ..., p_m$ e $q_1, q_2, ..., q_n$, (os valores m e n irão impactar o tamanho de chave e o nível de segurança).
- Calcule $\delta = \Pi_i^n p_i$, $\alpha = \Pi_i^n (p_i - 1)$ e $\beta = \Pi_j^m q_j$.

Padrão 2: δ como produto de cubos de primos, $\alpha|\beta^2$, mas $\alpha \nmid \beta$

- Escolha duas sequências de primos distintos $p_1, p_2, ..., p_m$ e $q_1, q_2, ..., q_n$, (os valores m e n irão impactar o tamanho de chave e o nível de segurança).
- Calcule $\delta = \Pi_i^n p_i^3$, $\alpha = \Pi_i^n p_i^2 (p_i - 1)$ e $\beta = \Pi_i^n p_i (p_i - 1) \cdot \Pi_j^m q_j$.

3.3. Um pequeno exemplo

Para ilustrar o funcionamento do algoritmo, vamos mostrar um pequeno exemplo usando o Padrão 1.

- Seja $n = 2$ e $m = 3$.
- Alice escolhe $p_1 = 3$, $p_2 = 11$, $q_1 = 5$, $q_2 = 7$ e $q_3 = 13$.
- Isto significa que $\delta = 33$, $\alpha = 20$ e $\beta = 455$.
- Alice escolhe $y = 10$.
- Alice escolhe $x_a = 14$ e $x_b = 37$.
- Alice calcula $\gamma = 20755$.

© 2017 Sociedade Brasileira de Computação

- Alice envia para Bob a sequência $(10, 20, 455, 20755, 33)$
- Bob recebe a sequência de Alice.
- Bob escolhe $x'_a = 11$ e $x'_b = 23$.
- Bob calcula $\gamma' = 12885$.
- Bob envia para Alice $\gamma' = 12885$.
- Alice calcula $k_{ab} = 10$.
- Bob calcula $k_{ba} = 10$.

4. Análise da segurança do esquema proposto

O esquema proposto na Seção 3 será inseguro se for possível obter informações sobre x_a, x_b ou k_{ab} a partir de $\{\alpha, \beta, \delta, \gamma, \gamma', y\}$. Podemos ver que se um espião Eva conhecer x_a, então Eva pode calcular x_b, assim como, se conhecer x_b, Eva consegue calcular x_a. A dificuldade de calcular x'_a é a mesma de calcular x_a. O mesmo acontece com x'_b. Se ele tiver x_b e x'_b ele consegue calcular a chave k_{ab}.

Nas próximas subseções, estudamos a segurança do esquema proposto face a diversos cenários de ataque. Na Subseção 4.1, estudamos um modelo de ataque utilizando técnicas "clássicas", enquanto na Subseção 4.2, estudamos um modelo de ataque utilizando um computador capaz de resolver o Problema do Logaritmo Discreto (como é o caso do modelo de computação quântica). A Subseção 4.3 apresenta cenários de ataque sobre o exemplo apresentado ao final da Seção 3.

Os resultados do estudo levam a crer que o esquema proposto parece ser robusto para a Computação Convencional, mas é frágil para ataque com um Computador Quântico que consiga resolver o Problema do Logaritmo Discreto.

4.1. Modelo de ataque clássico

Uma possível estratégia para resolver o problema é fatorar β e calcular as possíveis soluções módulo potências dos fatores primos de β. Este problema continua sendo difícil, mesmo conhecendo os primos que compõem β.

Se α e β não forem coprimos (como é o caso do padrão 2), podemos dividir toda a equação por $\text{mdc}(\alpha, \beta)$.

Se α e β forem coprimos, mas β e γ não, temos que $\text{mdc}(\beta, \gamma)$ divide x_a^2. Se β for produto de primos distintos (como no padrão 1), então $\text{mdc}(\beta, \gamma)$ divide x_a.

Em qualquer caso, temos que $\text{mdc}(\alpha, \beta)$ divide γ. E então dividimos toda a equação por $\text{mdc}(\alpha, \beta)$, obtendo os mesmos ou novos α_1 e β_1 coprimos e γ_1. Pela equação $\alpha_1 x_a^2 + \beta_1 x_b = \gamma_1$, tem-se que $\alpha_1 x_a^2 \equiv \gamma_1 \bmod \beta_1$, e consequentemente $x_a^2 \equiv \gamma_1 \alpha_1^{-1} \bmod \beta_1$. Esta última equação pode ter até $2^{n'}$ soluções, considerando que β_1 seja o produto de n' primos distintos. Da mesma forma, ao procurar $x_b \bmod \alpha$, tem-se $\beta x_b \equiv \gamma \bmod \alpha$, ou seja, $x_b \equiv \gamma\beta^{-1} \bmod \alpha$. A mesma consideração sobre o $\text{mdc}(\alpha, \beta) = 1$ pode ser feita aqui. E após, obter novos valores de α_1, β_1 e γ_1, visto que $\text{mdc}(\alpha_1, \gamma_1)$ divide x_b. Isto significa que sabemos que $x_b = \gamma_\alpha \beta_\alpha + t\alpha$, onde β_α e γ_α são os restos da divisão de β e γ por α respectivamente, para algum t. Substituindo x_b na equação $\alpha x_a^2 + \beta x_b = \gamma$, tem-se $\alpha x_a^2 + \gamma_\alpha \beta \beta_\alpha + t\alpha\beta = \gamma$, que pode ser escrito como $\alpha x_a^2 + \overline{\beta} t = \overline{\gamma}$, onde $\overline{\beta} = \alpha\beta$ e $\overline{\gamma} = \gamma - \gamma_\alpha \beta \beta_\alpha$, que é uma equação do mesmo formato que a equação original e portanto com o mesmo grau de dificuldade de ser calculada.

 © 2017 Sociedade Brasileira de Computação

Sobre o tamanho da chave

Considerando que a chave final é calculada módulo δ, obviamente o tamanho da chave depende do valor de δ, que por sua vez depende de m, quantidade de primos que o compõe. Se δ for muito pequeno, pode-se fazer um ataque de força bruta. Se δ for muito maior do que β, existirão poucos valores possíveis para t na expressão. $x_b = \gamma_\alpha \beta_\alpha + t\alpha$. Conforme comentado anteriormente, a quantidade de diferentes soluções possíveis para a expressão $x_a^2 \equiv \gamma\alpha^{-1} \bmod \beta$, depende do número de primos distintos, n, que compõe β. Isso significa que quanto maior valor de n, maior a quantidade de tentativas necessárias para encontrar a chave. O tamanho da chave depende do tamanho dos primos que compõe δ e β.

4.2. Modelo de ataque baseado no cálculo do Logaritmo Discreto

Considerando que $\alpha = \varphi(\delta)$.

- **Para encontrar x_b:** Um intruso pode encontrar x_b mod γ da seguinte forma:
  1. Escolha $y < \delta$ tal que $\text{mdc}(y, \delta) = 1$ e preferencialmente que y seja um gerador de Z_δ.
  2. Calcule $c = y^\gamma \bmod \delta$.
  3. Calcule $y' = y^\beta$.
  4. Encontre x tal que $c = (y')^x \bmod \delta$.

  Podemos observar que $x = x_b \bmod \gamma$.
- **Para encontrar a chave k_{ab}:**
  1. Calcule $c = y^{\frac{\gamma\gamma'}{mdc(\alpha,\beta)}} \bmod \delta$.
  2. Calcule $y' = y^{\frac{\beta}{mdc(\alpha,\beta)}}$.
  3. Encontre x tal que $c = (y')^x \bmod \delta$.

  A chave $k_{ab} = y^x \bmod \delta$.

Justificativa

Considerando que $\gamma\gamma' = \alpha^2 x_a^2 {x'_a}^2 + \alpha\beta x_a^2 x'_b + \alpha\beta {x'_a}^2 x_b + \beta^2 x_b x'_b$, temos que $\frac{\gamma\gamma'}{\text{mdc}(\alpha,\beta)} \equiv \frac{\beta}{\text{mdc}(\alpha,\beta)} \beta x_b x'_b \bmod \alpha$.

Podemos observar que $c \equiv y^{\frac{\gamma\gamma'}{mdc(\alpha,\beta)}} \bmod \delta \equiv (y')^{\beta x_b x'_b} \bmod \delta$.

4.3. Exemplo de aplicação dos ataques

A seguir, mostraremos como os ataques descritos na presente seção seriam executados para determinar os valores de x_a, x_b e k_{ab} do exemplo da Seção 3. Lembramos que os valores públicos do exemplo são os seguintes: $y = 10$, $\alpha = 20$, $\beta = 455$, $\gamma = 20755$ e $\delta = 33$).

Determinando x_a com o ataque 4.1

Como $\text{mdc}(\alpha, \beta) = 5 \neq 1$, não seria possível calcular $\gamma\alpha^{-1} \bmod \beta$ diretamente, uma vez que seria necessário para deduzir x_a. Por esse motivo, é necessário dividir α, β e γ, por

© 2017 Sociedade Brasileira de Computação

5, obtendo os novos valores $\alpha_1 = 4, \beta_1 = 91, \gamma_1 = 4151$. O mdc$(\beta_1, \gamma_1) = 7$. Portanto, sabemos que x_a é multiplo de 7.

Neste caso, para os *novos* $\alpha_1 = 4$, $\beta_2 = 13$ e $\gamma_2 = 593$, temos que $x_a^2 \equiv$ mdc$(\beta_1, \gamma_1)\gamma_2\alpha_1^{-1} \equiv 7 \times 8 \times 10 \bmod \beta_2$, ou seja $x_a^2 \equiv 1 \bmod 13$. Para encontrar os possíveis valores de $x_a \bmod \beta_2$, usamos o Teorema Chinês do Resto sobre o sistema com as congruências módulo fatores primos de β_2; que neste caso, apenas uma congruência. Esta congruência possui soluções 1 e -1 módulo β_2. Ou seja, x_a é da forma $1 + 13q$ ou $12 + 13q$, para algum q natural. Sabemos também que $x_a < \delta$ e é múltiplo de 7. Como havia uma única congruência no sistema, há 2 possíveis soluções para a congruência módulo β_2. Se houvesse n' congruências (n' primos distintos compondo β_2), haveria $2^{n'}$ possíveis soluções.

Determinando x_b com o ataque 4.1

Como mdc$(\alpha, \beta) \neq 1$, não é possível, o atacante não consegue calcular $\gamma\alpha^{-1} \bmod \beta$ para os valores originais. Por esse motivo, usamos os novos valores $\alpha = 4, \beta = 91, \gamma = 4151$, como no cenário anterior.

Determinando x_b com o ataque em 4.2

1. Tomemos $y = 10$ (como escolhido pela Alice).
2. $c \equiv 1 \bmod 33$
3. $y' \equiv 10 \bmod 33$
4. x deve satisfazer a congruência $1 \equiv 10^x \bmod 33$. Ou seja, x_b é múltiplo de 2.

Rodando novamente para outro valor de y temos

1. Tomemos $y = 2$.
2. $c \equiv 32 \bmod 33$
3. $y' \equiv 32 \bmod 33$
4. x deve satisfazer a congruência $11 \equiv 11^x \bmod 33$. Ou seja, x_b pode ser qualquer valor.

Mais uma tentativa...

1. Tomemos $y = 17$.
2. $c \equiv 11 \bmod 33$
3. $y' \equiv 11 \bmod 33$
4. x deve satisfazer a congruência $11 \equiv 11^x \bmod 33$. Ou seja, x_b pode ser qualquer valor.

Determinando k_{ab} com o ataque em 4.2

1. $c = 10^{(20755 \times 12885)} \bmod 33 \equiv 10 \bmod 33$.
2. $y' = y^\beta \equiv 10 \bmod 33$.
3. $x = 1$.
4. $k_{ab} = 10$.

 © 2017 Sociedade Brasileira de Computação

5. Esquema de Acordo de Chave de Conferência

A seguir, mostraremos como o esquema proposto no presente trabalho permite a instanciação eficiente de chaves de conferência. O segredo da eficiência do método é o fato de que o parâmetro γ_i enviado por um participante i a todos os outros pode ser utilizado diretamente para construir chaves de conferência para qualquer conjunto de participantes, mas γ_i, por si só, não configura uma chave criptográfica para nenhum grupo. Esta propriedade difere fundamentalmente do método de Diffie e Hellman, onde, para construir uma chave compartilhada $k_{s_1,...,s_n} := g^{s_1 \cdot ... \cdot s_n}$, um participante i precisa ter acesso a $k_{s_1,...,s_{i-1},s_{i+1},...,s_n} := g^{s_1 \cdot ... \cdot s_{i-1} \cdot s_{i+1} \cdot ... \cdot s_n}$.

Acordo de chave usando o Padrão 1

Considerando que foi utilizado o Padrão 1, é possível acordar uma chave comum com qualquer sub-cunjunto de um grupo da seguinte forma.

- Um representante do grupo:
  1. Escolhe δ e β conforme o padrão 1.
  2. Calcula $\alpha = \varphi(\delta)$.
  3. Escolhe y coprimo com δ.
  4. Publica y, α, β, γ e δ.
- Cada usuário i do grupo após receber a sequência:
  1. Escolhe um par (x_{a_i}, x_{b_i}), tal que $\text{mdc}(x_{b_i}, \alpha) = 1$.
  2. Calcula $\gamma_i = \alpha x_{a_i}^2 + \beta x_{b_i}$.
  3. Publica γ_i.

Seja $\{e_1, e_2, ..., e_r\} \in \{1, ..., s\}$ o sub-grupo de r partes (de um total de s) que se desejam comunicar. Para o e_1, por exemplo, saber a chave deste sub-grupo, basta calcular $k = y^z \bmod \delta$, onde $z \equiv x_{b_{e_1}} \Pi_{t=2}^{r} \gamma_{e_t} \bmod \alpha$. Neste caso, teremos

$$k = y^{\beta^{r-1} \Pi_{t=1}^{r} x_{b_{e_t}}} \bmod \delta.$$

O procedimento acima não pode ser usado sobre o Padrão 2, pois é necessário que $\varphi(\delta) \not| \beta^{r-1}$; no entanto, no Padrão 2, temos que $\varphi(\delta) \not| \beta^2$. A seguir, mostraremos outro protocolo para o acordo de chaves que não exige que os parâmetros sigam os requisitos do Padrão 1.

Atualização dinâmica de chaves

Outra vantagem do esquema proposto é a capacidade de se realizar a atualização dinâmica de chaves por meio da simples modificação do parâmetro y. O método é simples: uma vez realizadas as transmissões de todos os γ_i por cada participante i de um grupo, é possível redefinir a chave a partir dos novos γ_i. Cada novo valor de y pode ser transmitido via broadcast ou pode ser gerado por um gerador de números pseudo-aleatórios, conhecido por todos, usando y anterior como semente.

© 2017 Sociedade Brasileira de Computação

Acordo de chaves usando qualquer padrão

Apresentamos, a seguir, um método que permite o acordo de chaves de conferência mesmo que os parâmetros de domínio não atendam aos requisitos especificados no Padrão 1. Embora o procedimento possa ser considerado mais "geral" — na medida em que os parâmetros de domínio não precisam atender a restrições especiais — veremos que o método não permite o acordo de chaves de conferência em rodada única — ao contrário, é necessário um número de rodadas logarítmico no número de participantes.

Algoritmo 2: Procedimento para combinação de uma chave comum entre três partes

- Alice publica α, β e δ.
- Alice e Bob combinam uma chave k_{ab} conforme algoritmo 1 com o padrão escolhido.
- Alice escolhe $x''_a < \delta$.
- Alice calcula $\gamma'' = \alpha x''^2_a + \beta k_{ab}$.
- Alice escolhe y
- Alice envia γ'' e y para Bob e para Carlos.
- Carlos escolhe seu x_{c1} e x_{c2} e calcula $\gamma''' = \alpha x^2_{c1} + \beta x_{c2}$.
- Carlos envia para Alice e para Bob γ'''.
- Alice e Bob calculam $k_{abc} = y^{\gamma'''k_{ab}} \bmod \delta$.
- Carlos calcula $k_{abc} = y^{\gamma''x_{c2}} \bmod \delta$.

Podemos observar que $k_{abc} = y^{\beta k_{ab} x_{c2}}$.

Combinação de duas chaves de conferência

Embora o método acima possa parecer mais limitado, ele apresenta vantagens em relação ao primeiro método apresentado em um cenário específico, que é quando dois conjuntos C_1 e C_2 de usuários, cada um dos quais já tendo estabelecido uma chave comum k_{C_1} e k_{C_2}, desejam estabelecer uma chave comum a $C_1 \cup C_2$. Neste caso, não é necessária a transmissão de cada um dos γ_i de cada um dos participantes de cada conjunto C_1 e C_2. Basta que um "representante" do conjunto C_1 transmita a chave de conferência k_{C_1} aos participantes de C_2, e um "representante" do conjunto C_2 transmita a chave de conferência k_{C_2} aos participantes de C_1, que todos os participantes serão capazes de estabelecer uma chave de conferência para $C_1 \cup C_2$.

Depois, um representante do grupo 1 faz o papel de Alice e um representante do grupo 2 faz o papel de Bob. Se considerarmos que o procedimento recursivo divide o grupo em dois subgrupos de tamanhos aproximadamente iguais, então a chave comum é estabelecida em $O(\log n)$ rodadas.

6. Considerações Finais

No presente trabalho, apresentamos um esquema de acordo de chaves criptográficas baseado na dificuldade de se resolver equações Diofantinas de grau 2 com duas variáveis. O esquema proposto apresenta evidentes benefícios no cenário em que se deseja estabelecer "chaves de conferência".

 © 2017 Sociedade Brasileira de Computação

Pelo fato de se basear em um problema NP-completo, o esquema seria um potencial candidato a esquema resistente a ataques originado de um computador quântico. Infelizmente, como mostramos no presente artigo, o esquema é vulnerável a uma atacante que seja capaz de realizar a computação eficiente do logaritmo discreto — e, portanto, vulnerável a um atacante de posse de um computador quântico. Ainda assim, mantemos em aberto a questão sobre se seria possível, por meio de uma escolha adequada de parâmetros de domínio, tornar o esquema proposto resistente a tais ataques.

Referências

Al-Riyami, S. S. and Paterson, K. G. (2003). *Tripartite Authenticated Key Agreement Protocols from Pairings*, pages 332–359. Springer Berlin Heidelberg, Berlin, Heidelberg.

Ateniese, G., Steiner, M., and Tsudik, G. (2000). New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639.

Barker, E., Chen, L., Roginsky, A., and Smid, M. (2013). *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (NIST Special Publication 800-56A, Revision 2)*. National Institute of Standards and Technology.

Barua, R., Dutta, R., and Sarkar, P. (2003). Extending joux's protocol to multi party key agreement. *IACR Cryptology ePrint Archive*, 2003:62.

Boneh, D. and Silverberg, A. (2002). Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90.

Chunsheng, G. (2015a). Multilinear maps using ideal lattices without encodings of zero. Cryptology ePrint Archive, Report 2015/023. `http://eprint.iacr.org/2015/023`.

Chunsheng, G. (2015b). Multilinear maps using ideal lattices without encodings of zero.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654.

Diffie, W., Van Oorschot, P. C., and Wiener, M. J. (1992). Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125.

ElGammal, T. The first ten years of public-key cryptograph. *Proceedings of the IEEE*, 76.

ElGammal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31.

Garg, S., Gentry, C., and Halevi, S. (2013). *Candidate Multilinear Maps from Ideal Lattices*, pages 1–17. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hilbert, D. (1902). Mathematical problems. *Bull. Amer. Math. Soc.*, 8(10):437–479.

Hu, Y. and Jia, H. (2016). Cryptanalysis of ggh map. In *Proceedings of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9665*, pages 537–565, New York, NY, USA. Springer-Verlag New York, Inc.

© 2017 Sociedade Brasileira de Computação

Joux, A. (2000). *A One Round Protocol for Tripartite Diffie–Hellman*, pages 385–393. Springer Berlin Heidelberg, Berlin, Heidelberg.

Lee, Y.-R., Lee, H.-S., and Lee, H.-K. (2004). Multi-party authenticated key agreement protocols from multi-linear forms. *Applied Mathematics and Computation*, 159(2):317 – 331.

Manders, K. L. and Adleman, L. (1978). Np-complete decision problems for binary quadratics. *Journal of Computer and System Sciences*, 16(2):168 – 184.

Matijasevič, J. V. (1970). The diophantineness of enumerable sets (russian). *Dokl. Akad. Nauk SSSR*, 191:279–282.

Matsumoto, T., Takashima, Y., and Ima, H. On seeking smart public-key distribution systems. *The Transactions of the IECE of Japan*, E69.

Robinson, J. (1972). Review: Ju. v. matijasevic, a. doohovskoy, enumerable sets are diophantine. *J. Symbolic Logic*, 37(3):605–606.

Thue, A. (1902). Uber annaherungswerte algebraischer. *J. Reine Angew Math*, 135:284–305.

 © 2017 Sociedade Brasileira de Computação

Aplicação Dinâmica de Políticas Estáticas de Fluxo

Carina Capelão de Oliveira[1], **Glauco Gonçalves Cardoso**[2], **Fernando Magno Quintão Pereira**[1]

[1]Departamento de Ciência da Computação - UFMG
Avenida Antônio Carlos, 6627 - 31.270-010 - Belo Horizonte - MG - Brasil

[2]Companhia de Tecnologia do Estado de Minas Gerais - PRODEMGE
Belo Horizonte - MG - Brasil

{carina,fernando}@dcc.ufmg.br, glauco.cardoso@prodemge.gov.br

Abstract. *Information flow analysis is today one of the most important techniques to find vulnerabilities in programs. Since its conception, this technique has evolved to achieve great precision and scalability. However, being static, information flow analysis has one shortcoming: to ensure soundness, it must conservatively refuse some programs which are well-known to be safe. Even exponential-time algorithms cannot change this scenario, as long as they remain static. To address this problem, in this paper we propose a combination of static analysis with dynamic checks to increase the precision of information flow techniques. Our dynamic checks let us rule out unsafe paths during the execution of the program while preserving secure flows of information. In this way, we can certify the safety of programs even in face of features which are difficult to analyze statically, such as subtyping polymorphism. To validate our ideas, we have materialized them into a tool, InspectorJ, which inserts instrumentation in a program to prevent the execution of unsafe paths.*

Resumo. *A análise de fluxo de informação é atualmente uma das técnicas mais importantes para descobrir vulnerabilidades em programas. Desde a sua concepção, esta técnica evoluiu ao ponto de alcançar grande precisão e escalabilidade. Entretanto, por ser estática, a análise de fluxo de informação é conservadora e, para garantir corretude, ela precisa invalidar alguns programas que são conhecidamente seguros. Enquanto a análise permanece totalmente estática, nem mesmo algoritmos exponenciais conseguem mudar esse cenário. Para abordar esse problema, propomos uma combinação de análise estática e checagem dinâmica para ampliar a precisão de técnicas de análise de fluxo de informação. Nossas checagens dinâmicas nos permitem detectar e descartar caminhos inseguros durante a execução do programa e preservar fluxos seguros de informação. Dessa forma, podemos garantir a segurança de programas mesmo diante de propriedades difíceis de serem analisadas estaticamente, como por exemplo polimorfismo de subtipagem. Para validar nosso trabalho, materializamos nossas ideias em uma ferramenta chamada InspectorJ, que insere instrumentação em um programa para prevenir a execução de caminhos inseguros.*

1. Introdução

Análises de fluxo de informação são uma das tecnologias mais importantes para encontrar vulnerabilidades de segurança em software. Devido à sua importância, desde o trabalho de

© 2017 Sociedade Brasileira de Computação

Denning e Denning [Denning and Denning 1977] nos anos setenta, muito tem sido feito para aumentar a precisão e a eficiência de tais técnicas. Dentre as implementações estado da arte atualmente em uso, encontram-se abordagens esparsas [Bruno Rodrigues 2016], dinâmicas [Nethercote and Seward 2007], sensíveis ao fluxo [Hunt and Sands 2006] e sensíveis ao contexto. O processo de implementação de técnicas de fluxo de informação tem tornado possível a sanitização de extensas bases de código, o que aumenta a confiança dos usuários na segurança de sistemas de software que eles usam.

Entretanto, apesar de todos os desenvolvimentos recentes, existe ainda um empecilho à implementação de análises de fluxo totalmente precisas: a dificuldade para analisar e entender automaticamente linguagens que usam polimorfismo de subtipagem acoplado com invocação dinâmica. Essas características, comuns em linguagens de programação orientadas a objeto, comprometem a precisão de análises de fluxo, pois as mesmas precisam rastrear todos os possíveis alvos de qualquer invocação de métodos. Para lidar com esse tipo de comportamento de execução, técnicas de fluxo geralmente recorrem a análises dinâmicas [Russo and Sabelfeld 2010]. Contudo, essa alternativa traz uma desvantagem significativa: mesmo abordagens sofisticadas podem prejudicar a performance de programas em até 35% [Clause et al. 2007].

Neste artigo, nós apresentamos uma solução para mitigar esse *overhead*. Projetamos, implementamos e testamos uma técnica que combina análise estática de fluxo de informação com checagem dinâmica, para possibilitar a validação de programas que, devido ao uso de polimorfismo de subtipagem, seriam considerados inseguros. Resumidamente, usamos uma técnica de análise estática para descobrir o maior número possível de caminhos seguros no programa, além de caminhos inseguros e caminhos que podem ser inseguros dependendo de algum valor conhecido apenas dinamicamente. A esses caminhos que não somos capazes de validar durante a primeira etapa, inserimos predicados de checagem dinâmica. Esses predicados são inseridos apenas em pontos de chamada que podem invocar métodos inseguros. Como mostraremos nas próximas seções, o número de chamadas desse tipo é extremamente limitado, o que leva nossa abordagem a adicionar um *overhead* muito pequeno aos programas validados. Na seção 2, apresentamos uma rápida visão geral dessas ideias, usando um exemplo para explicá-las ao leitor.

Para validar nosso trabalho, materializamos nossas ideias em uma ferramenta chamada InspectorJ, implementada no Soot, um framework para análise e otimização de bytecodes Java [Vallee-Rai et al. 1999]. A ferramenta analisa programas Java e insere instrumentação no código para prevenir a execução de caminhos inseguros. Nós configuramos nosso mecanismo de fluxo de informação para detectar um ataque conhecido como *command injection*. *Command injection* é um ataque cujo objetivo é executar comandos arbitrários no sistema operacional do cliente através de uma aplicação vulnerável. Este ataque é possível quando uma aplicação transfere dados inseguros fornecidos pelo usuário para um shell do sistema. Nossa análise foi capaz de detectar *command injection* em uma aplicação para o Flickr que está disponível em um diretório público no GitHub, que utiliza polimorfismo de subtipagem e invocação dinâmica. Detectamos o caminho inseguro e fomos capazes de instrumentar o código para previnir a execução de tal caminho, que atravessa as fronteiras de pelo menos 2 métodos[1]. Esse experimento nos dá embasamento

[1]O comprimento exato desse caminho, em número de invocações de método, depende do polimorfismo de subtipagem.

 © 2017 Sociedade Brasileira de Computação

```
1  public static class T{
2      boolean comp(String password, String input){
3          int res = 0
4          for (int i=0; i < 8: ++i){
5              res = res | (password.charAt(i) - input.charAt(i));
6          }
7          return res == 0;
8      }
9  }
10
11 public static class U extends T {
12     boolean comp(String password, String input) {
13         for (int i = 0; i < 8; i++) {
14                 if (password.charAt(i) != input.charAt(i)) {
15                 return false;
16             }
17         }
18         return true;
19     }
20 }
21
22 public static void main(String args[]) {
23     T o = args.lenght % 2 == 0 ? new U() : new T();
24     o.comp(args[0], args[1]);
25 }
```

Figura 1. Exemplo de programa Java não isócrono que apresenta polimorfismo de subtipagem e invocação dinâmica.

para validar, ao menos preliminarmente, a seguinte conjectura: o número de pontos de chamada que não podem ser completamente desambiguados por análises estáticas é pequeno o suficiente para tornar a aplicação de checagem dinâmica barata o suficiente para ser usada na prática. Esse experimento e outros adicionais serão discutidos na seção 5.

2. Visão Geral

Este artigo descreve técnicas que melhoram a precisão de análises de fluxo de informação. Tais técnicas podem ser usadas para apontar, automaticamente, uma vasta gama de vulnerabilidades em software. A fim de exemplificar nossas ideias, escolhemos, dentre tais vulnerabilidades, um problema de segurança bem conhecido: canais laterais baseados em tempo. Ataques por variação de tempo têm como foco inferir informações confidenciais, tais como uma chave criptográfica. Eles podem ocorrer quando esses dados sigilosos influenciam os predicados, que são estruturas condicionais das instruções de desvio de um programa. Tais estruturas definem as partes do código que serão executadas, controlando o tempo de execução do programa. O ataque se baseia em medir esse tempo para cada dado de entrada fornecido, o que permite ao adversário descobrir alguns ou até todos os bits da informação sigilosa. Nesta seção, usaremos o programa da Figura 1 como exemplo. O código é não isócrono, portanto é suscetível a ataques por variação de tempo. Além disso, ele apresenta polimorfismo de subtipagem e invocação dinâmica de métodos: duas características que tornam programas difíceis de analisar, e com as quais lidamos.

A Figura 1 apresenta um programa Java que possui duas classes, *T* e *U*, e um método *main*. A classe *U* possui uma função *comp* de comparação de strings, que recebe uma chave *password* e uma entrada do usuário *input*. As strings são comparadas caractere a caractere, e quando estes diferem, a função retorna. Um retorno antecipado indica que os caracteres iniciais são diferentes, enquanto um retorno tardio indica uma diferença nos últimos bits. Visto que o usuário controla o que será atribuído à variável

© 2017 Sociedade Brasileira de Computação

input, esta pode estar contaminada com informações maliciosas a fim de controlar o fluxo do programa, afetando quais regiões do código serão executadas. Assim, variando em ordem lexicográfica o conteúdo de *input*, e medindo o tempo que a função demora para retornar, é possível aprender alguma informação sobre o conteúdo de *password*. Portanto, para evitar esse tipo de ataque, é necessário impedir que o dado secreto alcance os predicados do código, uma vez que um atacante consegue transformar um problema NP-Difícil (descobrir um *password*) em um problema polinomial, basicamente medindo os tempos de execução, quebrando assim o algoritmo criptográfico. Dessa forma, o método *comp* da classe *U* é inseguro.

Por sua vez, a classe *T* também possui um método *comp* de comparação de strings, com algumas diferenças. As strings não são comparadas diretamente, de forma que a chave *password* não alcança nenhum desvio condicional. É feita uma manipulação de bits com os caracteres das strings: uma variável *res* é inicializada com 0, e é feita uma operação de *OR* entre essa variável e a diferença entre os caracteres das duas strings. Se algum caractere for diferente, *res* será diferente de 0, e a função retorna *false*. Caso contrario, *res* será sempre 0, e a função retorna *true*. Outro ponto importante é que o método não realiza retornos antecipados ou tardios, uma vez que ele sempre percorre toda a string antes de retornar o resultado. Portanto, o método dessa classe é seguro.

Ainda na Figura 1 temos o método *main*. Uma variável *o* do tipo da classe *T* é definida, de forma que o seu objeto instanciado será definido apenas em tempo de execução, podendo ser do tipo da classe *T* ou da classe *U*. Isso ocorre pois o tipo do objeto instanciado depende do valor de *args.length*, em que *args* é recebida por parâmetro pela função, portanto definida apenas durante a execução. Após a definição de *o*, o objeto chama o método *comp*. Uma vez que a classe U estende a classe *T*, e as duas possuem o método *comp*, não é possível saber estaticamente qual método será chamado. Porém, como o método U é vulnerável, o programa será inseguro quando o objeto *o* for do tipo *U*, e será seguro quando o objeto for do tipo *T*.

Para garantir a segurança do código, e para não dizer que o mesmo possui uma falha independente do tipo de *o* para proteger o programa, devemos identificar quando este é inseguro. Para isso, rastreamos todos os possíveis caminhos de qualquer chamada de método, e identificamos o caminho contaminado que deve ser evitado. Assim, para impedir que o método seja executado caso o tipo de *o* seja *U*, inserimos guardas no código, que checam em tempo de execução se o objeto que chamará a função *comp* é do tipo da classe *U*. Em caso positivo, instrumentamos o código para lançar uma exceção antes da chamada de *comp*. O código resultante da instrumentação é mostrado na Figura 2. Dessa forma, uma vez instrumentado, o programa estará seguro, pois o método inseguro não será executado.

Essa combinação de análise estática com checagem dinâmica para aumentar a precisão da técnica de fluxo de informação cobre mais casos em que o programa seria considerado inseguro conservativamente. Considerando o exemplo da Figura 1, se fosse feita apenas uma análise estática, então o programa *inteiro* seria considerado inseguro, embora somente um método apresentasse uma potencial vulnerabilidade – independente do tipo do objeto instanciado. Supondo que a primeira linha do método *main* da Figura 1 fosse *T o = args[0] == "0"? new U() : new T();*. Nesse caso se *args[0]* fosse igual a 0 o tipo do objeto instanciado seria igual a *U*, tornando o código vulnerável. Para qualquer

 © 2017 Sociedade Brasileira de Computação

```
1  public static void main(String args[]) {
2      T o = args.lenght %2 == 0 ? new U() : new T();
3
4      // BEGIN AUTOMATIC CODE:
5      if (o instanceof U)
6          throw new Error("C\'odigo Inseguro");
7      // END AUTOMATIC CODE.
8
9      o.comp(args[0], args[1]);
10 }
```

Figura 2. Instrumentação resultante da análise feita no programa da Figura 1.

outro valor de *args[0]* que fosse diferente de 0, o tipo do objeto instanciado seria igual a *T*, resultando em um código seguro[2]. Dessa forma, na maioria das vezes o programa não é vulnerável, mas seria considerado como tal caso utilizássemos apenas análise estática. Utilizando a checagem dinâmica juntamente com a análise estática, diminuímos o número de falsos positivos, enquanto mantemos o programa seguro.

3. A Análise Estática

Para detectar vulnerabilidades de segurança nos programas Java utilizando técnicas de análise estática de fluxo de informação, nossa abordagem segue três etapas. Primeiramente, analisamos o código estaticamente para rastrear o fluxo de informação do código, identificando dependências de dados e de controle entre as variáveis do programa. Em segundo lugar, criamos um grafo de dependências, a partir do rastreamento feito anteriormente, para representar e analisar como as variáveis estão relacionadas. Finalmente, realizamos uma análise de fluxo contaminado no grafo criado para identificar os caminhos seguros e inseguros e, assim, detectar uma vulnerabilidade de segurança, como por exemplo o ataque por variação de tempo. Essas etapas serão melhor explicadas a seguir.

3.1. Rastreando o fluxo

Para ter uma análise mais eficaz e identificar também as falhas que podem ser introduzidas pelo compilador, nossa análise trabalha em nível de compilador, analisando a representação intermediária do programa gerada pelo mesmo. Tal representação intermediária possui o formato de *Static Single Assignment form (SSA)* [Cytron et al. 1991], em que cada variável tem somente um ponto de definição no código intermediário. Além disso, se existe um desvio condicional, tal como *if else*, em que uma variável é definida nos dois caminhos possíveis, então ao final do desvio é inserida uma função *phi*. Funções *phi* são da forma $\mathsf{i1} = \phi(\mathsf{i0}, \mathsf{i2})$ e são usadas para unificar múltiplas definições da mesma variável — i0 e i2 — em um nome único, no caso i1. Outra característica dessa representação intermediária é que todas as instruções possuem no máximo três endereços. Essas duas características fazem com que novas variáveis que antes não existiam no código original sejam criadas nesse novo programa.

Em seguida, é feito um rastreamento do fluxo no novo programa gerado para identificar dependências de dados e de controle entre as variáveis do código, em que *dependência de dados* e *dependência de controle* são definidas como:

[2]Esse exemplo é bastante trivial e dificilmente aconteceria em um cenário real, mas nos auxilia a entender o problema.

© 2017 Sociedade Brasileira de Computação

Definição 3.1 (Dependência de Dados) *Uma variável v depende de dados de uma variável u se uma instrução que usa u for usada para definir v. Um exemplo seria* `v=u+2`. *Dessa forma, dizemos que existe um fluxo explícito de u para v.*

Definição 3.2 (Dependência de Controle) *Uma variável v é dependente de controle de uma variável u se a definição de v depende do valor de u. Um exemplo seria* `se u então v=0`. *Dessa forma, dizemos que existe um fluxo implícito de u para v.*

3.2. Grafo de Dependências

Para representar as relações de dependências do programa, criamos um *grafo de dependências*, definido como:

Definição 3.3 (Grafo de Dependências) *Um grafo de dependências é criado a partir da relação de dependência de dados e de controle entre as variáveis do programa. Cada vértice "n_v"do grafo representa uma variável "v"do programa e os "sorvedouros"de uma análise específica, que serão definidos posteriormente. Cada aresta "a_{uv}"representa uma dependência. Arestas sólidas representam dependências de dados e arestas não sólidas representam dependências de controle. Podemos assegurar que cada vértice corresponde a uma e somente uma variável no programa devido ao formato SSA do programa analisado.*

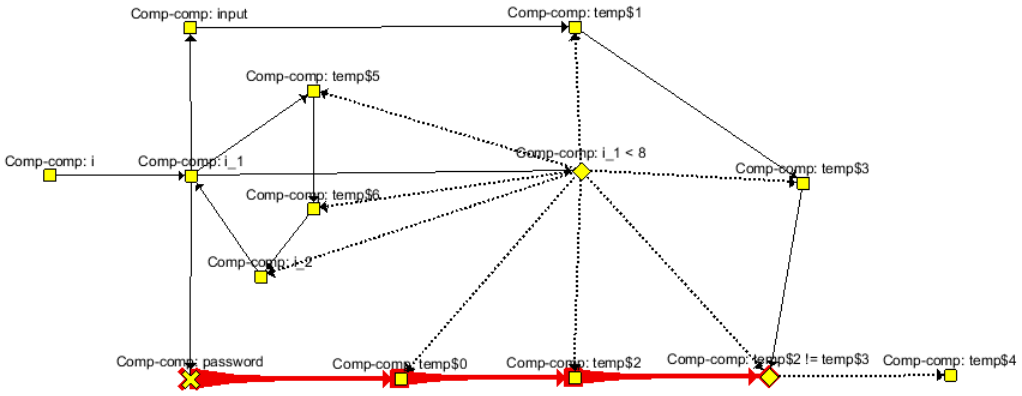


Figura 3. Grafo de dependências resultante para o método comp da classe U do exemplo da Figura 1.

O grafo nos permite visualizar como os dados trafegam dentro do programa, o que facilita sua análise e entendimento. A Figura 3 mostra o grafo produzido para o método *comp* da classe *U* da Figura 1. Uma variável é representada como um quadrado, um predicado de desvio condicional é um losango e a variável definida como fonte na análise é representada como um X. Uma aresta sólida de n_u para n_v indica que o programa contém uma instrução que usa a variável *u* e define a variável *v*. Uma aresta não sólida de *p* para *q* indica que o programa contém um teste condicional sob o predicado *p*, de forma que o valor a ser atribuído à variável *q* depende desse teste.

3.3. Análise de fluxo contaminado

Para detectar um determinado tipo de vulnerabilidade, como o ataque por variação de tempo explicado na seção 2, realizamos uma análise de fluxo contaminado no grafo criado. Definimos *análise de fluxo contaminado* como:

© 2017 Sociedade Brasileira de Computação

Definição 3.4 (Análise de Fluxo Contaminado) *A análise de fluxo contaminado recebe uma tripla T = (G, Fonte, Sorvedouro), em que G é o grafo de dependências definido anteriormente. Uma fonte é um dado fornecido pelo usuário, ou seja, é uma informação vinda do canal de entrada, que é passado para o programa. Um sorvedouro é um ponto sensível do programa, ou seja, são funções que realizam operações sensíveis (tal como escrever uma informação no canal de saída). Um caminho contaminado é um caminho no grafo de dependências entre a fonte e o sorvedouro. Assim, a análise de fluxo contaminado detecta se existe um caminho no grafo de dependências entre uma fonte e um sorvedouro. [Rimsa et al. 2011]*

Dizemos que um caminho é contaminado quando uma fonte alcança um sorvedouro pois a fonte é uma informação vinda do usuário, podendo conter dados maliciosos criados com o propósito de realizar algum ataque no sistema. Para cada tipo de vulnerabilidade a ser detectada, vamos definir uma fonte e sorvedouro específicos. Considerando o exemplo da Figura 1, em que queremos detectar se existe uma variação no tempo, a fonte seria a variável *password* do método *comp*, representada no grafo na Figura 3 como um X. Por sua vez, o sorvedouro, representado como um losango, seria o desvio condicional presente na linha 14. Como existe um caminho entre a fonte e o sorvedouro, identificamos esse caminho destacando-o de vermelho, como pode ser visto no grafo.

Para construir tais caminhos, a análise marca como contaminada a variável do programa que foi definida como a fonte. Todo vértice conectado com a fonte e as arestas que fazem essa ligação são também marcados. E assim sucessivamente: todo vértice conectado com um vértice contaminado é também marcado, e esta informação flui ao longo do grafo. Se um vértice marcado alcançar o sorvedouro, o caminho é contaminado. Os caminhos encontrados são identificados no grafo e reportados ao usuário. Cada vértice do grafo carrega o nome da variável que ele representa, além de conter o número da linha do programa Java da instrução que a definiu. O método e classe aos quais a variável pertence também são mantidos. Assim, o desenvolvedor consegue facilmente identificar no seu código o ponto em que a fonte alcança o sorvedouro e corrigir o problema.

4. A checagem dinâmica

Acoplada à análise estática, realizamos uma checagem dinâmica para possibilitar a validação de programas que, devido ao uso de polimorfismo de subtipagem e invocação dinâmica, seriam considerados inseguros. Esta segunda parte da análise possui 2 etapas. Primeiramente, tratamos o polimorfismo de subtipagem a fim de descobrir todos os tipos que um determinado objeto poderia assumir dinamicamente. Depois, identificamos todas as invocações dinâmicas feitas e instrumentamos o código nestes pontos afim de evitar que código inseguro seja executado. Essas etapas serão melhor explicadas a seguir.

4.1. Análise de polimorfismo de subtipagem

Uma característica comum em linguagens de programação orientadas a objeto é o uso de polimorfismo de subtipagem com invocação dinâmica. O polimorfismo de subtipagem ocorre quando um determinado objeto é definido com um tipo que tem vários subtipos, em que o subtipo vai ser definido no momento da instanciação. Um exemplo disso é quando uma classe possui várias subclasses que a estendem, ou quando uma interface é implementada por várias classes. Assim, quando todas as subclasses de uma classe possuem um método com a mesma assinatura, que realiza operações diferentes para cada

© 2017 Sociedade Brasileira de Computação

subclasse, só saberemos qual método vai realmente executar em uma chamada se soubermos o subtipo do objeto. Entretanto, se a instanciação de um objeto é feita apenas dinamicamente, não é possível saber qual o tipo instanciado do objeto através apenas da realização de análise estática. Considere por exemplo o método main da Figura 1. A variável *o* é definida como do tipo da classe *T*, em que seu objeto instanciado será definido apenas em tempo de execução, podendo ser do tipo da classe *T* ou da classe *U* (como foi explicado na seção 2). No momento em que o objeto *o* chama o método *comp*, não sabemos se o método chamado será da classe *T*, que é seguro, ou da classe *U*, que é inseguro. Assim, para detectar e descartar caminhos inseguros durante a execução do programa e preservar fluxos seguros de informação, devemos rastrear todos os possíveis tipos que um objeto pode assumir, e assim rastrear todos os possíveis alvos de qualquer invocação de métodos.

Para isso, durante a fase de rastreamento de fluxo da análise estática, realizamos uma análise de polimorfismo de subtipagem. Para variáveis de tipos primitivos - como um *int* - que possuem apenas tipo estático, a análise é feita como descrita na seção 3. Para objetos que podem assumir mais de um tipo dinâmico, é necessário uma análise extra. Tipo estático e tipo dinâmico são definidos como:

Definição 4.1 (Tipo Estático) *O Tipo Estático de um objeto é o tipo usado em sua declaração. Tal tipo nunca muda durante a execução do programa, uma vez que ele é parte do texto daquele programa.*

Definição 4.2 (Tipo Dinâmico) *O Tipo Dinâmico de um objeto é o tipo com o qual ele é instanciado, e pode mudar sempre que o objeto recebe uma nova instância. Tal tipo pode mudar durante a execução do programa se o tipo a ser instanciado depender de algum valor definido apenas dinamicamente.*

No exemplo da Figura 1, no método *main*, o objeto *o* pode ter mais de um tipo dinâmico, uma vez que o tipo instanciado depende do tamanho da variável *args*, valor este conhecido apenas dinamicamente. Dessa forma, o tipo do objeto *o* é definido apenas em tempo de execução, e portanto o mesmo pode assumir tipos diferentes em cada execução do programa. Por sua vez, a variável *args* possui apenas tipo estático, uma vez que ela possui tipo *builtin*, que não podem ter subtipos em Java.

Para todos os objetos que possuem tipo dinâmico, fazemos uma análise de definições alcançáveis de cada um desses objetos para identificar todos os possíveis tipos que o objeto pode assumir durante a execução do programa, e armazenamos tais dados sobre cada objeto junto com o objeto em si para uso posterior. Quando encontramos um ponto de chamada de um método feita por um objeto *o*, salvamos a linha do programa Java em que a chamada foi feita. Além disso, analisamos o método de cada um dos tipos que o objeto *o* pode ser, descobrindo e criando todos os possíveis caminhos que o programa pode seguir, como descrito na seção anterior, para descobrir se algum deles é inseguro. Considerando ainda o exemplo da Figura 1, no momento em que o objeto *o* chama o método *comp*, identificamos que *o* pode ser do tipo da classe *T* ou *U*, e então analisamos o método *comp* das duas classes, criando caminhos no grafo de dependências para os dois métodos. Como identificamos que o método *comp* da classe *U* é inseguro, vamos destacar tal caminho no grafo. Entretanto, uma vez que o tipo do objeto *o* é dinâmico, não sabemos se o método inseguro executará ou não. Para proteger o programa sem perder precisão, partimos para a segunda etapa da nossa checagem dinâmica.

 © 2017 Sociedade Brasileira de Computação

4.2. Instrumentação de código

Para todos os caminhos contaminados reportados no grafo de dependências, identificamos quais deles foram gerados a partir de uma invocação dinâmica de um método, como a invocação do método *comp*. Para cada método invocado, detectamos a qual classe ele pertence e inserimos uma guarda no código Java antes da chamada do método. Uma guarda é definida como:

Definição 4.3 (Guarda) *Uma Guarda é um predicado p(Cl, o, m), que permite a invocação do método m sobre o objeto o, sempre que o mesmo não for instância da classe Cl.*

No exemplo da Figura 1, a guarda inserida antes da chamada do método comp seria p(U,o,comp), significando que o método comp só será invocado sobre o objeto *o* se o objeto não for uma instância da classe U. Se o objeto *o* for uma instância da classe U, uma exceção será lançada informando que o código é inseguro, impedindo que o mesmo seja executado. A instrumentação resultante do exemplo da Figura 1 é mostrada na Figura 2. Dessa forma, essa instrumentação do programa previne a execução de caminhos inseguros, uma vez que uma guarda é inserida antes de uma invocação de um método detectado como inseguro. Tal instrumentação de código é feita diretamente em código Java, ao invés de ser feita no bytecode. A mesma é realizada dessa maneira para que o desenvolvedor que utilizou nossa ferramenta consiga ver e identificar o problema e o local em que a guarda foi inserida. Caso contrário, como no bytecode seria difícil identificar o código inserido, um usuário da ferramenta poderia questionar se o problema realmente existia no código ou se foi, erroneamente, gerado durante a instrumentação do mesmo.

5. Avaliação Experimental

Materializamos nossas ideias em uma ferramenta chamada InspectorJ, implementada no Soot, um framework para análise e otimização de bytecodes Java. Nós configuramos nosso mecanismo de fluxo de informação para detectar *command injection* e ataques por variação de tempo. Os experimentos foram realizados no sistema operacional Windows 7 Professional de 64 bits, com processador Intel Core i3 - 2120, com clock de 3.30GHZ e 8GB de memória RAM. Para validar a efetividade de nossas ideias, utilizamos a nossa ferramenta em uma aplicação para o Flickr, em que tal experimento será discutido como um estudo de caso, além de aplicar a ferramenta em uma série de *benchmarks*, para realizar uma análise qualitativa e uma análise de complexidade.

5.1. FlickrFaves - Estudo de Caso

O Flickr é um aplicativo online de gerenciamento e compartilhamento de fotos e vídeos com o mundo inteiro[3]. Por sua vez, FlickrFaves é uma pequena aplicação para o Flickr que permite que o usuário faça downloads em alta resolução de suas fotos e vídeos favoritos do Flickr. Tal aplicação está disponível em um diretório público no GitHub[4]. FlickrFaves é uma ferramenta multi-plataforma, escrita em Java. Dentre as classes implementadas em tal aplicação, existe uma classe que tem como finalidade detectar o sistema operacional em que a ferramenta está sendo utilizada. Isso permite que a *url* do Flickr

[3]https://www.flickr.com/

[4]https://github.com/magnusvk/FlickrFaves

© 2017 Sociedade Brasileira de Computação

```
1  public class BareBonesBrowserLaunch {
2      public static void openURL(String url) {
3          IBrowser browser = new BrowserFactory().getBrowser();
4          browser.launch(url);
5      }
6  }
7
8  public class BrowserFactory {
9      public IBrowser getBrowser(){
10         String osName = System.getProperty("os.name");
11         IBrowser browser = null;
12         if (osName.startsWith("Mac OS")) {
13             browser = new OSXBrowser();
14         } else if (osName.startsWith("Windows"))
15             browser = new WindowsBrowser();
16         else { // assume Unix or Linux
17             browser = new GenericBrowser();
18         }
19         return browser;
20     }
21 }
22
23 public class WindowsBrowser implements IBrowser {
24     public void launch(String url) throws Exception {
25         Runtime.getRuntime().exec("rundll32 url.dll,
               FileProtocolHandler " + url);
26     }
27 }
```

Figura 4. Código que detecta o sistema operacional em que o FlickrFaves está executando. Isso permite que a url passada como parâmetro seja aberta no browser específico de cada sistema operacional.

recebida pelo usuário seja aberta no *browser* específico de cada sistema operacional, possibilitando assim que a ferramenta seja multi-plataforma. As partes principais do código dessa funcionalidade são apresentadas na Figura 4, em que o código foi refatorado para se tornar mais modularizado, facilitando assim o entendimento da funcionalidade da classe.

A classe *BareBonesBrowserLaunch* possui um método *openURL* que recebe a *url* do Flickr que o usuário deseja abrir. Um objeto *browser* é declarado com o tipo estático *IBrowser*, que é uma interface que possui o método *launch*, implementada por três classes: *OSXBrowser*, *WindowsBrowser* e *GenericBrowser*, cada uma delas representando os sistemas operacionais OSX, Windows e Linux/Unix, respectivamente. O tipo dinâmico do objeto instanciado de *browser* será definido em tempo de execução através do método *getBrowser* da classe *BrowserFactory*. Após a definição de *browser*, o objeto chama o método *launch*, passando a *url* como parâmetro.

Por sua vez, o método *getBrowser* da classe *BrowserFactory* vai detectar, durante a execução do programa, qual o sistema operacional em que o programa está sendo executado, através da função do Java *System.getProperty("os.name")*. Assim, o tipo dinâmico que o objeto *browser* irá assumir dependerá do sistema operacional detectado. Uma vez que o objeto *browser* pode assumir um dos três tipos citados anteriormente, o método *launch* chamado na função *openURL* pode ser de qualquer uma das três classes. A princípio, isso não seria um problema, já que as classes *OSXBrowser* e *GenericBrowser* implementam o método *launch* de maneira segura. Entretanto, detectamos que o método da classe *WindowsBrowser* possui a vulnerabilidade de *command injection*.

Command injection é um ataque cujo objetivo é executar comandos arbitrários no

 © 2017 Sociedade Brasileira de Computação

```
1  public class BareBonesBrowserLaunch {
2      public static void openURL(String url) {
3          IBrowser browser = new BrowserFactory().getBrowser();
4          // BEGIN AUTOMATIC CODE
5          if(browser instanceof WindowsBrowser){
6              throw new java.lang.Exception("Codigo Inseguro");
7          }
8          // END AUTOMATIC CODE
9          browser.launch(url);
10     }
11 }
```

Figura 5. Instrumentação resultante da análise feita no programa da Figura 4 da aplicação FlickrFaves.

sistema operacional do host através de uma aplicação vulnerável. Este ataque é possível quando uma aplicação transfere dados inseguros fornecidos pelo usuário para um shell do sistema. Neste ataque, os comandos do sistema operacional fornecidos pelo invasor geralmente são executados com os privilégios do aplicativo vulnerável. Uma vez que o método recebe a *url* do usuário e concatena a mesma com o parâmetro passado para o método *Runtime.getRuntime().exec()*, a *url* pode conter dados inseguros, tal como um comando, e o mesmo será executado assim que o método *exec* executar. Dessa forma, um usuário mal intencionado poderia utilizar essa aplicação no computador de uma pessoa que utiliza o sistema operacional Windows e fazer um ataque de *command injection* em tal sistema.

Uma vez que só é possível saber dinamicamente qual método *launch* irá executar, é necessário realizar uma análise estática acoplada com uma checagem dinâmica, para permitir que o programa execute quando for seguro e impedir que ele execute quando for inseguro. Utilizamos nossa ferramenta em tal programa, rastreamos todos os possíveis caminhos que poderiam existir e detectamos o caminho inseguro existente quando o sistema operacional utilizado era o Windows. Assim, fomos capazes de instrumentar o código para previnir a execução de tal caminho, inserindo uma guarda antes da chamada de *launch*. O código instrumentado resultante de tal análise é mostrado na Figura 5.

5.2. Análise Qualitativa

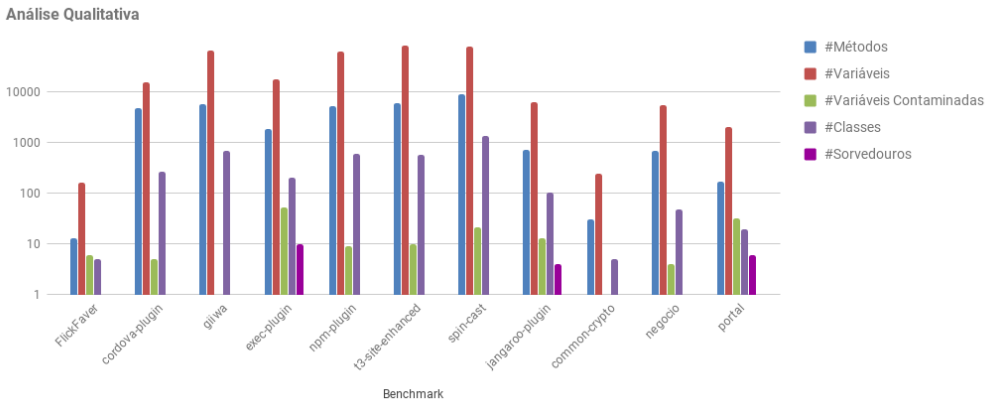


Figura 6. Gráfico resultante da análise qualitativa.

© 2017 Sociedade Brasileira de Computação

A Figura 6 mostra o resultado da aplicação de InspectorJ em 11 *benchmarks* diferentes que contêm vulnerabilidades já reportadas na literatura. Dentre os 11 *benchmarks*, os 9 primeiros são projetos de código aberto do GitHub e os 2 últimos são sistemas da Companhia de Tecnologia do Estado de Minas Gerais (PRODEMGE). InspectorJ foi configurado para detectar *command injection* e ataque por variação de tempo. A ferramenta corretamente reportou pelo menos uma das vulnerabilidades em 9 dos 11 *benchmarks* onde era esperado um problema, e não disparou qualquer aviso para os casos onde não era esperado uma vulnerabilidade. Podemos perceber que uma pequena parte de um programa já é suficiente para torná-lo inseguro: o *t3-site-enhanced*, por exemplo, possui 80390 variáveis, em que apenas 10 são contaminadas e tornam o programa inseguro. Dessa forma, inserir checagens dinâmicas para permitir que programas executem quando são seguros reduz a frequência com que programas são considerados vulneráveis.

5.3. Análise de Complexidade

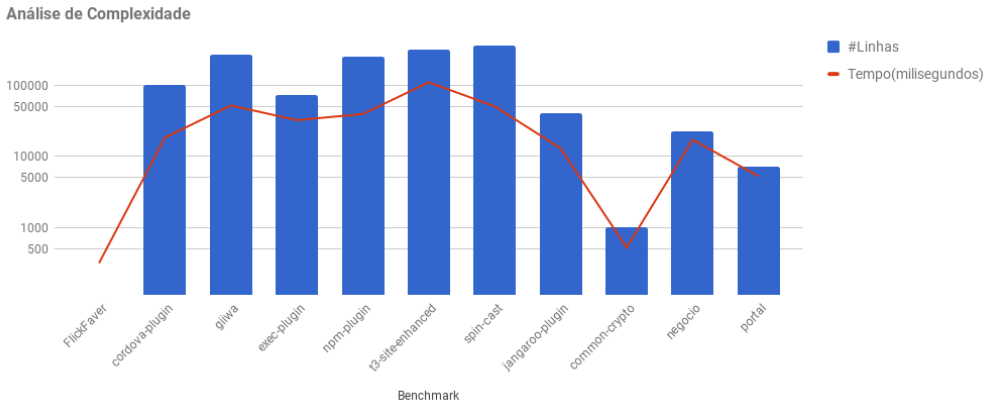


Figura 7. Gráfico resultante da análise de complexidade.

O resultado da análise de complexidade é mostrado na Figura 7. Medimos o tempo da nossa análise em todos os *benchmarks* citados anteriormente e percebemos que InspectorJ roda em tempo linear ao tamanho do programa. Além disso, a checagem dinâmica não aumentou o tempo que os programas demoram para executar, uma vez que a parte do código insegura era pequena em relação ao tamanho do programa inteiro.

6. Trabalhos Relacionados

Existem diversas técnicas de análise de fluxo de informação usadas para proteger programas orientados a objetos. Entretanto, este artigo é único na maneira em que ele combina análises estáticas e dinâmicas com tal propósito. A fim de salientar esse fato, esta seção descreve alguns dos resultados mais notáveis relacionados a técnicas de análise de fluxo de informação. Para tanto, serão discutidos trabalhos que buscaram combinar técnicas de proteção estáticas e dinâmicas. Enfatiza-se que nenhuma das técnicas discutidas a seguir utilizou tal combinação para permitir que mais programas orientados a objetos possam ser certificados como seguros.

Muitos pesquisadores diferentes combinaram análises dinâmicas e estáticas para rastrear o fluxo de informação durante a execução de programas. O propósito dessa

 © 2017 Sociedade Brasileira de Computação

combinação é velocidade; porém, há trabalhos que, como este artigo, também buscaram aumentar a precisão da análise estática. Entretanto, ao contrário do que é feito neste trabalho, previamente a análise estática complementava a informação adquirida pela sua contrapartida dinâmica, e não vice-versa. Por exemplo, Zhang *et al.* [Zhang et al. 2011] propuseram uma ferramenta que instrumenta e executa programas, procurando por potenciais fluxos que podem ser controlados por um adversário. Durante essa espécie de execução monitorada, a ferramenta proposta por Zhang *et al.* também constrói o gráfico de fluxo de controle do programa. Essa estrutura é posteriormente dada a um analisador estático, que executa uma segunda rodada de verificações sobre as partes do programa que não foram atingidas pela análise dinâmica. Vogt *et al.* [Vogt et al. 2007] também usam análise estática para estender os resultados da análise dinâmica. Contudo, eles fazem isso durante a execução do programa. Ou seja, uma vez que a análise dinâmica identifica as regiões do programa que podem ser controladas por dados corrompidos, a ferramenta de Vogt *et al.* executa uma passada linear nessa região, marcando todas as variáveis definidas dentro dela. Caso essas variáveis sejam usadas posteriormente, o ambiente de execução poderá identificá-las como potencialmente contaminadas. Há também trabalhos que mostram como usar a análise dinâmica para eliminar falsos positivos produzidos pelo algoritmo estático. Como um exemplo, Balzarotti *et al.* [Balzarotti et al. 2008] projetaram *Saner*, uma ferramenta que usa análise dinâmica baseada em teste para verificar possíveis vulnerabilidades relatadas pela análise estática.

A análise estática também pode ser usada para melhorar o tempo de execução da técnica dinâmica, conforme mencionado anteriormente. Possivelmente, o trabalho mais influente nesta área pertence a Huang *et al.*; Trata-se de WebSSARI [Huang et al. 2004]. Esta ferramenta usa um sistema de tipos para identificar variáveis de programas que podem ser corrompidas por um usuário malicioso. A partir dessa identificação, WebSSARI instrumenta as funções sensíveis que lêem essas variáveis. Nosso trabalho é diferente do de Huang porque temos objetivos diferentes. Enquanto WebSSARI usa os resultados da análise estática para reduzir o overhead sobre a análise dinâmica, esse tipo de ganho de desempenho não é nosso objetivo. Nossa instrumentação é motivada pela existência de chamadas polimórficas em um programa orientado por objetos, e ocorre somente quando a análise estática falha em analisar o programa totalmente. A instrumentação de Huang *et al.* é o caso oposto: ela é usada como operação fim, sendo a análise estática uma forma de diminuir seu custo de execução. Além disso, Huang *et al.* não discutem como eles lidam com chamadas polimórficas de métodos. Em uma direção totalmente oposta ao trabalho de Huang ou ao nosso trabalho, mas ainda motivados por eficiência, Chebaro *et al.* [Chebaro et al. 2012] usaram análise estática para remover partes trivialmente seguras de um programa, a fim de que o mesmo pudesse ser testado exaustivamente. Em seguida, eles usam análise dinâmica baseada em testes (sobre o programa reduzido) para procurar por vulnerabilidades em suas partes inseguras.

7. Conclusão

Este artigo apresentou uma técnica de análise estática acoplada a checagem dinâmica para ampliar a precisão de técnicas de análise de fluxo de informação. Instrumentamos o código em pontos de chamada que podem invocar métodos inseguros. Tal checagem dinâmica nos permite detectar e descartar caminhos inseguros durante a execução do programa e preservar fluxos seguros de informação. Essa combinação garante a segurança

© 2017 Sociedade Brasileira de Computação

de programas mesmo diante de propriedades difíceis de serem analisadas estaticamente, como por exemplo polimorfismo de subtipagem.

Acredita-se que nosso trabalho seja único na maneira que combina análise estática com checagem dinâmica com o propósito de aumentar a precisão da análise para permitir que mais programas orientados a objetos possam ser certificados como seguros. Como trabalho futuro, pretendemos integrar nossa ferramenta à IDE Eclipse, que hoje é amplamente usada pelos desenvolvedores Java.

Referências

Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., and Vigna, G. (2008). Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *SP*, pages 387–401. IEEE Computer Society.

Bruno Rodrigues, D. A. e. F. M. Q. a. P. (2016). Sparse representation of implicit flows with applications to side-channel detection. In *Compiler Construction*, pages 1–20. ACM.

Chebaro, O., Kosmatov, N., Giorgetti, A., and Julliand, J. (2012). Program slicing enhances a verification technique combining static and dynamic analysis. In *SAC*, pages 1284–1291. ACM.

Clause, J., Li, W., and Orso, A. (2007). Dytan: A generic dynamic taint analysis framework. In *ISSTA*, pages 196–206. ACM.

Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N., and Zadeck, F. K. (1991). Efficiently computing static single assignment form and the control dependence graph. *TOPLAS*, 13(4):451–490.

Denning, D. E. and Denning, P. J. (1977). Certification of programs for secure information flow. *Commun. ACM*, 20:504–513.

Huang, Y., Yu, F., Hang, C., Tsai, C., Lee, D., and Kuo, S. (2004). Securing web application code by static analysis and runtime protection. In *WWW*, pages 40–52. ACM.

Hunt, S. and Sands, D. (2006). On flow-sensitive security types. In *POPL*, pages 79–90. ACM.

Nethercote, N. and Seward, J. (2007). Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI*, pages 89–100. ACM.

Rimsa, A., Quintão Pereira, F. M., and d'Amorim, M. (2011). Tainted flow analysis on e-ssa-form programs. In *CC*, pages 122 – 141, Heidelberg, Alemanha. Springer.

Russo, A. and Sabelfeld, A. (2010). Dynamic vs. static flow-sensitive security analysis. In *CSF*, pages 186–199. IEEE Computer Society.

Vallee-Rai, R., Co, P., Gagnon, E., Hendren, L., Lam, P., and Sundaresan, V. (1999). Soot - a java bytecode optimization framework. In *CASCON*. IBM Press.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Krügel, C., and Vigna, G. (2007). Cross site scripting prevention with dynamic data tainting and static analysis. In *NDSS*.

Zhang, R., Huang, S., Qi, Z., and Guan, H. (2011). Combining static and dynamic analysis to discover software vulnerabilities. In *IMIS*, pages 175–181. IEEE Computer Society.

 © 2017 Sociedade Brasileira de Computação

An operational costs analysis of similarity digest search strategies using approximate matching tools

Vitor Hugo Galhardo Moia, Marco Aurélio Amaral Henriques

[1]School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas, SP, Brasil 13083-852

[vhgmoia,marco]@dca.fee.unicamp.br

Abstract. *Approximate matching functions are suitable tools for forensic investigators to detect similarity between two digital objects. With the rapid increase in data storage capacity, these functions appear as candidates to perform Known File Filtering (KFF) efficiently, separating relevant from irrelevant information. However, comparing sets of approximate matching digests can be overwhelming, since the usual approach is by brute force (all-against-all). In this paper, we evaluate some strategies to better perform KFF using approximate matching tools. A detailed analysis of their operational costs when performing over large data sets is done. Our results show significant improvements over brute force and how the strategies scale for different database sizes.*

1. Introduction

With the vast amount of data available nowadays, stored in a variety of different devices (computers, smartphones, tablets, etc.), forensic examiners face a new challenge: Process all these data in a relatively short time. Even ordinary investigations are becoming a problem since users tend to acquire multiple devices with high storage capabilities. The increased usage of cloud storage services is also an issue. Now, users have more space to store content and, in most cases, data is replicated among all devices (redundancy). Also, data is increasing in size due to high-resolution picture and videos, more sophisticated software and operating systems etc., requiring more time to be processed.

One of the methods used to handle this huge amount of data efficiently is the Known File Filter (KFF), a way to separate relevant from irrelevant information by the use of white or black lists of reference data. Forensic investigators can either eliminate data from the analysis, such as operating system files, known software, and other inoffensive objects (white list), or separate information that is considered illegal or suspicious (black list). In both cases, the examiner holds databases of known interest objects which are used in the filtering processes [Breitinger and Baier 2012].

Cryptographic hash functions (SHA-1, SHA-2, etc.) appear as suitable candidates to perform the KFF in a very efficient manner. To cooperate with forensic experts, NIST (National Institute of Standards and Technology) provided a white list database (NSRL - National Software Reference Library [NIST 2016]) in the form of hashes that can be used to perform this filtering process. However, some limitations arise with the use of such functions. By design, hashes of two objects with the same content will be the same; but changing a single bit in one of the objects, make its hash completely different. This way, only binary answers are provided with hashes: The objects are identical or not. This

 © 2017 Sociedade Brasileira de Computação

limitation force examiners to keep databases with hashes of every possible version of the interest object, since attackers could make random changes on objects to bypass detection. Also, some objects get new versions as software or operating systems are updated, and keeping track of each object changing is an impossible task.

To mitigate the limitations presented above, researchers have come up with new functions capable of detecting similarity between objects, called approximate matching or similarity hashes. They have the same idea of cryptographic hashes: The creation of small and compact representations for data (digest). Using a generation function, digests are created in a way that similar objects will produce similar representations, and by the utilization of a particular digest comparison function, a score related to the amount of the content shared between the corresponding objects is produced.

The downside of using approximate matching functions are the high costs associated with both processes: Digest generation and comparison. This fact becomes evident when comparing them to cryptographic hashes, although they have distinctive goals. However, hashes are the baseline, and new approximate matching functions always try to achieve similar time costs as traditional hashes.

Investigations involving the use approximate matching functions to perform KFF are very time-consuming. In the first stage, called preparation phase, digests are created and stored for every object in the reference list of interest objects. Then, in the operational phase, the target system under analysis has digests created for each object in this media and compared to the reference list. In case a certain pre-defined threshold is achieved, a match is found, and the corresponding object is separated for further analysis. However, the major bottleneck on this task is not the digest generation process, but the comparison one. Usually, the latter is done by brute force, where every digest in the target system is compared to every digest in the reference list. Since these two data sets tend to be very large, the whole comparison process could take days or even weeks to be accomplished.

In this work, we evaluate alternatives to the expensive brute force approach. These new methods referred as similarity digest search strategies, aim at performing digest lookups very efficiently for KFF. This paper extends the work done in [Moia and Henriques 2017], where the authors performed the analysis of three different strategies (F2S2, MRSH-NET, and BF-based Tree) regarding three main aspects: precision, memory requirement, and lookup complexity. We show how the same strategies carry out the search and develop equations to estimate their operational costs, allowing a more precise time comparison. Our results indicate that these strategies are a better choice than the simple brute force, and can reduce significantly the time consumed in KFF investigations. We also show how they scale when performing over different data set sizes. To the best of our knowledge, this is the first work to compare these strategies regarding operational costs.

2. Approximate matching tools and similarity digest search

According to NIST [Breitinger et al. 2014b], approximate matching functions are designed to identify the similarity between objects, in the level of resemblance (objects that resemble each other) or containment (objects contained in other, e.g., images inside a document). Also, these sort of functions can operate in three different levels when creating the digests: bytewise, syntactic or semantic [Breitinger et al. 2014b]. We restrict

 © 2017 Sociedade Brasileira de Computação

our research to only those functions working at the byte level since they are more time efficient and do not depend on the object format; moreover, they do not try to interpret the object either to consider any data structure.

Approximate matching tools usually have two primary functions: Digest generation and comparison. In the first one, a digest is created for an object as a small and compact representation of it. Its size can be fixed or not, depending on the chosen tool. For instance, ssdeep produces representations of at most 96 bytes (no metadata included), while sdhash digests are proportional to the object size ($\approx 2.6\%$). The second function of such tools is intended to compare two objects, where the output is a confidence measure about their similarity, usually in the scale of 0 (dissimilar) to 100 (very similar), but this changes according to the tool.

The applications of approximate matching are vast, encompassing the detection of new versions of documents, libraries or software, embedded objects, code reuse, malware clustering, among others. A broader view on approximate matching tools can be found in [Harichandran et al. 2016, Martínez et al. 2014].

The process of comparing two data sets of digests to find similar ones is referred to similarity digest search. The forensics examiner needs to check if any of the objects in the media under analysis is similar to any reference list object. Usually, this process is done efficiently using ordinary databases, but in such cases, the search aims at finding exact matches. Looking for objects that resemble each other or are contained one in another is a much harder task, and cannot be done using traditional ways but only by the simple brute force approach. The similarity detection is done by a particular function of the approximate matching tool chosen, which needs to be taken into consideration in the design of an efficient structure to store digests and allow similarity lookups. The strategies evaluated in this paper (F2S2, MRSH-NET, and BF-based Tree) follow this idea and are most specific for a particular kind of tool.

3. Strategies to perform known file filtering

The most time-consuming part of an investigation involving the KFF method and approximate matching functions is the comparison step, usually performed by the brute force approach. However, to avoid this expensive solution to perform comparisons, researchers came up with strategies to better find similar objects on large data sets. In this section, we present some of these strategies and show how they perform the similarity digest search.

3.1. Brute force

The simple method of brute force consists in comparing all digests from the reference list (usually pre-computed) with all digests generated for the target system. The best matching for each object is selected in case the value returned by the comparison function is above a certain threshold. The comparison using traditional hash functions (SHA-1, SHA-2, etc.) is fast since it only requires to verify whether two fixed size strings (digests) are equal or not. However, when using approximate matching tools, this operation becomes more expensive. Checking the similarity of two digests, in this case, requires interpreting the digests, using a special function designed for the chosen particular tool.

© 2017 Sociedade Brasileira de Computação

3.2. F2S2 strategy

The Fast Forensic Similarity Search (F2S2) strategy, proposed by Winter, C. et al. [Winter et al. 2013], is an alternative to brute force when performing the KFF method. This approach uses ssdeep as its approximate matching tool, but it is not limited to it. It requires that similar objects produce similar digests, a requirement fulfilled by ssdeep when looking at its comparison function: edit distance (number of operations to transform one string into another). Roughly speaking, ssdeep divides the object input into variable size pieces and codifies them into two digests of at most 96 base64 characters, in a way that similar objects will produce similar digests when compared [Kornblum 2006].

F2S2 is an index strategy which uses n-grams (n consecutive bytes) extracted from the digests and store them in a chained hash table. The n-grams are extracted using a fixed window size that moves through the digest byte-to-byte. They are used as lookup keys in the search for similarity, pointing out possible candidates for being similar to the queried object. At the end of the search, a comparison using ssdeep is made to only those objects sharing the same n-grams as the queried item. According to Winter's experiments [Winter et al. 2013], a speedup above 2000 times was achieved in comparison to the brute force approach using ssdeep.

The hash table used by F2S2 is composed of a central array with buckets of variable size, which means that more than one n-gram can be stored in each bucket (multiple entries), in a chaining fashion. Each n-gram is made of two parts: e-key and bucket address. The former identifies the sequence, while the latter is used to find the position in the hash table where it will be stored by a mapping function. Also, an ID is assigned to each digest meant to be inserted in the table and stored along with the n-gram in the bucket, as a way of linking it with the digest and allowing later identification [Winter et al. 2013].

The first step before using F2S2 is to build the index structure. By using ssdeep, the examiner creates digests for each reference list object and then inserts it in the index. The first step of this process is to assign IDs to each digest, followed by the n-grams extraction. Then, all n-grams are inserted in the hash table along with their corresponding ID. Here, collisions in the hash table buckets are resolved by using a chaining technique, where each bucket can store multiple n-grams, one followed by the other (linked list).

3.3. MRSH-NET: A Bloom filter approach

Another strategy for performing the KFF approach is the MRSH-NET, proposed by Breitinger, F. et al. [Breitinger et al. 2014a]. The motivation behind this strategy is to improve the lookup procedure of sdhash and other approximate matching tools which encode their digests using Bloom filters (BF). The F2S2 strategy presented previously cannot work with this sort of technology, since there is no effective way to index bloom filters.

The main idea of MRSH-NET is to create a single, huge bloom filter to represent all reference list objects instead of one or more BFs separately for each item. However, due to the characteristics of BFs, MRSH-NET is limited to only membership queries. It can answer if object A is contained in the set or not, but it can not point out what is/are the one(s) sharing similarity with it. According to Breitinger, F. et al. [Breitinger et al. 2014a], this could be enough for a blacklist case. However, we argue that it could generate lots of false positives that could not be confirmed since the strategy does not tell us which were the similar objects.

 © 2017 Sociedade Brasileira de Computação

To use MRSH-NET for KFF, the examiner needs to create the bloom filter structure and then insert all reference list objects on it. This process is done by first using the chosen approximate matching tool (sdhash in our work) to extract the features from each object in the list and inserting them into the single, huge Bloom filter.

3.4. BF-based tree

A possible solution to overcome MRSH-NET limitation regarding membership queries and yet to work with sdhash and related tools is presented by Breitinger, F. et al. [Breitinger et al. 2014c]. This new similarity digest search strategy is based on the divide and conquer paradigm, where a Bloom filter-based tree structure is used to store all reference list objects, allowing similarity detection and also object identification. However, this approach only exists in theory and lacks a working prototype. Also, it demands an enormous amount of memory to operate in comparison to other strategies [Moia and Henriques 2017].

Holding a reference list of objects of interest S with i items, the examiner must build the BF tree structure and fill it up with all the elements. For this purpose, a single bloom filter is created as the root node, and all items are inserted into the filter, by first using the chosen approximate matching tool to extract the object's features and then inserting them in the filter. The next step involves dividing the set S in half, where each part has its features extracted and added in new Bloom filters, child of the root node. These two parts are broken in half again, and four new filters are created with the features of the items belonging to each subset. Each filter is a child of the previous node. The process repeats until the division of the set results in one object. Finally, the parameters *FI* (File Identifier) and *FIC* (File Identifier Counter) are created at the end of the tree, as leaves. The former variable is used to link the corresponding BF to a database containing the digest of the item, while the latter is initially set to 0 (zero) and incremented when the corresponding *FI* is traced in a lookup procedure.

Lookups are expected to be very fast using this strategy since only a subset of nodes needs to be checked. The first step requires the feature extraction of the queried item, followed by a lookup procedure which is done for each extracted feature, starting from the root of the tree. In the case of a non-match in the root, we know for sure that this feature is not in the tree structure and we can proceed to the next object in the target system. Otherwise, the search continues through the rest of the tree until it reaches a leaf. The *FIC* of the corresponding leaf is incremented. After comparing all features from the queried item, the *FI* of the highest *FIC* is selected and compared to a threshold t, which is the minimum number of consecutive features that need to be found in the filter. When the value is higher than the threshold, the item is said to belong to the tree. Finally, a comparison of the selected item and the queried one is done in the digest level, using the chosen approximate matching tool with the purpose of verifying similarity. According to Breitinger, F. et al. [Breitinger et al. 2014c], most of the comparisons yield a non-match, especially for blacklist cases, leading the search to stop in the root node. They argue that the number of bad or illegal objects are usually much smaller than the total number of objects in the target system, making the BF-based tree strategy very time efficient.

© 2017 Sociedade Brasileira de Computação

4. Steps in a similarity digest search procedure

In this section, we evaluate the necessary steps in the operational phase of each strategy and present equations that we developed to estimate the time needed for each of them. We performed this analysis because most strategies do not have their source codes or a compiled version available to calculate their costs and compare them.

4.1. Brute force

The procedure involving the brute force can be summarized in the following steps:

1. For each item in the target system, perform:
   (a) digest generation;
   (b) For each item in the reference list, perform:
      i. digests comparison.
   (c) Return the digest with higher value if above a threshold *t*.

The steps aforementioned are part of the operation phase of a simple brute force approach. We can estimate the time to perform such task using Eq.1.

$$T_{op} = i \cdot (T_{digCalc} + (r \cdot T_{compFunc})) \tag{1}$$

Here i is the number of objects in the target system, $T_{digCalc}$ is the time for calculating a single digest using the chosen approximate matching tool (ssdeep, sdhash, etc.) or hash function (SHA-1, SHA-2, etc.), r the number of objects in the reference list and $T_{compFunc}$ is the time to compare two entries using the same tool.

Brute force can be used with any similarity tool. In our work, we chose ssdeep [Kornblum 2006] and sdhash [Roussev 2010] to perform the brute force approach and hence compare them with other strategies regarding time performance. We chose SHA-1 hash function as a benchmark since most approximate matching functions aim at achieving times close to it. We emphasize that SHA-1 vulnerabilities regarding collisions are not an issue here, as this hash function is not being used with a security role. Even simpler (and also compromised) functions as MD-5 would be useful here.

4.2. F2S2

Upon an investigation process where the forensic examiner wants to perform the KFF approach in a target system, the following steps must be done using F2S2:

1. Load the index structure into main memory;
2. For each item in the target system, perform:
   (a) ssdeep digest generation;
   (b) n-gram extraction;
   (c) n-gram lookup (*L* digests that share the same n-grams as the ones of the queried item will be selected as candidates).
   (d) ssdeep comparison of the queried item with the *L* selected digest.
   (e) return the digests with value above a threshold *t*.

 © 2017 Sociedade Brasileira de Computação

To estimate the time to perform the operational phase of F2S2, we can use Eq.2.

$$T_{op} = T_{indexLoad} + i \cdot (T_{genSsdeep} + T_{ngramExtr} + T_{ngramLookup} + (L \cdot T_{compSsdeep})), \quad (2)$$

where $T_{indexLoad}$ represents the time to load the index into memory (step 1), i the number of objects in the target system, $T_{genSsdeep}$ the time to compute a single ssdeep digest (step 2.a), and $T_{ngramExtr}$ the time to extract all n-grams from the corresponding object (step 2.b). $T_{ngramLookup}$ refers to the time to perform a lookup procedure in the index (step 2.c), calculated by Eq. 3. L represents the number of candidates sharing the same n-grams as the queried item, returned by the lookup process, and $T_{compSsdeep}$ denotes the time to calculate the Edit Distance for two digests (ssdeep comparison function)(step 2.d).

To calculate the time to perform the lookup procedure, we can use Eq. 3.

$$T_{ngramLookup} = g \cdot (T_{hashI} + (T_{compStrE} \cdot b)). \quad (3)$$

Here g is the number of n-grams extracted from the digest ($g = l_{dig} - n + 1$, where l_{dig} is the digest size (bytes) and n the n-gram sequence size), T_{hashI} the time to hash a string (n-gram index), $T_{compStrE}$ the time to compare two strings (n-gram e-key) and b the average number of different n-grams in each bucket (section 5.2 shows how to compute it).

4.3. MRSH-NET

The operational process using MRSH-NET involves the following steps:

1. Load the Bloom filter into main memory;
2. For each item in the target system, perform:
   (a) feature extraction (with sdhash), resulting in z features;
   (b) z lookups.

To estimate the time to perform this task, we first need to calculate the number of features (z) presented in the target system (average). To this end, we can use Eq. 4, derived from Breitinger, F. et al [Breitinger et al. 2014c] statement: *"sdhash maps 160 features into a Bloom filter for every approximately 10 KiB of input file"*.

$$z = (\mu \cdot 2^{20} \cdot 160)/(10 \cdot 2^{10}) = 2^{14} \cdot \mu, \quad (4)$$

where μ is the size of all objects in the reference list (MiB) and both factor, 2^{20} and 2^{10}, are used to change from MiB and KiB to bytes, respectively.

Then, we can use Eq. 5 to estimate the required time of MRSH-NET, as follows:

$$T_{op} = T_{bfLoad} + i \cdot (T_{featureExtr} + (z \cdot T_{bfLookup})). \quad (5)$$

where T_{bfLoad} is the time to load the bloom filter data structure into memory (step 1), i the number of objects in the target system, $T_{featureExtr}$ the time to extract the features from an object (step 2.a), and z the number of features extracted from it (Eq. 4). The $T_{bfLookup}$ is the time to perform a lookup procedure in the filter (step 2.b), calculated by eq. 6.

$$T_{bfLookup} = T_{hashF} + (k \cdot T_{compStrF}) \quad (6)$$

Here T_{hashF} denotes the time to hash each feature of β bytes and k the number of sub-hashes. MRSH-NET inserts a feature in the bloom filter by first hashing it and breaking the hash into k parts (sub-hashes). The resulting pieces are used to set the bloom filter. $T_{compStrF}$ is the time to compare two strings of F bytes each (feature size divided by k).

© 2017 Sociedade Brasileira de Computação

4.4. BF-based tree

We can sum up the required steps using BF-based tree by the following ones:

1. Load the Bloom filter-based tree structure into main memory;
2. For each item in the target system, perform:
   (a) feature extraction (with sdhash), resulting in z features;
   (b) z lookups in the tree.
   (c) If any object feature *FIC* is higher than the threshold *t*,
      i. compare the corresponding *FI* digest with the queried one (with sdhash).

To estimate the operational phase time of BF-based tree, we can use Eq.7.

$$T_{op} = T_{bfTreeLoad} + i \cdot (T_{featureExtr} + (z \cdot (T_{bfLookup} \cdot h)) + T_{compFunc}). \quad (7)$$

where $T_{bfTreeLoad}$ corresponds to the time to load the bloom filter-based tree structure into memory (step 1), i is the number of objects in the target system, $T_{featureExtr}$ the time to extract the features from a single object (2.a), z the number of features extracted from the object (calculated by Eq. 4), and $T_{bfLookup}$ the time to lookup each feature in the tree (Eq. 6) (step 2.b). The h parameter denotes the number of steps required to reach an object in the tree (see section 5.3), while $T_{compFunc}$ is the time to compare two digests using the chosen approximate matching tool (step 2.c).

5. Strategies operational costs

The similarity digest search strategies costs were estimated using the formulas developed in the previous section. We have designed pieces of code simulating the operations required by the strategies and then measured the necessary time for performing them. However, some parameters required a deeper analysis to understand their behavior and to estimate their values. To this end, we created a database of real data objects to perform the measures. In this section, we discuss details about this database, the singularities of some particular parameters, and finally, our results.

In this work, the analysis is focused on one of the most important parameters in forensics investigations: The database size. Other variations in the strategies parameters to find their best-operating conditions will be addressed in future works.

5.1. Test database details

To better understand the behavior of some operations and then estimate their values more precisely, we created a database from real data. This set encompasses over one million objects extracted from two Linux operating systems (Elementary OS client - Ubuntu 16.04-based - and Ubuntu 16.04 server), Microsoft Windows 10 Home, and also from personal data, which includes photos, documents, videos, applications, etc. Our database has 1,256,356 objects, corresponding to about 233.32 GiB of data. There were several typical applications installed in each operating systems, including Latex, LibreOffice, Microsoft Office 2013, Foxit Reader, NetBeans IDE, Internet Explorer, Google Chrome, Firefox, and default operating system applications. The type of the objects varies, including pdf, jpg, png, bmp, txt, doc, docx, odf, mkv, avi, py, mp3, wma, html, jar, rar, c, bin, among others. Since this work focus on the best performance of the approximate matching tools in their best conditions and our experience shows that ssdeep do not produce reliable results with large objects, we limited the object size in our database to 200 MiB. Future works will address scenarios using larger objects.

 © 2017 Sociedade Brasileira de Computação

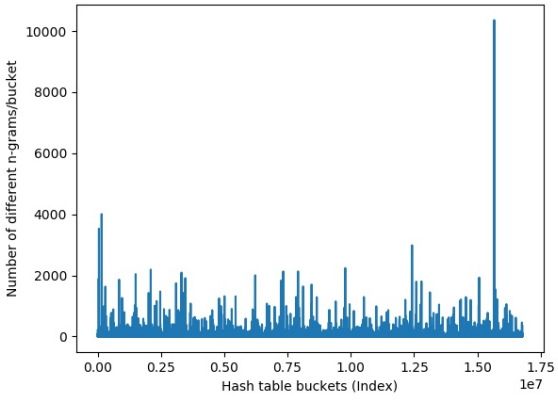


Figure 1. Different n-grams per bucket

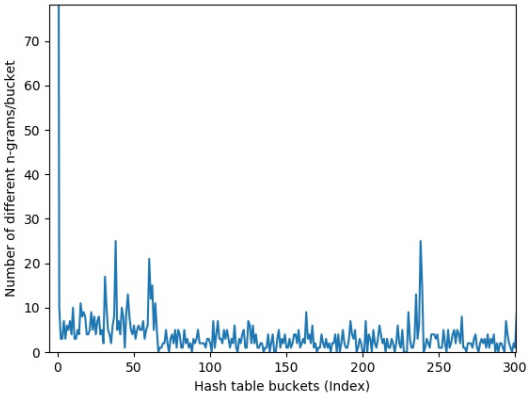


Figure 2. Number of different n-grams per bucket (amplified): index 0 to 300.

5.2. The F2S2 case analysis: Using different database sizes

To estimate some parameters of the F2S2 strategy, we implemented a simplified version of this approach, aiming to observe how data is spread over the hash table. Then, we estimated the values of variables b (number of different n-grams per bucket) and L (number of objects sharing the same n-gram as the queried item), necessary to the calculus of F2S2 operational phase. To this end, we generated ssdeep hashes for every object in our database (encompassing all four data sets) and inserted them in the index structure. The chosen hash table size was 2^{24} (4 6-bit base64 characters from the n-gram).

After inserting all n-grams obtained from our database objects in the index structure, a total of 75,549,716 n-grams were counted. However, 2,218,267 of 16,777,216 (13.22%) buckets were still empty, which means that lots of buckets contained multiple entries. Fig 1 shows the distribution of the number of different n-grams per bucket (b), while Fig. 2 is an amplified version of it, presenting only the b value for the 300 first buckets. Estimating b can be done using Eq. 8, as follows:

$$b = d_{ngrams}/n_{buckets}, \tag{8}$$

where d_{ngrams} is the number of different n-grams inserted in the hash table and $n_{buckets}$ the hash table size (number of buckets). The statistic information about the distribution of b in the table follows: standard deviation = 5.22, median = 2, and mode = 2. To find the average number of different n-grams in each bucket, we applied Eq. 8 for the data inserted in the F2S2 structure, obtaining a result of 2.27. Although we may find a few buckets with lots of n-grams, a significant portion of the table is empty and will lead to empty buckets, as we can see by Fig. 2. There is a peak in one bucket with a significant number of n-grams occurrences and others, but as expressed by the mode and median got from the analysis, most buckets have two or fewer elements. This fact led us to choose the average of the number of different n-grams as the b value (Eq. 8) since this seems the better choice when comparing it to the other statistic components got from the analysis.

Concerning to the number of objects sharing the same n-gram L, we have used

© 2017 Sociedade Brasileira de Computação

Eq. 9 to calculate this value.

$$L = n_{obj}/d_{ngrams}. \tag{9}$$

Here, the new variable n_{obj} denotes the number of objects in our database. We cannot use the average as we did in the previous case because digests may share multiple n-grams and we are only interested in finding the number of different digests sharing at least one n-gram with the queried item, not the number of similar n-grams. When we apply Eq. 9 to the data of the F2S2 hash table, we get a result of 0.033, which express our expectation for the average number of different objects sharing the same n-gram.

The values calculated so far are concerning to our data set only. Other sets of similar or different sizes may present different values for b and L since these parameters depend exclusively on the data being handled, which could have a different n-gram distribution. To extend our results to other sets, one needs to compute the b and L values for them. To this end, equations 8 and 9 can be applied. However, if by on hand we can define our hash table size ($n_{buckets}$) and count the number of objects in our database (n_{obj}) quickly, on the other hand, it is hard to know the number of different n-grams (d_{ngrams}) without inserting all ssdeep digests in the hash table and counting them.

To estimate the value of different n-grams (d_{ngrams}) for a data set with different type and size, required to the calculus of the parameters b and L, we need to find an expression that give us such value based on the total number of n-gram in this set. First, we consider the different data sources that forms the database (Linux client, Linux server, Windows 10, and personal data) to simulate different systems and get a more general idea on how the n-grams are spread across them. Then, we insert each set separately in the F2S2 index structure and count the number of different n-grams. We also use the data got from our first analysis where the entire database (all former sets together) was inserted in the structure since it can represent a different and larger set.

To find the expression that gives us the value of d_{ngrams}, we use regression analysis techniques. This way, we can determine the relationship between the values of different n-grams (unknown) and the total number of n-grams in the database (known). We chose the least squares method to this end. Applying this technique to our results (Tab. 1), we came up with the following expression:

$$d_{ngrams} = 2237231.04 + (0.4816 \cdot n_{ngrams}), \tag{10}$$

where n_{ngrams} is the number of n-grams of the database. We can obtain this number by multiplying the number of objects for the average number of n-grams in a single digest.

Table 1. Number of different n-grams in each data source

Source	Total number of n-grams	Number of different n-grams
Linux client	22,937,595	12,828,701
Linux server	28,071,532	18,405,835
Microsoft Windows	11,376,395	7,357,119
Personal data	13,164,194	7,316,768
All sources	75,549,716	38,053,476

The expression above allows us to estimate the number of different n-grams in any database size and hence calculate both b and L values.

 © 2017 Sociedade Brasileira de Computação

5.3. BF-based tree case analysis: Average steps in a search

Estimating the time needed for the BF-based tree strategy demands the knowledge of the number of times we will need to go through the tree structure to identify a given element. In a lookup procedure, there will be cases where the search stops in the root when the given element is not present in the structure, but in others, we may have to go through the entire structure to find it. Knowing the number of steps performed by element (h) will allow us to estimate the operational cost of the BF-based tree strategy (Eq. 7). To this end, one can develop an algorithm to count the average number of steps considering two scenarios: when the input is present in the tree and when it is not. In the first case, we go through the tree structure checking each node (bloom filter) for the presence of the given object until we find it, stopping the procedure in a leaf. For the second one, the search ends in the root tree, but there is also the possibility of occurring false positives, leading the process to go through the tree until all nodes of any level return a negative result. In the end, a weighted average between the number of steps found in each scenario and the number of elements must be returned, where the percentage of the two events occur are used as weight.

5.4. Parameters definitions and measurements

Using the proposed formulas we can estimate the costs for performing the searches with different database sizes. The values adopted in our experiments are presented in Table 2. We performed the tests using the following machine: Elementary OS 0.4.1 Loki 64-bit (built on Ubuntu 16.04.2 LTS), i7-5500U CPU @2.40GHz processor, 8 GB of memory, and NVIDIA GeForce 920M. We measured the time for each operation using the clock library from the C language, except for ssdeep and sdhash times (generation and comparison functions) which were measured using the time command (sys + user times) available on Linux distributions, since we used the compiled version of these tools. The times to compute the ssdeep, sdhash, and SHA-1 (both hashes generation and comparison) were calculated over the average object size, presented in the table. We repeated all experiments 20 times and took the average, taking care of clearing the cache each time to prevent previous results influencing new ones.

Before calculating the operational costs, we determined the strategies structure size. To this end, we used the formulas and parameters presented in another work [Moia and Henriques 2017], adapting only the database and average object sizes. We measured the time to load an object of 1 GiB from disk to memory (buffer of s_{buf} bytes) and adjusted this value according to the strategies structure size to simulate the loading.

5.5. Operational costs evaluation

We have measured the time for the three presented strategies (F2S2, MRSH-NET, and BF-based Tree) and also for the brute force one when using SHA-1, ssdeep and sdhash. In our experiments, we first considered the database presented in section 5.1 as our reference list and then measured the time for performing over it using different target system sizes, varying them from 1 GiB to 10 TiB. It is important to mention that our database encompasses objects from different fonts gathered in a single set. Using the equations from Section 3 along with the values of section 5.4, we can estimate the strategies operational costs, shown in Fig. 3 (a).

© 2017 Sociedade Brasileira de Computação

Table 2. Similarity digest search strategies experiments - Parameters

Parameter	Description	Value	STD
s_{obj}	Average object size	195 (KiB)	-
$s_{strategy}$	Strategies' structure size	see section 5.4	-
s_{buf}	Buffer size	4096 (bytes)	-
T_{read}	Time to read 1 GiB from disk to memory	429.7565 (ms)	8.1452 (ms)
T_{genSH}	SHA-1 digest generation	0.6581 (ms)	0.0949 (ms)
T_{compSH}	SHA-1 comparison function	0.0102 (ms)	0.0037 (ms)
s_{ngram}	n-gram size	7 (bytes)	-
s_{index}	index size	4 (bytes)	-
s_{ekey}	e-key size	3 (bytes)	-
T_{genSS}	ssdeep digest generation	5.6000 (ms)	2.3324 (ms)
T_{compSS}	ssdeep comparison function	1.0000 (ms)	1.7320 (ms)
$T_{indexLoad}$	Time to load the index structure	$s_{strategy} \cdot T_{read}$ (ms)	-
$T_{ngramExtr}$	Time to extract n-grams from ssdeep digest	0.0207 (ms)	0.0062 (ms)
b	N-grams in each bucket	see section 5.2	-
L	Candidates sharing the same n-gram	see section 5.2	-
T_{hashI}	Time to hash s_{index} bytes	0.0813 (ms)	0.0115
$T_{compStrE}$	Time to compare two strings (s_{ekey} bytes)	0.0010 (ms)	0.0001 (ms)
T_{genSD}	sdhash digest generation	19.2000 (ms)	5.3066 (ms)
T_{compSD}	sdhash comparison function	19.4000 (ms)	4.7791 (ms)
T_{bfLoad}	Time to load the Bloom filter structure	$s_{strategy} \cdot T_{read}$ (ms)	-
$T_{featureExtr}$	Time to extract features from an object	12.2975 (ms)	4.4667 (ms)
β	Feature size	64 (bytes)	-
T_{hashF}	Time to hash a feature of β bytes	0.0929 (ms)	0.0295 (ms)
$T_{compStrF}$	Time to compare two strings (β bytes)	0.0010 (ms)	0.0001 (ms)
k	Number of hash functions for the BF	5	-
$T_{bfTreeLoad}$	Time to load the BF tree structure	$s_{strategy} \cdot T_{read}$ (ms)	-
h	Average steps in a lookup procedure	see section 5.3	-
x	Degree of the tree	2	-

According to our results, the strategies presented a linear grow as the target system size increased. As expected, the brute force approach had the worst results, with sdhash as the most expensive one, followed by ssdeep and SHA-1. The latter one had similar results as the BF-based tree but was a bit more costly. The best strategy regarding the operational time was F2S2, presenting, on average, a speedup of 1,933,723.39, 99,676.83, and 996.82 times compared to the brute force approaches sdhash, ssdeep, and SHA-1, respectively, and 778.89 and 25.24 times better than BF-tree and MRSH-NET strategies, respectively.

Fig. 3 (b) shows how the strategies perform over a reference list size variation (10 GiB to 250 GiB) and using a fixed target system size database (1 TiB). We limited our experiments to this particular range since it is the one covered by our data set. Estimating the behavior of b and L parameters in the F2S2 equation for database sizes far beyond ours could lead to wrong results once the data distribution in such cases could be very different. For larger database sizes, additional studies are required to estimate these variables and hence the strategies operational costs.

 © 2017 Sociedade Brasileira de Computação

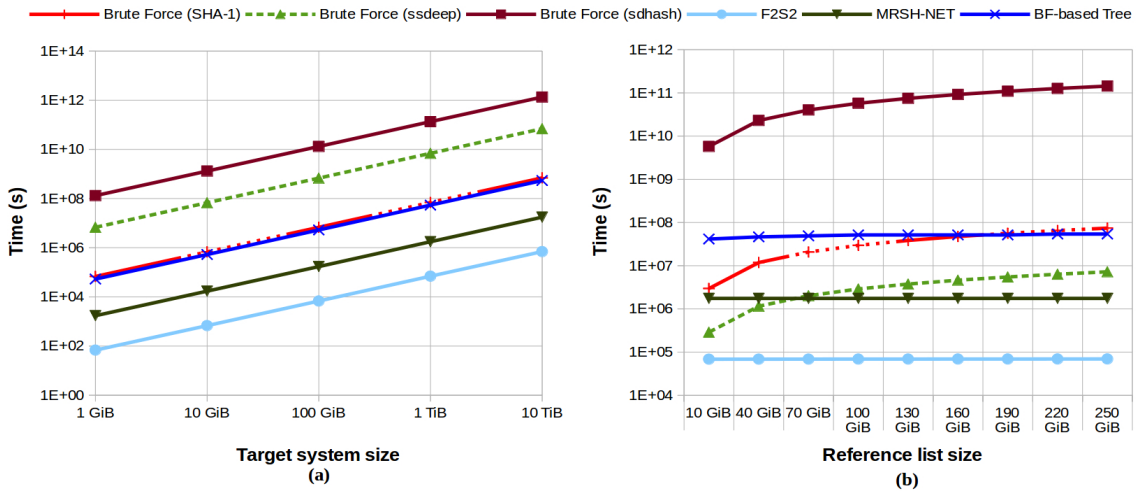


Figure 3. Strategies operational costs:
(a) Target system size variation, considering a fixed size ref. list (233.32 GiB)
(b) Ref. list size variation, considering a fixed size target system (1 TiB)

Again, our results pointed out that sdhash brute force is the most time-consuming approach and, on average, F2S2 is the best one. However, despite the linear behavior of the brute force method, the other strategies presented a different time variation as the reference list size increased. When performing queries over these new database sizes, MRSH-NET showed constant time as expected due to the lookup procedure time be independent of the structure size. The time on the BF-based tree strategy was not constant but increased slowly due to the tree structure getting larger as the reference list expanded (number of elements grow), elevating the average number of steps in a lookup procedure and hence the time to go through the tree. The F2S2 strategy time had a very small increase due to the number of similar n-grams increase. For databases ranging from 10 GiB to 250 GiB, the F2S2 time increase was only 1.24% of the initial value.

Our findings showed the prohibitive costs of the brute force method when performing over large database sizes. For instance, an investigation of a 1 TiB target system using a reference list of 233.32 GiB and sdhash brute force approach could take more than 4 thousand years to be accomplished (around 6,917,958,887,350 comparisons between the data sets are necessary). The same search using the strategies MRSH-NET and the BF-based tree could take only 70 and 2145 days, respectively. Using the ssdeep brute force, the same process would require around 754 years, while F2S2 would only demand no more than 20 hours. It is important to mention that this enormous amount of time is due to the hardware used to perform the experiments. In a working station with powerful hardware and using parallelism techniques, these times are expected to decrease, but the relative differences found among the strategies will remain.

Another important fact to mention is that even though F2S2 presents higher lookup complexity than MRSH-NET and BF-based tree [Moia and Henriques 2017], it performed better in our experiments when comparing large data sets. As expected, MRSH-NET was better than the BF-based tree.

© 2017 Sociedade Brasileira de Computação

6. Conclusion

Approximate matching functions appear as candidates to speedup the process of digital forensic investigations and yet being accurate in the search for objects of interest in medias with high capacity. However, traditional search procedures are based on the inefficient brute force method, which is too time-consuming and even prohibitive for larger data sets. In this paper, we evaluated similarity digest search strategies aiming at reducing this overwhelming cost. A detailed analysis of the operational costs of these approaches was presented, showing significant improvements over the simple brute force ones. Our results point out how the strategies scale with different database sizes and show the ones that performed best. Future studies encompass extending our analysis to other similarity digest search strategies, along with the estimation of the preparation phase costs and variations of the strategies parameters to find the best-operating conditions of each approach. We also plan to find new methods to estimate the parameters b and L of F2S2 to scale with any database size as well as developing a new strategy to overcome current limitations.

References

Breitinger, F. and Baier, H. (2012). *Performance Issues About Context-Triggered Piecewise Hashing*, pages 141–155. Springer Berlin Heidelberg, Berlin, Heidelberg.

Breitinger, F., Baier, H., and White, D. (2014a). On the database lookup problem of approximate matching. *Digital Investigation*, 11:S1–S9.

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., and White, D. (2014b). Approximate matching: definition and terminology. *NIST Special Publication*, 800:168.

Breitinger, F., Rathgeb, C., and Baier, H. (2014c). An efficient similarity digests database lookup-a logarithmic divide & conquer approach. *The Journal of Digital Forensics, Security and Law: JDFSL*, 9(2):155.

Harichandran, V. S., Breitinger, F., and Baggili, I. (2016). Bytewise approximate matching: The good, the bad, and the unknown. *The Journal of Digital Forensics, Security and Law: JDFSL*, 11(2):59.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3:91–97.

Martínez, V. G., Álvarez, F. H., and Encinas, L. H. (2014). State of the art in similarity preserving hashing functions. In *Proceedings of the International Conference on Security and Management (SAM)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Moia, V. H. G. and Henriques, M. A. A. (2017). A comparative analysis about similarity search strategies for digital forensics investigations. In *XXXV Simpósio Brasileiro de Telecomunicacões e Processamento de Sinais (SBrT 2017)*, São Pedro, Brazil.

NIST (2016). National software reference library. `http://www.nsrl.nist.gov/`. Accessed 2016 Set 13.

Roussev, V. (2010). Data fingerprinting with similarity digests. In *IFIP International Conf. on Digital Forensics*, pages 207–226. Springer.

Winter, C., Schneider, M., and Yannikos, Y. (2013). F2s2: Fast forensic similarity search through indexing piecewise hash signatures. *Digital Investigation*, 10(4):361–371.

 © 2017 Sociedade Brasileira de Computação

Flow-Based Intrusion Detection for SCADA networks using Supervised Learning

Gabriel Vasquez[1], **Rodrigo S. Miani** [2], **Bruno B. Zarpelão**[1]

[1]Computer Science Department - State University of Londrina (UEL)
Londrina, PR - Brazil

[2]School of Computer Science - Federal University of Uberlândia (UFU)
Uberlândia, MG - Brazil

gabriel@vasquez.com.br, miani@ufu.br, brunozarpelao@uel.br

Abstract. *Recent attacks on industrial networks have brought the question of their protection, given the importance of the equipment that they control. In this paper, we address the application of Machine Learning (ML) algorithms to build an Intrusion Detection System (IDS) for these networks. As network traffic usually has much less malicious packets than normal ones, intrusion detection problems have class imbalance as a key characteristic, which can be a challenge for ML algorithms. Therefore, we study the performance of nine different ML algorithms in classifying IP flows of an industrial network, analyzing the impact of class imbalance in the results. The algorithms were evaluated taking as main metrics the F1-Score and Averaged Accuracy. Our experiments showed that the three algorithms based on decision trees were superior to the others. Particularly, the Decision Jungle algorithm outperformed all the others.*

1. Introduction

Industrial control systems (ICS) are very common in large industries, as they monitor and control many devices through constant polling. This is accomplished using SCADA (Supervisory Control and Data Acquisition) systems, which usually communicate with the various devices using Modbus Protocol. Recent attacks on these networks have brought much attention to the industry about how to protect those networks [Loukas 2015] [Piggin 2015].

An analysis of incidents in critical infrastructures and other industrial networks in the period of 1982-2012 by Miller and Rowe [2012] shows that the number of attacks has a tendency to rise over the years. Moreover, the main motivation behind those incidents is to disrupt the services provided, hitting both industry and government infrastructures. This shows that the risk of an attack on such networks is increasing continuously.

Piggin [2015] conducted a survey involving 599 companies in which he found that 52% were unaware that industrial networks had vulnerabilities. According to him, we can consider that security in industrial networks is still in its early childhood, meaning that not only their administrators are unconscious of the susceptibility of an attack, but also about the risks that it represents. An example of a serious attack that hit industrial networks was Stuxnet, a malware specifically developed to disrupt this kind of network. Its main target was a Siemens controller used in Iran's nuclear centrifuges, and the main goal was to destroy the uranium-enriching centrifuges [Ntalampiras 2015], [Yusheng et al. 2017].

As ICS are gradually being connected to the Internet, the risks faced by these networks tend to increase even more. Hence, we note that there is a great need for an

 © 2017 Sociedade Brasileira de Computação

Intrusion Detection System (IDS) that can correctly detect attacks against these networks with the lowest number of false positives possible. An anomaly-based IDS using Machine Learning (ML) could provide satisfactory results to the network administrator as regular ICS traffic is related to a limited number of requests and responses, making it clearly different from malicious traffic.

In this paper, we propose that the protection of ICS could be partially accomplished by implementing an Anomaly-based IDS using ML algorithms. This IDS will be responsible for collecting and analyzing the packets that travel through the SCADA network. Therefore, we will gather and analyze Modbus/TCP traffic, aggregate it into IP flows and use this data as an input to several ML algorithms to find the ones that have the best results. We also will study the impacts of class imbalance, which is a common issue in anomaly detection problems. The evaluation will be performed using a dataset provided by Lemay and Fernandez [2016], which is publicly available.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 explains the proposed approach. Section 4 presents the dataset, the test environment, the results and discusses the experiments. Lastly, Section 5 presents the conclusions.

2. Related Work

In recent years, many methods have been proposed to aid intrusion detection in many areas, some of them specifically on industrial networks. The intrusion detection usually is divided into tree types: signature-based IDS, in which the detection of the malicious traffic depends on previously created signatures for the malicious traffic; specification-based IDS, in which the detection occurs from differences between the designed normal states of the network and the operational states; and, anomaly-based IDS, in which the detection of the malicious traffic occurs based on the deviations from the normal learned behavior of the network.

In the work of Linda et al. [2009], the authors use two Neural Network algorithms, namely the Error Back-propagation and the Levenberg-Marquardt, to analyze a SCADA network traffic. In their approach, the 8 most relevant features are extracted from the dataset according to a defined window size and merged with artificially generated intrusion traffic to compose the training dataset.

In the work of Junejo and Goh [2016] , the authors developed an anomaly-based IDS to be used on the SWaT water purification testbed. In the work, the authors used nine machine learning algorithms to classify the malicious traffic. Although the authors reached low false positive rates, high precision and recall, the work did not focus on the impacts of the class imbalance of the dataset generated or the benefits of grouping data into flows.

In the work of Yang et al. [2013], it is presented a hybrid approach between signature-based IDS and specification-based IDS for IEC 60870-5-104 networks. The first part of the proposed IDS detects malicious attacks using a set of Snort rules developed by the authors. The second part of the IDS increases the detection capability of the solution by detecting unknown attacks using the specifications of the expected behavior of the entire system. The work of the authors did not present the source of the used dataset or a comparison table to demonstrate that this approach provides relevant results. Also, the definition of the specifications in the second part would require an expert to define the normal states.

 © 2017 Sociedade Brasileira de Computação

Schuster et al. [2015] use One-Class Support Vector Machine (OCSVM) to develop an anomaly based IDS. In their approach, they group each 3 packets from the same origin in a vector. Then the authors employ SVM to discover the normal data distribution on a production ICS dataset in a Profibus network. The authors reach high precision, recall and F-score using different kernels for the SVM.

In the work of Yusheng et al. [2017], the authors developed an IDS for ICS using the ModBus/TCP Protocol. Their approach consists of two modules: rule extraction and deep inspection. The first part dissects the ModBus TCP packet into three subparts, namely network layer, transport layer and application layer. Then, the rule extraction module creates normal and abnormal rules based on the correlation among those three subparts. The deep inspection module is a resident module responsible for continually correlating the classifications to determine whether the normal or abnormal rule current applied is a false positive. It is not clear how the rules would be updated over time to detect new attacks or if the rule extraction module could be susceptible to manipulation on the sliding window process due to slow-rate attacks.

In this work, we choose a different approach than Linda et al. [2009] and Schuster et al. [2015], grouping data into flows, which allows an overview of the stream that is passing through the network, reducing the volume of data to be analyzed. We also choose to use a publicly available dataset differently from Yusheng et al. [2017] and Yang et al. [2013]. This allows the reproducibility of the attained results by future works in the area. The imbalance between normal traffic and anomalous is addressed by this work as well, analyzing the results with more suitable metrics than those used by Junejo and Goh [2016]. At last, the usage of machine learning algorithms may be more efficient in detecting unknown attacks since it does not require the development of signatures for new each attack or the definition of the specifications of the workload as in Yang et al. [2013].

3. Proposed Approach

In this section, we present our approach to detect intrusions in Modbus/TCP SCADA networks using a supervised classifier. Our approach is based on the hypothesis that attacks against a Modbus/TCP SCADA network will cause a significant deviation from normal traffic patterns. Therefore, using machine learning algorithms, we analyze those patterns to detect those variations and notify the network administrator about ongoing attacks.

The proposed system is divided into five parts: the Network Packets Gathering, the Flow Generation, the Feature Creation, the Training, and the Supervised Classification. An overview of the system is presented in Figure 1.

Figure 1. The proposed approach.

The first step of our approach is the Network Packets Gathering. It is responsible for capturing the Modbus/TCP packets. Modbus/TCP [Swales et al. 1999] is the connection-oriented version of traditional Modbus. Although there are other SCADA

© 2017 Sociedade Brasileira de Computação

protocols like DNP3, IEC-61850 and Profibus, our work is focused specifically on the Modbus/TCP traffic.

The following data from each packet is extracted to compose the initial dataset: a number to identify the packet, the timestamp, the source IP address, the source port, the destination IP address, the destination port, the communication protocol, the length of the packet, and the info field, which contains the information that allows the poll and control of the Modbus devices.

In the second step of our approach, referred to as Flow Generation, the packet data extracted in the first step is used to group the packets into IP flows according to the IP Flow Information Export (IPFIX) Protocol [Claise et al. 2013]. IPFIX protocol specifies that a IP flow is composed of packets with the same source and destinations addresses, source and destination ports, and transport protocol. Besides, the IPFIX protocol suggests a timeout window of 1800 seconds to consider a flow as inactive, which we adopted. The main advantage of the IP flows is that they provide an overview of the network behavior, showing the main statistics of the interactions among the network nodes, which can reveal deviations from the normal behavior caused by attacks. In this work, we consider an IP flow as malicious if any packet of this flow is related to a malicious activity.

The next step of our approach is the Feature Creation. In this step, features are extracted from the IP flows generated in the previous step, creating data instances that will be the input of the supervised classification algorithm. One of the main characteristics of the traffic in SCADA networks is that it is very well established, showing recognizable pattern in the communication among devices. That means that the polling transactions between masters and slaves follow a very foreseeable schedule, and the packets that travel in these networks usually have similar sizes. Based on these particularities, we believe that IP flow features, namely average packet size, source and destination port numbers, and packets per second may indicate that an IP flow is malicious. All the features and their descriptions can be seen in Table 1.

In the Training step, a set of data instances labeled as malicious or legitimate is provided to the machine learning algorithm so it can build a model to classify the future data instances automatically.

In the last step, the Supervised Classification, the machine learning algorithm trained in the previous step is employed to classify the instances produced by the Feature Creation step as malicious or legitimate. Since we are dealing with traffic patterns, we believe that machine learning algorithms will deal with this type of traffic better than signature-based detection systems, specially when dealing with unknown attacks.

 © 2017 Sociedade Brasileira de Computação

Table 1. Flow based dataset features.

Feature	Description
source_port	Source Port, originally created on Flow Generation step, represents the source transport port used on the traffic between two devices.
dest_port	Destination Port, originally created on Flow Generation step, represents the destination transport used on the traffic between two devices.
prot	Transport Protocol, originally created on Flow Generation step, represents the transport protocol used on the traffic between two devices. It is expected to have only the ModBus/TCP inside.
num_pack	Number of Packets that compose the flow, created on the Feature Generation step, represents the count of number of packets on a flow.
size_pack	Size of Packets, created on the Feature Generation step, represents the sum of the size of all packets that compose the flow.
ppf	Packets per flow, created on the Feature Generation step, represents the count of all packets that compose the flow.
bpf	Bytes per Flow, created on the Feature Generation step, represents the ratio between size_pack and flow_duration.
avg_size_pack	Average of packets size, created on the Feature Generation step, represents the average size of the packets that compose the flow.
flow_duration	Flow Duration, created on the Feature Generation step, represents the duration of the flow.

4. Evaluation

In this section of the paper, we use a publicly available dataset, created by [Lemay and Fernandez 2016], to evaluate our approach. The evaluation was accomplished as follows. First, the dataset generated by Lemay and Fernandez [2016] was used as input for the Flow Generation step. Then, the IP flows extracted in the previous step were sent to the Feature Creation step. Finally, in the Supervised Classification step, we employed nine machine learning algorithms on the extracted features.

This section provides, in the first place, a description of the dataset, its testbed, and the methodology. Then, we present the evaluation metrics and the results using each algorithm.

4.1. Dataset

The main reason that discourages companies from sharing their datasets with the scientific community is the sensitivity of the data that travel on SCADA networks. Yussof et al. [2014] discussed this issue in their work. Therefore, there are few datasets available publicly, and the few that are available are generated in controlled environments or have limitations on data confidentiality.

Aware of the limitations of the few datasets available, Lemay and Fernandez [2016] developed in a controlled environment a labeled dataset for SCADA networks using the Modbus protocol, making it public for studies and evaluations in intrusion detection area. The dataset, which was used in this paper, includes about 6 hours of both malicious and non-malicious traffic on a simulated industrial network. For further information about the workload and the network, please check Lemay and Fernandez [2016].

© 2017 Sociedade Brasileira de Computação

Although most of the traffic in the dataset is related to automated transactions between masters and slaves, the few discrepancies in time patterns or packet size can represent a human interacting with the system or also, an attack. The dataset includes attacks such as running a remote exploit, uploading files to the infected network, fingerprinting attack, and sending an unauthorized command to a controller.

The *pcap* files provided by the original dataset were then processed by the Network Gathering module, and finally aggregated into flows by the Flow Generation Module. It is important to mention that only the dataset files with malicious activity were used, meaning that only the files named Moving_two_files_Modbus_6RTU.pcap, Send_a_fake_command_Modbus_6RTU_with_operate.pcap, CnC_uploading_exe_modbus_6RTU_with_operate.pcap and 6RTU_with_operate.pcap were imported. The description of each file can be seen in [Lemay and Fernandez 2016].

The results of the conversion of the original dataset to a flow-based dataset are presented in Table 2.

Table 2. Original and flow-based datasets.

	Original Dataset (packets)	**Flow-Based Dataset (flows)**
Majority Class (non malicious)	16362	5488
Minority Class (malicious)	1405	38
Unbalance Ratio	11:1	144:1
Total	**17767**	**5526**

4.2. Test Environment and Methodology

In the Supervised Classification step, we tested nine two-class algorithms as follows: Support Vector Machine, Locally Deep Support Vector Machine, Averaged Perceptron, Neural Network, Logistic Regression, Bayes Point Machine, Boosted Decision Tree, Decision Forest, and Decision Jungle.

In this work, we aimed to use the most relevant algorithms from each classifiers family. From the family of discriminative classifiers, we selected Support Vector Machine (SVM), Locally Deep Support Vector Machine (LD-SVM), Averaged Perceptron and Neural Network. From the family of decision tree classifiers, we picked Boosted Decision Tree, Decision Forest, and Decision Jungle. At last, from the statistical classifiers family, Logistic Regression and Bayes Point Machine were chosen.

Support Vector Machine (SVM) and Locally Deep Support Vector Machine (LD-SVM) are ML algorithms that aim to predict the correct class by building a multidimensional vector, in which each instance of the dataset is classified. The goal is to obtain the largest margin of separation of the hyper-plane between classes. The main difference between the regular SVM and the LD-SVM is that the last one uses a different kernel that works better on large datasets.

Averaged Perceptron and Neural Network are ML algorithms that belong to the Neural Network family. Averaged Perceptron is the simplest type of Neural Network algorithm and uses a single neuron. It linearly classifies data using an activation function, in which the weighted data is used as input. The Neural Network algorithm is the gen-

 © 2017 Sociedade Brasileira de Computação

eralization of Averaged Perceptron, which, instead of using a single neuron, has a set of neurons interconnected in a directed, weighted, and acyclic graph.

Logistic Regression is an ML algorithm that predicts the statistical probability of a sample belonging to a class using a logistic distribution of the samples. Bayes Point Machine is also a ML algorithm, which uses a Bayesian linear approach to classify data.

Boosted Decision Tree, Decision Forest, and Decision Jungle are ML algorithms based on decision tree classification. The Boosted Decision Tree is an ensemble learning method, in which each tree corrects the errors of its predecessor, and the prediction is made by evaluating the entire ensemble. Decision Forest, also known as Random Decision Forest, is an ensemble algorithm that classifies data by randomly constructing multiple decision trees and voting in whichever provides the closest probability of correct label. Decision Jungle is considered an extension of Decision Forest, which instead of using one path to every node, uses directed acyclic graphs to allow multiple paths from the root to each leaf.

The classifier module and all algorithms were implemented on Microsoft Azure Machine Learning Studio [Microsoft 2017]. Microsoft Azure Machine Learning Studio is a cloud predictive analytic tool that allows the implementation of many machine learning models and executes them in parallel. We executed all the algorithms using their default parameters. The algorithms tested and their defaults parameters can be seen in Table 3.

Table 3. Algorithm parameters.

Algorithm	Parameters
Support Vector Machine	**Create trainer mode:** Single Parameter **Number of iterations:** 1 **Lambda:** 0.001 **Normalize features:** True **Project to the unit-sphere:** False **Allow unknown categorical levels:** True
Locally-Deep Support Vector Machine	**Create trainer mode:** Single Parameter **Depth of the tree:** 3 **Lambda W:** 0.1 **Lambda Theta:** 0.01 **Lambda Theta Prime:** 0.01 **Sigmoid sharpness:** 1.0 **Number of iterations:** 15000 **Feature normalizer:** Min-Max normalizer **Allow unknown categorical levels:** True
Averaged Perceptron	**Create trainer mode:** Single Parameter **Learning rate:** 1.0 **Maximum number of iterations:** 10 **Allow unknown categorical levels:** True

© 2017 Sociedade Brasileira de Computação

Neural Network	**Create trainer mode:** Single Parameter **Hidden layer specification:** Fully-connected case **Number of hidden nodes:** 100 **Learning rate:** 0.1 **Number of learning iterations:** 100 **The initial learning weights:** 0.1 **The momentum:** 0 **The type of normalizer:** Min-Max normalizer **Shuffle examples:** True **Allow unknown categorical levels:** True
Logistic Regression	**Create trainer mode:** Single Parameter **Optimization tolerance:** 10^{-7} **L1 regularization weight:** 1 **L2 regularization weight:** 1 **Memory size for L-BFGS:** 20 **Allow unknown categorical levels:** True
Bayes Point Machine	**Number of training iterations:** 30 **Include bias:** True **Allow unknown values in categorical features:** True
Boosted Decision Tree	**Create trainer mode:** Single Parameter **Maximum number of leaves per tree:** 20 **Minimum number of samples per leaf node:** 10 **Learning rate:** 0.2 **Number of trees constructed:** 100 **Allow unknown categorical levels:** True
Decision Forest	**Resampling method:** Bagging **Create trainer mode:** Single Parameter **Number of decision trees:** 8 **Maximum depth of the decision trees:** 32 **Number of random splits per node:** 128 **Minimum number of samples per leaf node:** 1 **Allow unknown values for categorical features:** True
Decision Jungle	**Resampling method:** Bagging **Create trainer mode:** Single Parameter **Number of decision DAGs:** 8 **Maximum depth of the decision DAGs:** 32 **Maximum width of the decision DAGs:** 128 **Number of optimization steps per decision DAG layer:** 2048 **Allow unknown values for categorical features:** True

To evaluate the different algorithms, we used the hold-out approach. According to the hold-out approach, the dataset was randomly divided into two subsets. The first subset was called training set. It contained 60% of the original data and was used to build the predictive model. The second subset was called test set. It contained the remaining 40% of the original data and was used to test and measure the performance of the model on unseen inputs.

 © 2017 Sociedade Brasileira de Computação

In addition to the hold-out approach, we also applied a downsampling on the majority class of the training set. Downsampling is a technique that removes some data from a given dataset randomly. The objective of this technique is to balance the classes on the training set. According to He and Ma [2013], some ML algorithms perform better on anomaly detection when the imbalance between classes is reduced. Therefore, we tested each algorithm reducing the imbalance as presented in Table 4, where the values 1%, 2%, 3%, 4% and 5% mean the reduction rate in the training dataset regarding the 100% value. For example, a reduction to 5% will reduce from 3293 to 132 samples. Also, we made the same tests without the downsampling to compare results.

When detecting malicious traffic, we usually deal with imbalanced classes, since most of the traffic is non-malicious rather than malicious. In the original dataset, handled in the Network Gathering step, we dealt with an 11:1 ratio between malicious and non-malicious packets. After the aggregation into a flow based dataset, this ratio increased to 144:1 between malicious and non-malicious flows, rising the importance of studying the impact of downsampling on the overall results.

Table 4. Downsampling values and imbalance ratio

	Training Dataset						**Test Dataset**	**Total**
	1%	**2%**	**3%**	**4%**	**5%**	**100%**		
Non-malicious flows	33	66	99	132	165	3293	2195	**5488**
Malicious flows	23	23	23	23	23	23	15	**38**
Ratio	1:1	3:1	4:1	5:1	7:1	143:1	146:1	**144:1**

4.3. Evaluation Metrics and Results

He and Ma [2013] recommend the use of F1-Score and Averaged Accuracy as the main metrics to evaluate the performance of any ML algorithm with an imbalanced dataset. We also added Accuracy for further reference. According to He and Ma [2013], a dataset is considered imbalanced if the imbalance between classes has a higher ratio than 10:1. In our case, the total ratio is 144:1, as previously presented in Table 2. In the training set, the downsampling technique reduced that imbalance from 144:1 up to a 1:1 ratio, although it represents heavy losses in information due to the random characteristic of the technique. According to the authors, the biggest problem of class imbalance is the increase of error rate as the imbalance increases, which impacts directly on the ability to learn the minority class, biasing the major class. That means that not every algorithm will perform within an acceptable level in this situation. The evaluation metrics, their formulas, scores, and a brief description are presented next.

1. $F1_Score = 2\frac{Precision*Recall}{Precision+Recall} = \frac{2TP}{2TP+FP+FN}$
   It measures the harmonic mean of Precision and Recall. Precision stands for the ratio between correct predictions (TP) and the number of all correct (TP) and incorrect predictions (FP). Recall stands for the ratio between correct predictions (TP) and the number of all correct predictions (TP + FN). F1 Score is represented as a continuous number between 0 and 1, being 1 as the best value and 0 the worst.
2. $Averaged_Accuracy = \frac{Recall+Specifity}{2} = \frac{TP}{2TP+2FN} + \frac{TN}{2TN+2FP}$
   It measures the balanced accuracy, balancing the scores between Recall and Specifity. Specifity stands for the ratio between correct predictions on the majority class

© 2017 Sociedade Brasileira de Computação

(TN) and the number of all correct predictions on the majority class (TN + FP). Averaged Accuracy is represented as a continuous number between 0 and 1, being 1 as the best value and 0 the worst.

3. $Accuracy = \frac{TP+TN}{TP+FN+FP+TN}$
   It measures the closest an implemented model correctly predicts all cases. It is represented as a continuous number between 0 and 1, being 1 as the best value and 0 the worst

TP, TN, FP, and FN stand for true positives, true negatives, false positives and false negatives respectively.

On the execution of each classification technique, we ran each algorithm ten times for each downsampling value and also for the non-downsampled training dataset. After each execution, each one of the nine algorithms generated 100 values corresponding to the thresholds evaluated. After all executions, we collected 54000 values as results and their averages are presented in Table 5, where the corresponding downsampling ratios are presented between brackets.

Table 5. Results for evaluation metrics regarding the nine algorithms

	F1-Score	**Average Accuracy**	**Accuracy**
Support Vector Machine	0.1278 (5%)	0.5877 (2%)	0.9934 (100%)
Locally Deep Support Vector Machine	0.7523 (100%)	0.9489 (5%)	0.9970 (100%)
Averaged Perceptron	0.4455 (5%)	0.8114 (1%)	0.9942 (100%)
Neural Network	0.5399 100%	0.8597 (3; 4%)	0.9935 (100%)
Logistic Regression	0.4416 (100%)	0.8744 (3; 4%)	0.9946 (100%)
Bayes Point Machine	0.7730 (100%)	0.9518 (100%)	0.9970 (100%)
Boosted Decision Tree	0.8467 (100%)	0.9684 (4; 5; 100%)	0.9977 (100%)
Decision Forest	0.8026 (100%)	0.9694 (100%)	0.9972 (100%)
Decision Jungle	0.8508 (100%)	0.9690 (4; 100%)	0.9979 (100%)

Analyzing the results presented in Table 5, the five best algorithms were Decision Jungle, Boosted Decision Tree, Decision Forest, Bayes Point Machine and Locally Deep Support Vector Machine. Further discussion on those results will be carried out in the Discussion subsection.

Another two evaluators were also implemented to compare the results of the five best algorithms. The first, presented in Figure 2, is the Receiver Operating Characteristic (ROC) curve, allowing to visually see the variation of the True Positive Rate versus the False Positive Rate. The second evaluator is presented in Figure 3, showing the trade-off between Precision and Recall.

 © 2017 Sociedade Brasileira de Computação

In Figure 2, we present the ROC curve for the five best algorithms, each of them showing their best results without the use of the downsampling technique. In those graphs, we can see that Decision Jungle and Decision Forest have the highest and almost identical performance, followed by Bayes Point Machine.

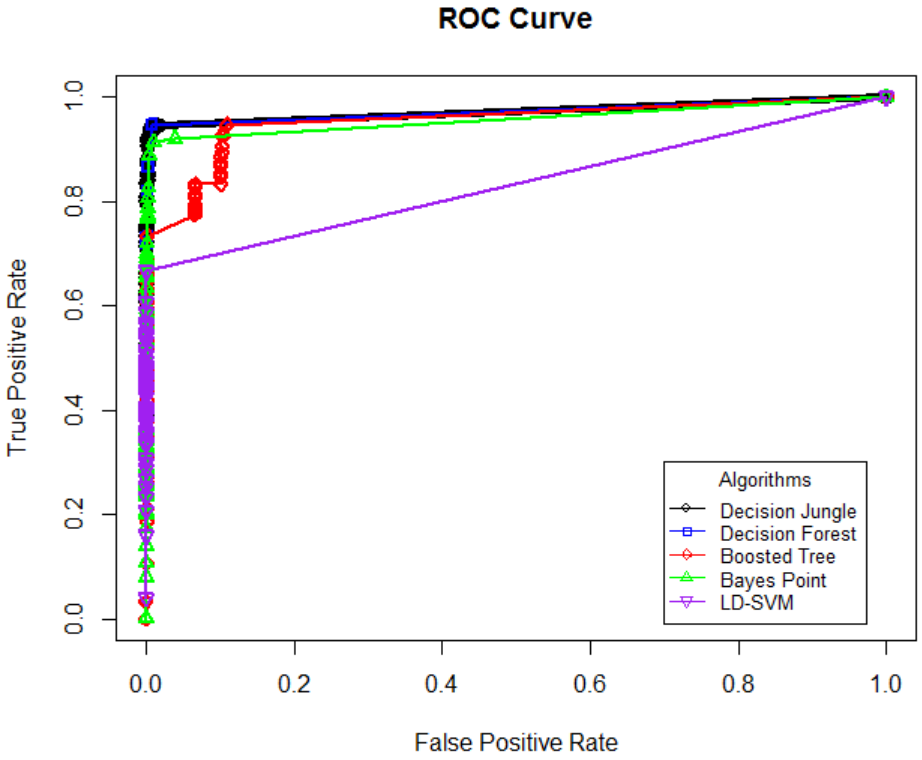


Figure 2. ROC Curve

In Figure 3, we present the Precision-Recall curve for the five best algorithms, each of them showing their best results without the use of the downsampling technique. In those graphs, we can see that Decision Jungle has the highest performance followed by Boosted Tree.

4.4. Discussion

The results presented in Table 5, and the results shown in Figure 2 and Figure 3 allow us to reach some conclusions to be discussed in this section.

First, based on F1-Score and Average Accuracy presented in Table 5, we can conclude that all three Decision Tree algorithms outperformed the other ones, presenting the best of all F1-Score over 80% and Average Accuracy over 96%. The only other two algorithms that showed similar, although inferior, results are Bayes Point Machine and Locally Deep Support Vector Machine.

Second, analyzing the overall results presented in Table 5, we conclude that the downsampling technique applied on the training dataset did not provide significant aid to the best-scored algorithms. In fact, the downsampling helped only the Averaged Perceptron and the Support Vector Machine, but not enough to make them relevant toward the construction of an IDS.

Figure 2 shows the ROC curve of the five best algorithms selected through the results presented in Table 5. The ROC curve allowed us to compare the trade-off between

© 2017 Sociedade Brasileira de Computação

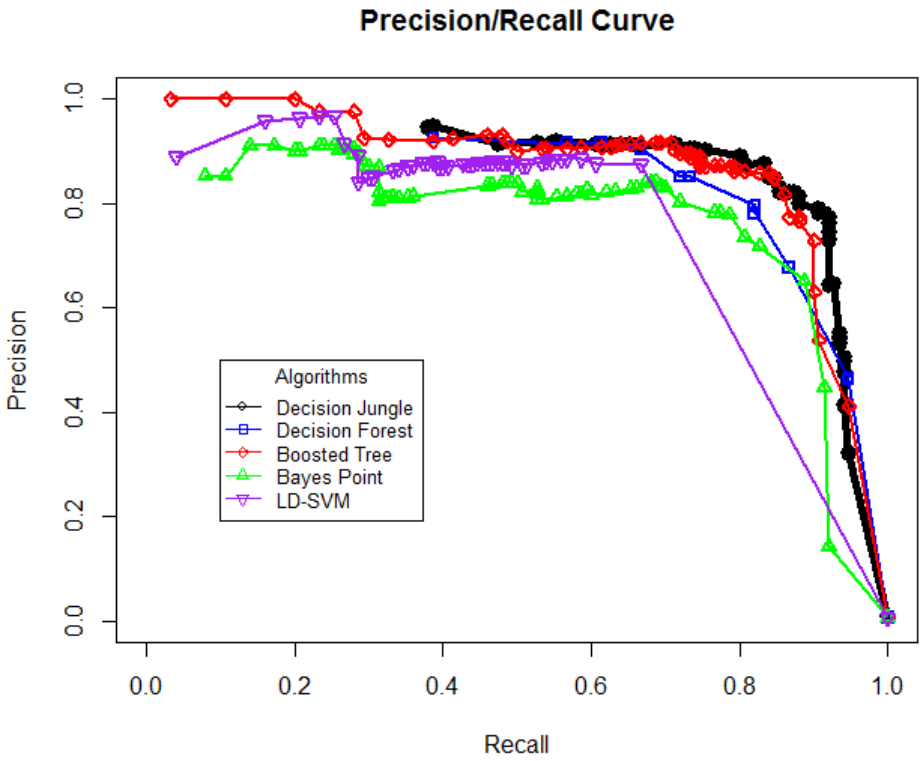


Figure 3. The Precision Recall Curve

the True Positive Rate (TPR) and the False Positive Rate (FPR) of the algorithms. In ROC curves, the better results are in the upper left corner, where the TPR is closer to 1 and the FPR is closer to 0. In Figure 2, we could see that the algorithms Decision Jungle and Decision Forest sustained a higher performance through all the threshold variation when compared to the other algorithms. These two algorithms reached, for the most of the graph, the desired top left corner. On the other hand, Boosted Decision Tree could not sustain similar results. Here, it is noteworthy that Bayes Point Machine did not outperform the two best algorithms, but had an overall good performance. Locally Deep SVM had the worst results on this evaluator.

Figure 3 shows the Precision-Recall Curve of the same five best algorithms selected through the results presented in Table 5. The Precision-Recall Curve is a type of curve that visually shows the quality of the results of any given classifier. A classifier with high recall but low precision will detect the most of the attacks, but will generate many false positives. A classifier with low recall but high precision will generate few false positives, but will miss many attacks. The desired output is reached on the upper right corner, where there is a high Precision with a high Recall.

In the presented results, we could see that the Decision Jungle algorithm presented higher overall results than any other one. The next best result was reached by Boosted Tree algorithm. Here, we could also see that although Decision Forest presented high results in Table 5 and in the ROC Curve, it did not have a good precision-recall trade-off. All other algorithms did not perform well enough on this evaluation.

After analyzing the results, we observed that the Decision Jungle algorithm presented the highest scores in Table 5 and an overall good performance in the ROC Curve and Precision/Recall Curve. Therefore, we pointed out it as the best solution among the

 © 2017 Sociedade Brasileira de Computação

algorithms we evaluated.

We believe that there are two main reasons for the good performance of Decision Jungle. The first one is the ensemble learning. It employs multiple learning models, leveraging the classification power of multiple base learners, such as decision trees, to improve the classification performance over traditional algorithms.

In the case of Decision Jungle [Shotton et al. 2013], multiple decision trees are connected using directed acyclic graphs, allowing multiple paths from the root to each leaf. That type of connection among nodes allowed a higher degree of merging between leafs, reducing the complexity of the classifier and the number of misclassifications. For Modbus/TCP traffic, this approach presented a significant leverage in comparison with other non-ensemble algorithms.

The second reason that helped the Decision Jungle to be the best among the others is its awareness of class imbalance. That awareness is due to ensemble model. In our aggregation, in Flow Generation step, we ended with a class imbalance of 144:1. That imbalance could have impacted the poor results of some algorithms like SVM or Perceptron, but it did not impact more advanced models like the Decision Tree family algorithms. We believe that this is directly related to the ensemble model. We also noticed that the downsampling had no benefit on the Decision Jungle algorithm, since its best results were reached with no downsampling.

5. Conclusion

In this paper, we evaluated the performance of nine different ML algorithms in classifying IP flows of an SCADA ModBus/TCP network, analyzing the impact of class imbalance in the results. Those algorithms may be the core of the detection engine of an Intrusion Detection System (IDS). The role of an IDS is to monitor the network, collecting data to notify the administrators.

In a SCADA ModBus/TCP network, the traffic presented is different from a regular TCP/IP network. The ModBus/TCP network is a restricted network, where the addition of a new device is rare. The traffic is standardized and predictable between masters and slaves due the polling process. This characteristic allowed us to arrange similar packets into IP flows. In the flows, we also compute the average packet size, count number of packets and the flow duration. The predictability in the SCADA traffic also allowed us to establish a communication pattern that could be analyzed using a machine learning supervised classifier.

The analysis of the results of the studied algorithms showed us that the Decision Jungle presented the best scores for the metrics results (F1 Score and Average Accuracy) and the overall performance in the ROC Curve and Precision/Recall Curve. The second best performance was reached by Boosted Tree algorithm, presenting a good score for the evaluation metrics, almost the same performance that Decision Jungle in the ROC Curve and a standard performance in the Precision/Recall Curve. We believe that those results were accomplished due to the ensemble model in the construction of those algorithms, allowing them to outperform all other simpler algorithms.

As future work, we intend to extend this work by using semi-supervised ML algorithms. That will allow us to reduce the dependency of the labeled data in the dataset. We also intend to use differents settings on Microsoft Azure parameters to improve the detection rate of all algorithms.

© 2017 Sociedade Brasileira de Computação

References

Claise, B., Trammell, B., and Aitken, P. (2013). Specification of the IP Flow Information Export (IPFIX) Protocol for the exchange of flow information (2013).

He, H. and Ma, Y. (2013). *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons.

Junejo, K. N. and Goh, J. (2016). Behaviour-based attack detection and classification in cyber physical systems using machine learning. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 34–43. ACM.

Lemay, A. and Fernandez, J. M. (2016). Providing SCADA network data sets for intrusion detection research. *9th USENIX Workshop on Cyber Security Experimentation and Test (CSET '16)*, pages 1–8.

Linda, O., Vollmer, T., and Manic, M. (2009). Neural network based intrusion detection system for critical infrastructures. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 1827–1834. IEEE.

Loukas, G. (2015). *Cyber-physical attacks: A growing invisible threat*. Butterworth-Heinemann.

Microsoft (2017). How to choose algorithms for Microsoft Azure machine learning.

Miller, B. and Rowe, D. (2012). A survey SCADA of and critical infrastructure incidents. In *Proceedings of the 1st Annual conference on Research in information technology*, pages 51–56. ACM.

Ntalampiras, S. (2015). Detection of integrity attacks in cyber-physical critical infrastructures using ensemble modeling. *IEEE Transactions on Industrial Informatics*, 11(1):104–111.

Piggin, R. (2015). Are industrial control systems ready for the cloud? *International Journal of Critical Infrastructure Protection*, 9(C):38–40.

Schuster, F., Paul, A., Rietz, R., and König, H. (2015). Potentials of using one-class SVM for detecting protocol-specific anomalies in industrial networks. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 83–90. IEEE.

Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., and Criminisi, A. (2013). Decision jungles: Compact and rich models for classification. In *Proc. NIPS*.

Swales, A. et al. (1999). Open ModBus/TCP specification. *Schneider Electric*, 29.

Yang, Y., McLaughlin, K., Littler, T., Sezer, S., Pranggono, B., and Wang, H. (2013). Intrusion detection system for IEC 60870-5-104 based SCADA networks. In *Power and Energy Society General Meeting (PES), 2013 IEEE*, pages 1–5. IEEE.

Yusheng, W., Kefeng, F., Yingxu, L., Zenghui, L., Ruikang, Z., Xiangzhen, Y., and Lin, L. (2017). Intrusion detection of industrial control system based on Modbus TCP protocol. In *Autonomous Decentralized System (ISADS), 2017 IEEE 13th International Symposium on*, pages 156–162. IEEE.

Yussof, S., Rusli, M. E., Yusoff, Y., Ismail, R., and Ghapar, A. A. (2014). Financial impacts of smart meter security and privacy breach. In *Information Technology and Multimedia (ICIMU), 2014 International Conference on*, pages 11–14. IEEE.

 © 2017 Sociedade Brasileira de Computação

Análise de Alertas de Sistemas de Detecção de Intrusão: Uso de Aprendizado Supervisionado na Redução de Alertas Falsos Positivos

Eduardo Alves Moraes[1 2], **Carlos Alexandre Carvalho Tojeiro**[2], **Rodrigo Sanches Miani**[3], **Bruno Bogaz Zarpelão**[1]

[1]Departamento de Computação - Universidade Estadual de Londrina (UEL)
Londrina, PR - Brasil

[2]Faculdade de Tecnologia de Ourinhos - (FATEC - OU) - Ourinhos, SP - Brasil

[3]Faculdade de Computação - Universidade Federal de Uberlândia (UFU)
Uberlândia, MG - Brasil

{eduardo.moraes, carlos.tojeiro}@fatecourinhos.edu.br

miani@ufu.br, brunozarpelao@uel.br

Abstract. *Intrusion Detection Systems (IDS) detect various types of malicious behavior in computer systems, which can compromise their security and reliability. Although IDSs improve system protection, there is a problem: the generation of alerts that do not represent the real situation of the computational environment, referred to as false positive alerts. This paper presents an approach for reducing false positive alerts, using priority alert filtering and machine learning methods. A separation of alerts is performed based on their priorities as well as the insertion of new information based on other data sources. Then machine learning algorithms (kNN and Random Forest) are applied based on a supervised classifier to identify false positives. The approach has achieved its goal by presenting a significant reduction of false positive alerts in a case study conducted on a real corporate network.*

Resumo. *Sistemas de Detecção de Intrusão (IDS - Intrusion Detection System) detectam vários tipos de comportamentos maliciosos em sistemas computacionais, que podem comprometer sua segurança e confiabilidade. Embora os IDS melhorem a proteção dos sistemas, existe um problema: a geração de alertas que não representam a real situação do ambiente computacional, chamados de alertas falsos positivos. Este artigo apresenta uma abordagem de redução de alertas falsos positivos, utilizando filtragem de alertas por prioridade e métodos de aprendizado de máquina. É realizada uma separação de alertas com base nas suas prioridades e a inserção de novos atributos com base em outras fontes de dados. Em seguida, são aplicados os algoritmos de aprendizado de máquina (kNN e Random Forest) com base em um classificador supervisionado para identificar os falsos positivos. A abordagem atingiu o seu objetivo, apresentando uma redução significativa dos alertas falsos positivos em um estudo de caso realizado em uma rede corporativa real.*

 © 2017 Sociedade Brasileira de Computação

1. Introdução

Nas décadas de 80 e 90, o computador pessoal se popularizou, assumindo um papel fundamental no dia a dia das pessoas e organizações. Atualmente, as redes de computadores são os principais meios que as empresas, organizações e governos utilizam na disponibilização de diversos serviços de rede como *Web Services*, Bancos de Dados, Serviços em Nuvem, dentre outros. O maior desafio não é apenas manter a disponibilidade destas redes, mas também detectar a presença de indivíduos conectados, que por razões diversas, tentam paralisar o fornecimento dos serviços, ou interceptar as informações trocadas entre os dispositivos. Grande parte dos administradores de sistemas computacionais, em um dado momento, irão se deparar com um evento de intrusão de segurança durante suas carreiras. Ter um plano de detecção de intrusão resultará em uma notificação mais ágil, minimizando as consequências e permitindo uma rápida recuperação, em caso de algum sinistro [Kurose et al. 2010] [Julisch and Dacier 2002].

A quantidade de incidentes de segurança reportados no Brasil tem mostrado uma tendência de crescimento nos últimos anos, segundo as estatísticas da CERT.br (2015). Dos 722.205 incidentes reportados, 54,17% são ataques do tipo *scan*, 23,37% são ataques do tipo fraude ou páginas falsas, 9,09% são ataques que comprometem servidores Web, 6,61% são ataques do tipo *worm*, 3,51% são ataques de negação de serviços ditribuídos, 0,34% são ataques do tipo invasão à sistemas e os 2,91% são as demais categorias de ataques menos expressivas. Tais dados reforçam a demanda por ferramentas dedicadas à defesa das redes de computadores. Os Sistemas de Detecção de Intrusão (IDS - *Intrusion Detection System*) englobam o processo de monitoramento de redes e dos *hosts*, identificando e notificando a ocorrência de atividades maliciosas ou não autorizadas [Granadillo et al. 2016]. Entende-se como intrusão toda atividade que pode resultar em alteração de privilégios de usuários e permissões em arquivos, instalação de *malware*, acesso não autorizado a arquivos e sistemas, ataques de negação de serviços, presença de *worms* e vírus, estouros de *buffers*, entre outros [Ebrahimi et al. 2011]. Os IDS apresentam seus alertas no formato de *logs*. Por meio da análise destes *logs*, os administradores de sistemas conseguem efetivar um plano de detecção e resposta às intrusões.

No entanto, existe um problema relacionado à efetividade dos IDS: a geração de um grande volume de alertas, dos quais muitos são considerados falsos, dificultando a análise feita pelos administradores de sistemas. Alvarenga et al. (2015) e Granadillo et al. (2016) abordam em seus trabalhos o problema de grandes volumes de alertas que um IDS pode gerar, dificultando o trabalho dos analistas de segurança. Ambos apostam em técnicas de correlação de alertas para facilitar a análise destes grandes volumes de informação. Elshoush et al. (2014), como primeiro passo do processo de correlação de alertas que eles propõem, buscam descartar os alertas falsos, pois desta forma podem reduzir a quantidade de alertas a serem processados. Verma et al. (2015) afirmam que é necessário combinar técnicas de mineração de dados e aprendizado de máquina para compreender as grandes estruturas de dados formadas por alertas, suas correlações e seus comportamentos.

Neste trabalho, é proposto um processo de redução de alertas que se baseia na prioridade dos alertas e na remoção de falsos positivos. Para tanto, assumimos que é possível diferenciar alertas falsos de alertas verdadeiros a partir da análise de informações dos alertas como origem e destino do ataque, assinatura reportada no alerta, função que os *hosts*

 © 2017 Sociedade Brasileira de Computação

envolvidos desempenham na rede, e horário do evento. A partir desta hipótese, desenvolvemos um modelo baseado em aprendizado de máquina supervisionado para identificar os falsos positivos. Para avaliar a proposta, foi realizado um estudo de caso em uma rede corporativa real, utilizando dois algoritmos de classificação supervisionada: o *Random Forest* e o *kNN (k Nearest Neighbors)*. Os resultados mostraram altos índices de redução de alertas e capacidade de classificação de alertas falsos com taxas de acerto próximas a 100%.

Este artigo está organizado na seguinte forma: a Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta a proposta do trabalho, detalhando o processo de redução de alertas. Na Seção 4, é discutido o estudo de caso realizado em uma rede corporativa real e a análise dos resultados obtidos. A Seção 5 contém a conclusão deste artigo.

2. Trabalhos Relacionados

Pesquisadores têm proposto diferentes soluções para lidar com os grandes volumes de alertas gerados pelos IDS, incluindo, entre os pontos abordados, a identificação e tratamento dos alertas falsos positivos. Do ponto de vista da redução de falsos positivos, podemos organizar estes trabalhos em dois grupos.

O primeiro grupo inclui trabalhos que têm a redução de falsos positivos apenas como um passo intermediário de um processo maior. No trabalho de Shittu et al. (2012), é proposta uma abordagem de visualização de alertas de intrusão, utilizando um agente de correlação de alertas que melhora a compreensão de cenários de ataques. Dado um determinado conjunto de eventos gerados por um IDS, a abordagem dos autores seleciona o método mais eficaz para a correlação de eventos de segurança. A eliminação de falsos positivos é apenas um passo presente na etapa inicial do processo, a agregação dos alertas.

Elshoush (2014) propôs um processo no qual a filtragem de alertas falsos é um passo inicial, que visa diminuir o volume de dados a ser analisado durante a correlação, aumentando a sua eficiência. Primeiramente, o autor removeu uma serie de alertas considerados de baixo risco. Depois, os alertas restantes foram agrupados de acordo com os seus *hosts* de origem e destino. Os alertas que não foram agrupados nesta fase foram classificados como falsos. Os autores consideraram que alertas falsos têm um comportamento aleatório e por isso acabam não sendo agrupados de acordo com os critérios utilizados no trabalho. Em nosso trabalho, observamos que alertas falsos não estão ligados obrigatoriamente a situações incomuns. Pelo contrário, alertas falsos têm algumas características em comum, que costumam diferenciá-los de alertas verdadeiros.

No trabalho de Kawakani et al. (2016), os autores propõem uma abordagem baseada no agrupamento de alertas. Duas etapas se destacam: a primeira etapa é a correlação dos alertas e a observação do histórico destes alertas para a identificação das estratégias de ataque utilizadas. A segunda etapa é a associação dos alertas em tempo real às estratégias de ataque descobertas na primeira etapa. Nesta proposta, de maneira similar ao trabalho de Elshoush (2014), alertas que não atenderam aos critérios de agrupamento foram removidos do processo de correlação. No entanto, não houve uma análise mais cuidadosa para certificar se eles eram mesmo falsos ou não.

O segundo grupo de trabalhos inclui aqueles que têm a priorização dos alertas, e a consequente identificação de falsos positivos, como o objetivo final do processo proposto.

© 2017 Sociedade Brasileira de Computação

Em um trabalho mais recente de Shittu et al. (2014), os autores propõem uma métrica para priorização de alertas de intrusão usando a correlação dos alertas. Primeiramente, os alertas são correlacionados em grafos. Então, os alertas pertencentes a grafos que correspondem aos comportamentos menos frequentes recebem prioridades mais altas. Os alertas com prioridades mais baixas são identificados como alertas falsos, sendo filtrados.

Vidal et al. (2015) propuseram a utilização de algoritmos de clusterização na priorização de alertas. No trabalho, primeiramente, há uma fase de treinamento na qual o IDS analisa qual é o comportamento normal da rede. Logo após, o IDS gera alertas para pacotes que se distanciam deste comportamento normal. De acordo com a proposta, para cada novo alerta gerado, é calculada a distância entre o pacote que causou a geração do alerta e os pacotes normais do treinamento. Quanto maior a distância entre o pacote responsável pelo alerta e os pacotes não maliciosos do treinamento, menor será a prioridade do alerta. Alertas com prioridade baixa são considerados falsos.

O trabalho proposto neste artigo pertence ao grupo dos trabalhos que tem a redução de falsos positivos como objetivo final. No entanto, ao invés de correlacionar os alertas para estabelecer a prioridade deles e definir se são falsos ou não, propomos uma abordagem que analisa atributos do alerta que podem indicar se ele é falso ou não, como a função dos *hosts* de origem e destino na rede e o horário do alerta. Desta forma, cada alerta é analisado individualmente, sem necessitar de correlação com outros alertas.

3. Proposta

A proposta deste trabalho é reduzir a quantidade de alertas, filtrando aqueles que possuem prioridade baixa e utilizando algoritmos de aprendizado de máquina para identificar os falsos positivos. A Figura 1 apresenta uma visão geral do modelo proposto.

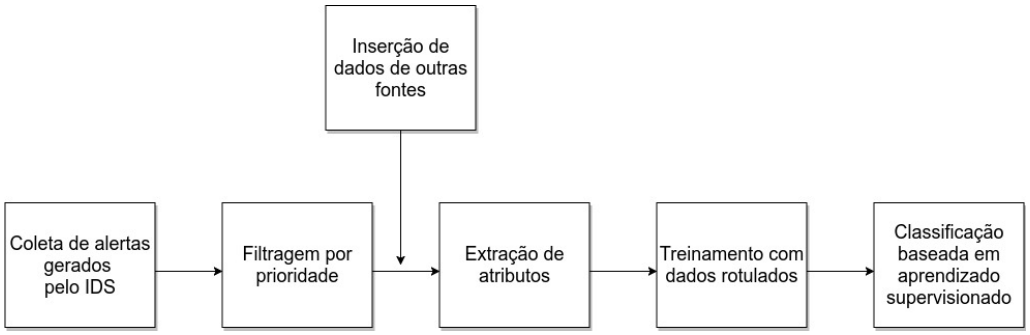


Figura 1. Modelo proposto para redução do número de alertas.

O primeiro passo do modelo proposto é a coleta dos alertas gerados pelo IDS. Logo após coletá-los, os alertas serão filtrados de acordo com a prioridade deles. Existe uma classificação dos alertas que permite aos administradores de sistemas um entendimento rápido e uma priorização mais efetiva de cada evento. Esta classificação normalmente é feita pelos IDS com base na assinatura do alerta. No IDS Snort[1], por exemplo, cada alerta tem uma prioridade determinada por um valor: 1 para prioridade alta, 2 para prioridade média, e 3 para prioridade baixa.

[1]https://www.snort.org/

 © 2017 Sociedade Brasileira de Computação

Na prioridade 1, os alertas representam uma ameaça muito grande ao sistema, precisando de uma análise imediata. A prioridade 2 é considerada importante, podendo se tornar uma ameaça caso não seja analisada e tratada. Na maioria dos casos, pode-se desconsiderar os alertas de prioridade 3 em uma primeira análise, pois são aqueles que não representam uma ameaça, mas sim uma característica mais informativa sobre o estado do sistema. A Figura 2 mostra um exemplo de alertas com prioridade baixa, que reportam uma tentativa de comunicação com a porta 80 do servidor de autenticação do *wireless access point*, gerando alertas com assinatura *(portscan) TCP Portsweep* com prioridade igual a 3. Estes alertas foram gerados para a rede utilizada no estudo de caso apresentado na Seção 4. Este tipo de evento é muito típico e normalmente está relacionado à solicitação de um *host* para que suas credenciais sejam verificadas durante a conexão com o *access point*. Mesmo que haja uma intenção maliciosa, esta tentativa de conexão tem um baixo grau de risco.

```
04/17/2016-16:52:23.705315 [**] [122:3:0] (portscan) TCP Portsweep [**]
[Classification: Attempted Information Leak] [Priority: 3] {TCP}
10.0.4.172:3556 -> 10.0.0.1:80

04/17/2016-16:54:23.913522 [**] [122:3:0] (portscan) TCP Portsweep [**]
[Classification: Attempted Information Leak] [Priority: 3] {TCP}
10.0.5.100:3447 -> 10.0.0.1:80
```

Figura 2. Exemplos de alertas de prioridade 3.

Na filtragem por prioridade, o nosso modelo prevê que todos os alertas de uma determinada prioridade sejam separados do conjunto de alertas que será submetido ao processo de busca por falsos positivos a ser realizado no próximo passo.

Depois de filtrar os alertas de acordo com a prioridade definida pelo IDS, se inicia o processo de identificação de falsos positivos utilizando aprendizado de máquina supervisionado. O primeiro passo neste processo é inserir novas informações de outras fontes à base de alertas que serão analisados. Esta tarefa é realizada pelo administrador da rede. Para demonstrar a importância de inserir informações de outras fontes à base de alertas, tomaremos como exemplo um alerta gerado pelo IDS utilizado no estudo de caso apresentado na Seção 4. O alerta é apresentado na Figura 3.

```
04/17/2016-13:52:23.053985 [**] [1:2001219:20] ET SCAN Potential SSH
Scan [**] [Classification: Attempted Information Leak] [Priority: 2]
{TCP} 218.93.17.146:38454 -> [redacted]:22
```

Figura 3. Exemplo de alerta gerado por um IDS Snort.

A Tabela 1 detalha quais são as informações providas no alerta. São informações básicas, que indicam algumas características do evento, mas não conseguem mostrar sozinhas, por exemplo, qual é a localização geográfica dos *hosts* envolvidos no ataque.

Buscando informações em outras fontes, é possível descobrir mais alguns detalhes sobre o evento que podem ajudar a identificar se o alerta é falso ou não. Utilizando um recurso de geolocalização de endereços IP como o GeoIP2 [2], pode-se extrair informações

[2]https://www.maxmind.com/pt/geoip-demo

© 2017 Sociedade Brasileira de Computação

Tabela 1. Descrição do alerta.

Atributo	Valor
Mês / Dia / Ano	04/17/2016
Hora / Minuto / Segundo	13:52:23.053985
Código da Assinatura	[1:2001219:20]
Assinatura	ET SCAN Potential SSH Scan
Classificação	[Classification: Attempted Information Leak]
Prioridade	[Priority: 2]
Protocolo	TCP
Endereço de IP de Origem	218.93.17.146
Porta de Origem	38454
Endereço de IP de Destino	(Endereço IP do Servidor Web da faculdade)
Porta de Destino	22

mais detalhadas sobre o endereço IP do atacante, como cidade, país, latitude, longitude, raio de ação, provedor de Internet e organização. Como estas informações estão relacionadas ao endereço IP, é importante considerar que existe a possibilidade do endereço IP ser falsificado pelo atacante. A Tabela 2 apresenta estas informações para o endereço IP 218.93.17.146. Este endereço IP é apontado como endereço do atacante no alerta da Figura 3.

Tabela 2. Geolocalização de endereço IP.

Atributo	Valor
Código do País	CN
Localização	Changzhou, Jiangsu, China, Ásia
Coordenadas aproximadas (Latitude e Longitude)	31.7833, 119.9667
Raio de ação (em Km)	50
Internet Service Provide (ISP)	China Telecom
Organização	Liyang Hongkou Primary School
Fonte: https://www.maxmind.com/pt/geoip-demo	

Os atributos destacados na Tabela 2 podem melhorar a qualidade das informações contidas no alerta. Neste caso, o endereço IP 218.93.17.146 está localizado na China, província de Jiangsu. A princípio, muitos analistas de segurança pensariam que, pelo endereço IP, o ataque teria sido realizado por algum invasor tentando conexão SSH na porta 22, utilizando um serviço de *Proxy* externo ou a Rede Tor, dificultando sua localização exata. Porém, o provedor do serviço de Internet é a China Telecom, e a organização é uma escola primária denominada Liyang Hongkou. De acordo com consulta no Google Maps[3], a escola realmente existe e as coordenadas do endereço IP coincidem com a localização física. Assim, levanta-se também a possibilidade de que o invasor tenha comprometido um sistema desta escola, passando a utilizar este endereço IP. De qualquer forma, todas as características ajudam a mostrar que o alerta é verdadeiro, já que a rede alvo pertence a uma faculdade brasileira, cujos serviços não são normalmente acessados por *hosts* na China, de acordo com seus administradores. Em nossa proposta, bus-

[3]https://www.google.com.br/maps

 © 2017 Sociedade Brasileira de Computação

camos, além de informações sobre a localização geográfica dos endereços IP, informações sobre a topologia da rede, determinando quais dos *hosts* envolvidos fazem parte da rede alvo, e caso façam parte, se eles são servidores ou não.

Considerando os dados dos próprios alertas e as informações adicionais, extraímos os seguintes atributos para cada alerta.

- Atributos referentes ao período - contribuem na identificação de alertas falsos conforme o horário da ocorrência do alerta. É atribuído um valor lógico para estes atributos (0 ou 1):
  - *manhã*: indica se o alerta foi gerado entre 6h00 e 11h59;
  - *tarde*: indica se o alerta foi gerado entre 12h00 e 17h59;
  - *noite*: indica se o alerta foi gerado entre 18h00 e 23h59;
  - *madrugada*: indica se o alerta foi gerado entre 0h00 e 5h59;
- Atributos referentes ao tipo de ataque - auxiliam no entendimento da natureza do ataque. Determinadas assinaturas e prioridades podem apresentar uma possibilidade maior de ter alertas falsos:
  - *assinatura*: código da assinatura presente no alerta;
  - *prioridade*: prioridade atribuída pelo IDS ao alerta;
- Atributos referentes à localização física do *host* e aos serviços atacados- a ideia é identificar *hosts* e serviços de rede que têm maior propensão a gerar alertas falsos e usar esta informação para classificar os alertas:
  - *porta_origem*: porta de origem do ataque presente no alerta;
  - *porta_destino*: porta de destino do ataque presente no alerta;
  - *parte1_end_origem*: 1º octeto do endereço IP de origem;
  - *parte2_end_origem*: 2º octeto do endereço IP de origem;
  - *parte3_end_origem*: 3º octeto do endereço IP de origem (o quarto octeto não foi necessário, pois na maioria dos casos os três primeiros já permitem a identificação da sub-rede);
  - *parte1_end_dst*: 1º octeto do endereço IP de destino;
  - *parte2_end_dst*: 2º octeto do endereço IP de destino;
  - *parte3_end_dst*: 3º octeto do endereço IP de destino (o quarto octeto não foi necessário, pois na maioria dos casos os três primeiros já permitem a identificação da sub-rede);
  - *país_origem*: indica se o endereço IP de origem está localizado no mesmo país da rede monitorada pelo IDS;
  - *país_dst*: indica se o endereço IP de destino está localizado no mesmo país da rede monitorada pelo IDS;
- Atributos referentes à função do *host* - auxilia na identificação de alertas verdadeiros e falsos com base na funcionalidade do *host* de origem ou destino. Por meio da análise do endereço IP, identifica se o *host* de origem ou destino pertence à rede monitorada. Em caso positivo, é possível verificar se ele é um servidor da rede interna ou não. O campo foi preenchido a partir da análise dos endereços IP e do conhecimento sobre a alocação destes endereços na rede:
  - *origem_servidor*: a partir da análise do endereço IP de origem, indica se a origem é um servidor da rede monitorada;
  - *dst_servidor*: a partir da análise do endereço IP de destino, indica se o destino é um servidor da rede interna;

© 2017 Sociedade Brasileira de Computação

– *origem_host_externo*: a partir da análise do endereço IP de origem, indica se a origem é um equipamento externo à rede monitorada;
– *dst_host_externo*: a partir da análise do endereço IP de destino, indica se o destino é um equipamento externo à rede monitorada.

Na etapa de treinamento dos dados, os algoritmos supervisionados construirão um modelo de classificação com base nos alertas rotulados pelo administrador de rede como falsos ou verdadeiros. Nos experimentos que serão apresentados na Seção 4, os algoritmos *Random Forest* e *kNN* serão utilizados no modelo para que os seus desempenhos sejam comparados. O último passo do modelo é a classificação feita pelos algoritmos sobre o conjunto de dados definido no passo anterior. Estes algoritmos, com base na fase de treinamento, classificarão cada alerta como falso ou verdadeiro.

4. Resultados e Discussões

Nesta seção, será apresentado um estudo de caso realizado em uma rede corporativa real para avaliar o modelo de redução de alertas proposto neste trabalho. A topologia de redes de computadores onde foi aplicado este estudo é representada na Figura 4.

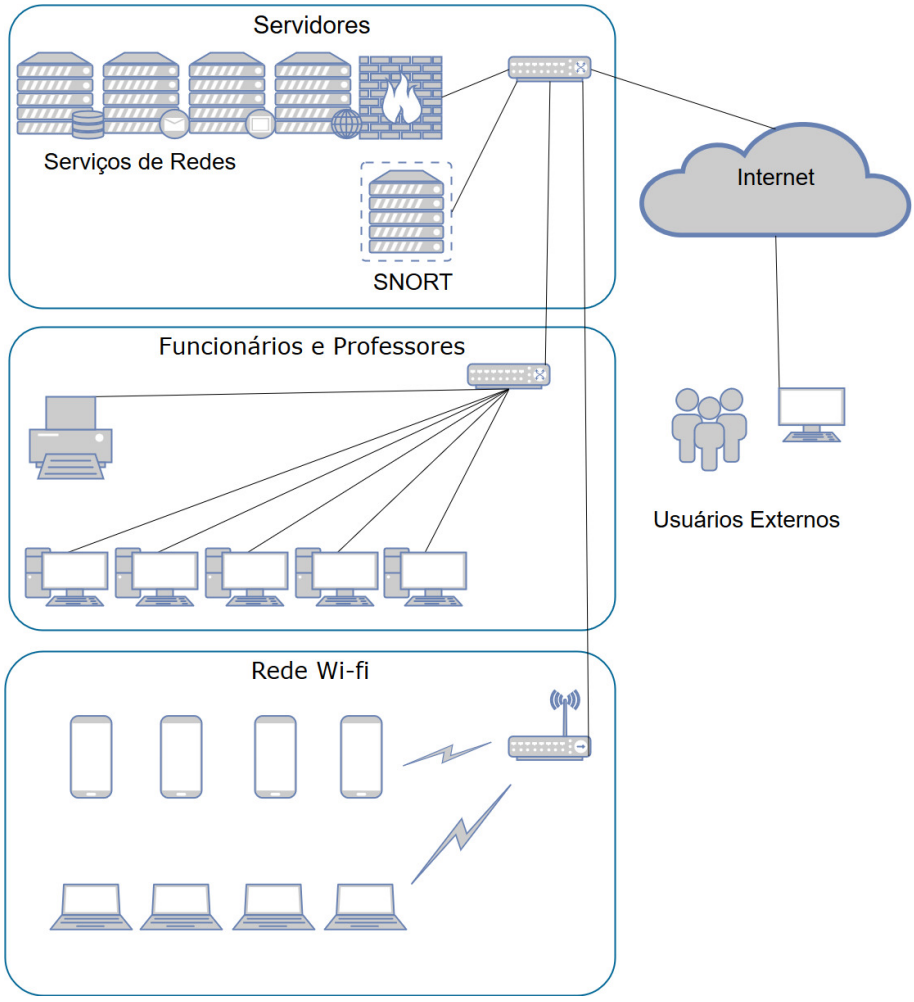


Figura 4. Topologia da rede de computadores.

© 2017 Sociedade Brasileira de Computação

A rede de computadores visualizada na Figura 4 pertence a Faculdade de Tecnologia de Ourinhos (FATEC-OU). Ela é acessada por cerca de 1750 usuários, entre alunos, professores e funcionários, tanto internamente por uma rede sem fios e pela rede cabeada dos departamentos, como externamente. Os servidores disponibilizam principalmente serviços Web, como a página principal da faculdade, o *blog*, serviços acadêmicos, materiais de ensino e bancos de dados acadêmicos.

Com base nas estatísticas mensais, nestes servidores são estabelecidas cerca de 1.000 conexões diárias. Na rede monitorada, há um IDS Snort instalado, cujas assinaturas são atualizadas pelo próprio sistema, que gera em média 230.000 alertas diários, ou seja, mais de 2 alertas por segundo.

Para o estudo de caso realizado neste trabalho, foram utilizados os alertas referentes ao mês de março de 2016 gerados pelo IDS Snort, pois neste mês ocorreram dois fatos importantes que geraram certas mudanças no comportamento dos alertas. O primeiro fato foi o início das atividades práticas nos laboratórios de informática da faculdade para o primeiro semestre de 2016, ocorrido no início da terceira semana deste mês, o que costuma propiciar ações inadequadas na rede e um grande aumento do número de alertas. O segundo fato foi a ocorrência de um feriado prolongado na quarta semana, causando a diminuição das atividades da rede e alterando o volume e as características dos alertas. A divisão por semanas ocorreu para se obter resultados mais precisos, pois se a análise fosse feita no mês como um todo, não se conseguiria observar se fatos como início das aulas práticas e feriados teriam algum impacto nos resultados. A Tabela 3 mostra a quantidade de alertas por prioridade gerados neste mês.

Tabela 3. Total de alertas por prioridade referente ao mês de Março de 2016

	Prioridade 1	Prioridade 2	Prioridade 3	Total geral
Semana 1	239	930	26.704	27.873
Semana 2	601	4.124	275.168	279.893
Semana 3	9.878	2.466	1.302.703	1.315.047
Semana 4	445	4.524	284.177	289.146
Semana 5	479	3.095	296.459	300.033
TOTAL	11.642	15.139	2.185.211	2.211.992

O primeiro passo do modelo proposto é a filtragem de todos os alertas com uma determinada prioridade. Em nosso estudo de caso, filtramos todos os alertas com prioridade 3, pois são aqueles que representam normalmente uma característica informativa sobre o estado do sistema. Cerca de 91% destes alertas são do tipo *Portsweep*, ou seja, tratam de uma varredura por portas abertas na rede. Normalmente, na rede monitorada, este alerta é gerado quando um dispositivo móvel de um aluno, funcionário ou professor solicita uma autenticação ao *access point* para poder se conectar à Internet. Os outros 9% de alertas são majoritariamente dos tipos:

- *ICMP test detected*: evento de segurança gerado quando um teste é realizado para verificar se o servidor está respondendo;
- *Protocol mismatch*: evento de segurança gerado quando um teste é realizado para verificar se um serviço está respondendo em uma determinada porta;

© 2017 Sociedade Brasileira de Computação

Seguindo o modelo proposto, foram adicionadas aos alertas informações de outras fontes de dados sobre os eventos de segurança como a localização geográfica do endereço IP de origem e de destino, e a função do endereço IP na rede analisada. Então, foi realizada a extração dos atributos elencados na Seção 3, formando o conjunto de dados que será analisado pelos algoritmos de classificação supervisionada. Um algoritmo supervisionado trabalha em duas etapas. Primeiro, há um treinamento no qual o algoritmo recebe como entrada um conjunto de dados rotulados. Neste treinamento, o algoritmo constrói um modelo de classificação com base nos dados rotulados. No estudo de caso, os alertas foram rotulados como verdadeiros ou falsos pelos profissionais da administração de redes da faculdade. Na segunda etapa, o classificador usa o modelo para classificar novas instâncias de dados que são apresentadas a ele. Utilizamos dois algoritmos de classificação supervisionada: *kNN* e *Random Forest*.

O *kNN* classifica uma instância com base em seus k vizinhos mais próximos. Para definir a distância entre uma instância e seus vizinhos, é necessário usar uma métrica como a distância Euclidiana. Uma vez que o algoritmo encontra os k vizinhos mais próximos, ele realiza um processo de votação. A classe que tiver mais instâncias entre os k vizinhos mais próximos é eleita como a classe da instância em análise. O algoritmo *Random Forest*, por sua vez, combina diversas árvores de decisão, sendo então um algoritmo de *ensemble learning*. Cada árvore é formada a partir de um conjunto de atributos obtidos aleatoriamente e instâncias amostradas de acordo com a técnica *bootstraping*. Durante a classificação, a instância a ser analisada é apresentada a todas as árvores e cada uma delas vota em uma classe como a mais adequada. A classe que obtiver mais votos será escolhida como a classe para a instância em questão.

Para medir o desempenho do classificador, foram utilizadas as seguintes métricas:

- Precisão: no conjunto de alertas classificados como verdadeiros, esta métrica mostra quantos realmente eram verdadeiros;

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

- Sensibilidade: considerando o conjunto de alertas que realmente eram verdadeiros, esta métrica mostra quantos destes alertas foram classificados corretamente;

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

- Especificidade: considerando o conjunto de alertas que realmente eram falsos, esta métrica mostra quantos destes alertas foram classificados corretamente;

$$Specificity = \frac{TN}{TN + FP} \tag{3}$$

- F-score: avalia a cobertura e a precisão da classificação, considerando a média ponderada da precisão e da sensibilidade. Quanto mais próxima de 1, melhores são a cobertura e a precisão da classificação;

$$F - Score = \frac{2 * (Precision * Recall)}{(Precision + Recall)} \tag{4}$$

 © 2017 Sociedade Brasileira de Computação

TP, FP, TN, e FN correspondem respectivamente a *true positive* (positivo verdadeiro), *false positive* (falso positivo), *true negative* (negativo verdadeiro), e *false negative* (falso negativo).

Para avaliar os dois classificadores escolhidos, utilizamos o método de validação cruzada. Este método tem como proposta particionar o conjunto de dados em n subconjuntos de mesmo tamanho. A partir deste ponto, um dos subconjuntos é utilizado para teste e o restante para treinamento. As rodadas de avaliação são repetidas n vezes e em cada uma destas rodadas um sub-conjunto diferente é usado para teste e os outros são usados para treinamento. Aplicamos o método de validação cruzada em cada uma das semanas do mês de março de 2016. Em cada semana, dividimos os alertas em 7 subconjuntos, isto é, $n = 7$. A Figura 5 mostra os resultado obtidos com a aplicação do algoritmo *kNN*.

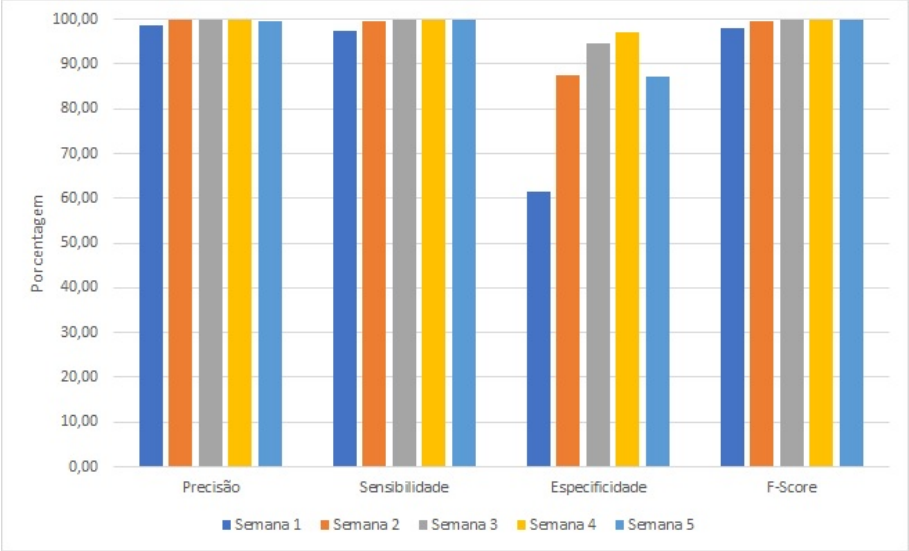


Figura 5. Resultados para o algoritmo kNN.

Na Figura 5, podemos observar que os resultados obtidos com o algoritmo de aprendizado de máquina *kNN* foram próximos para todas as semanas, com exceção da semana 1. Nesta semana, a métrica Especificidade (equação 3) teve o resultado de 61,54%, diferente das outras semanas que variaram entre 87,34% e 97,19%. Este resultado indica que na semana 1 o *kNN* encontrou dificuldades em classificar corretamente os alertas falsos, apontando erroneamente que alguns seriam verdadeiros. Este fato pode ter ocorrido em decorrência da quantidade de alertas falsos positivos ser muito pequena (53 alertas falsos positivos) quando comparada com a quantidade de alertas verdadeiros (1.116 alertas verdadeiros). O desbalanceamento entre as classes pode ser um obstáculo para o *kNN*. Apesar deste resultado ruim para a semana 1, nota-se que os resultados para todas as métricas nas outras semanas foram bons, ficando sempre próximos a 100%.

A Figura 6 mostra os resultados obtidos com a aplicação do algoritmo *Random Forest*.

Na Figura 6, os resultados obtidos com o *Random Forest* foram praticamente iguais para todas as semanas, sem exceção. Todas as métricas, para todas as semanas se situam entre 95,08% e 100%. O *Random Forest* conseguiu superar o *kNN* na semana

© 2017 Sociedade Brasileira de Computação

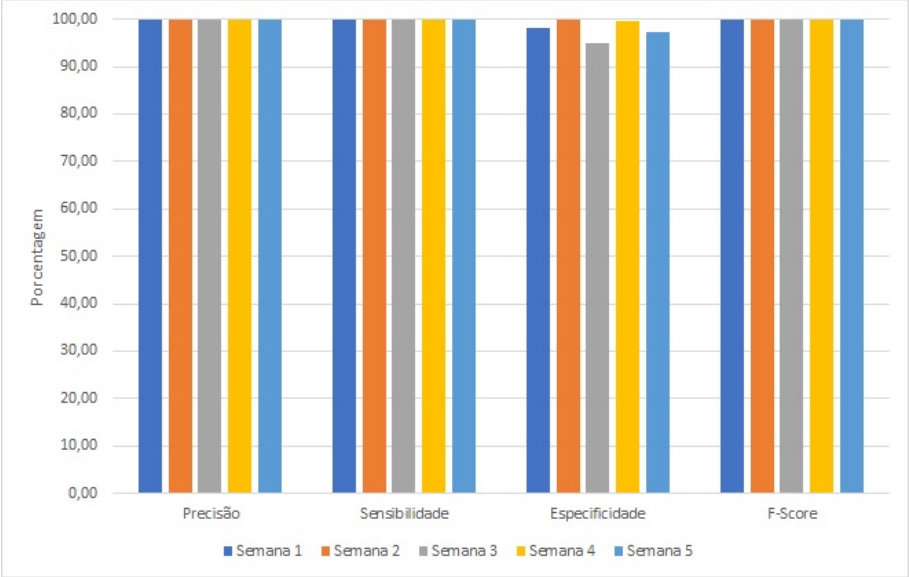


Figura 6. Resultados para o algoritmo Random Forest.

1, quando este último teve um resultado ruim para a Especificidade. No caso do *Random Forest*, mesmo com o desbalanceamento das classes para a semana 1, a Especificidade alcançada foi bastante alta. Isto ocorre porque o *Random Forest*, ao ser um algoritmo de *ensemble learning*, é mais robusto para lidar com o desbalanceamento de classes. Em geral, os resultados do *Random Forest* foram levemente melhores que os obtidos com o *kNN*.

À seguir, serão apresentados um caso de alerta classificado acertadamente como alerta falso e também um caso de alerta classificado corretamente como alerta verdadeiro. O objetivo é mostrar dois casos típicos que ilustram o funcionamento da proposta. A Figura 7 exibe um alerta classificado corretamente como falso positivo.

```
03/09/2016-16:24:49.373444  [**]  [1:1201:7]  ATTACK-RESPONSES  403
Forbidden [**] [Classification: Attempted Information Leak] [Priority:
2] {TCP} [redacted]:80 -> [redacted]:53207
```

Figura 7. Alerta falso positivo.

O alerta exibido na Figura 7 se refere a uma resposta de um servidor Web interno da faculdade para outro servidor interno, responsável por hospedar o *blog* da faculdade. Esta resposta indica que uma página está proibida de ser acessada, por estar passando por alguma atualização ou inserindo um novo aviso interno para os alunos ou professores. Apesar de ser apenas uma resposta informativa, ela foi classificada pelo IDS como ataque. É importante observar que os atributos que extraímos para cada alerta ajudam a mostrar que o alerta é falso. Como exemplo, podemos destacar os atributos *origem_servidor* e *dst_servidor*, que neste caso mostram que o alerta retrata um suposto ataque entre dois servidores da rede da faculdade. Este tipo de ocorrência, apesar de possível, é bastante incomum no ambiente em questão, como pode ser verificado durante o treinamento dos modelos supervisionados. Desta forma, as informações ajudam o classificador a determinar corretamente que o alerta é falso.

 © 2017 Sociedade Brasileira de Computação

A Figura 8 exibe uma alerta classificado corretamente como verdadeiro.

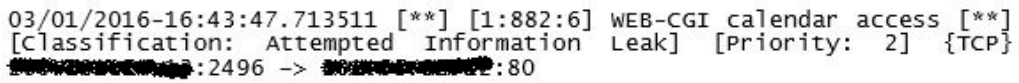


Figura 8. Alerta verdadeiro.

O alerta exibido na Figura 8 é referente a um ataque realizado por um *host* externo, localizado na cidade de Jacarezinho, no estado do Paraná. O atacante tenta executar um *script* de manipulação de calendário escrito em linguagem *Perl*, que permite a execução de comandos sem verificação de entrada de dados. O modelo que propomos neste trabalho classificou esta ocorrência como ataque ao analisar características como o *host* de origem, sua localização física e a assinatura do alerta.

Finalmente, podemos concluir que o modelo proposto para a redução de alertas obteve sucesso. Ambos os algoritmos de classificação supervisionada, destacando o *Random Forest* por sua robustez frente ao desbalanceamento de classes, mostraram ótimos resultados na identificação de falsos positivos. Antes, os dados brutos somavam 2.211.992 alertas para o mês de março de 2016. Com a filtragem dos alertas de baixa prioridade (prioridade 3), este número caiu para 26.781 alertas. Após a inserção de dados de outras fontes e a aplicação do algoritmo de aprendizado supervisionado *Random Forest*, foram identificados 784 alertas falsos, cerca de 3% dos alertas de prioridades 1 e 2. Ao final da aplicação do modelo proposto, ao invés de analisar 2.211.992 alertas, os administradores poderiam analisar 25.997 alertas em um mês, sem se preocupar com alertas falsos.

5. Conclusão

Este trabalho apresentou um modelo de redução de alertas falsos positivos por meio de filtragem de prioridades de alertas e técnicas de aprendizado de máquina utilizando um classificador supervisionado. Dois algoritmos foram utilizados e observou-se que ambos atingiram bons resultados: *kNN* e *Random Forest*. Dos algoritmos, o *Random Forest* foi o algoritmo que apresentou, de modo discreto, os melhores resultados quando comparado com o *kNN*, devido à sua capacidade de lidar melhor com desbalanceamento entre as classes. A filtragem de alertas por prioridade e a inserção de dados de outras fontes foram importantes no desempenho dos classificadores. Com o conhecimento sobre a topologia e as funções dos *hosts* internos da rede de computadores, pode-se destacar características mais específicas de cada evento de segurança gerado pelo IDS Snort como: função do *host* de origem e de destino, localização física do *host* e comportamento de *hosts* de dentro e fora do país. Selecionando os alertas relevantes para o classificador e filtrando os alertas que apresentaram baixa prioridade, os algoritmos, de modo geral, apresentaram um bom desempenho na identificação de alertas falsos positivos e, consequentemente, alcançaram os resultados esperados.

Para os trabalhos futuros, pretende-se propor métodos para correlacionar os alertas resultantes do processo de redução de forma a mostrar os cenários de ataque aos administradores de rede de forma mais amigável.

© 2017 Sociedade Brasileira de Computação

Referências

Alvarenga, S. C. d., Zarpelão, B. B., and Miani, R. S. (2015). Discovering attack strategies using process mining. In *The Eleventh Advanced International Conference on Telecommunications*, pages 119–125.

CERT.br − Centro de Estudos, R. e. T. d. I. d. S. n. B. (2015). *Estatísticas dos Incidentes Reportados ao CERT.br.* Disponível em: https://www.cert.br/stats/incidentes/ . Acessado em Junho de 2017. CERT.br.

Ebrahimi, A., Navin, A. H. Z., Mirnia, M. K., Bahrbegi, H., and Ahrabi, A. A. A. (2011). Automatic attack scenario discovering based on a new alert correlation method. In *Systems Conference (SysCon), 2011 IEEE International*, pages 52–58. IEEE.

Elshoush, H. T. I. (2014). An innovative framework for collaborative intrusion alert correlation. In *Science and Information Conference (SAI), 2014*, pages 607–614. IEEE.

Granadillo, G. G., El-Barbori, M., and Debar, H. (2016). New types of alert correlation for security information and event management systems. In *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*, pages 1–7. IEEE.

Julisch, K. and Dacier, M. (2002). Mining intrusion detection alarms for actionable knowledge. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375. ACM.

Kawakani, C. T., Junior, S. B., Miani, R. S., Cukier, M., and Zarpelão, B. B. (2016). Intrusion alert correlation to support security management. In *Proceedings of the XII Brazilian Symposium on Information Systems on Brazilian Symposium on Information Systems: Information Systems in the Cloud Computing Era-Volume 1*, page 42. Brazilian Computer Society.

Kurose, J. F., Ross, K. W., Marques, A. S., and Zucchi, W. L. (2010). *Redes de Computadores ea Internet: uma abordagem top-down*. Pearson.

Shittu, R., Healing, A., Bloomfield, R., and Muttukrishnan, R. (2012). Visual analytic agent-based framework for intrusion alert analysis. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on*, pages 201–207. IEEE.

Shittu, R., Healing, A., Ghanea-Hercock, R., Bloomfield, R., and Muttukrishnan, R. (2014). Outmet: A new metric for prioritising intrusion alerts using correlation and outlier analysis. In *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pages 322–330. IEEE.

Verma, R., Kantarcioglu, M., Marchette, D., Leiss, E., and Solorio, T. (2015). Security analytics: essential data analytics knowledge for cybersecurity professionals and students. *IEEE Security & Privacy*, 13(6):60–65.

Vidal, J. M., Orozco, A. L. S., and Villalba, L. J. G. (2015). Quantitative criteria for alert correlation of anomalies-based NIDS. *IEEE Latin America Transactions*, 13(10):3461–3466.

 © 2017 Sociedade Brasileira de Computação

Uma arquitetura para detecção de *botnet*s baseada na análise do tráfego DNS descartado

Luiz Gonzaga Mota Barbosa, Joaquim Celestino Júnior, André Luiz Moura dos Santos

[1]Universidade Estadual do Ceará (UECE)
Fortaleza/CE

Abstract. *Botnets are one of the main threats on the Internet and are able to assist malicious activities. After a host infection, a bot tries to communicate with its command and control servers. At this point, recent* bots *use Domain Generation Algorithms to create a list of candidate domain names for command and control servers. One consequence of this behavior is the increase in negative DNS answers. This traffic, usually discarded or ignored by network administrators, can be used to model a* botnet *behavior. Based on the increase in the number of negative DNS answers and having access to samples collected from a controled environment, this paper discusses an architecture capable of generating detection models to these* bots*. The tests showed an acuracy level above 90% in most test cases.*

Resumo. Botnets *são uma das principais ameaças na Internet, capazes de auxiliar na realização de atividades maliciosas. Ao infectar um hospedeiro, um* bot *tenta comunicar-se com o servidor de comando e controle. Neste ponto,* bots *recentes utilizam Algoritmos de Geração de Domínios para criarem uma lista de nomes de domínio candidatos a servidores de comando e controle. Uma consequência deste comportamtento é o aumento de respostas negativas do protocolo DNS. Esse tráfego, geralmente descartado ou ignorado pelos administradores de rede, pode ser utilizado para modelar o comportamento de uma* botnet*. Com base no aumento no número de respostas negativas do protocolo DNS e acesso a amostras coletadas em um ambiente controlado, este trabalho apresenta uma arquitetura capaz de gerar modelos de detecção para estes* bots*. Os experimentos mostraram um nível de acurácia acima de 90% na maioria dos cenários.*

1. Introdução

Recentemente, o mundo observou a onda de ataques do malware *WannaCry*, o qual utilizava o protocolo DNS para controlar a sua disseminação [Khandelwal 2017]. Caso o domínio requisitado, respondesse as requisições, o ataque seria abortado. Enquanto o domínio não fosse registrado pelo criador do *malware*, haveria um aumento de respostas negativas do protocolo DNS na rede infectada.

Uma classe específica de *malware*, permite que o atacante controle remotamente as máquinas infectadas. O conjunto de máquinas infectadas por um mesmo *malware* desta classe compõe uma *botnet*. Uma das primeiras ações executadas pelo *malware* após uma infecção é tentar localizar seu servidor de comando e controle. Para isso, *bots* mais recentes tem utilizado o protocolo DNS, desfrutando das mesmas vantagens que usuários legítimos.

 © 2017 Sociedade Brasileira de Computação

Uma maneira de contactar seus servidores é carregar o domínio destes em seu código. Porém, para dificultar a detecção a partir de técnicas de engenharia reversa, desenvolvedores de *bots* passaram a utilizar Algoritmos de Geração de Domínios (*Domain Generation Algorithms - DGAs*) com a finalidade de criar uma lista de candidatos a servidores de Comando e Controle (C&C) para o estabelecimento de sua comunicação [Stone-Gross et al. 2009, Bilge et al. 2011, Antonakakis et al. 2012].

A utilização de DGAs ocasiona um aumento na quantidade de respostas negativas do protocolo DNS na rede e esse tráfego é normalmente descartado. Entretanto, esse tráfego pode ser utilizado para modelar a presença de infecções na rede e gerar perfis de comportamento dos *bots* presentes na rede.

Este trabalho apresenta uma arquitetura para detecção de *botnet*s com base na análise so tráfego DNS que seria descartado pela rede, em especial daquelas que utilizam DGAs para a comunicação com seus servidores de C&C.

O restante deste trabalho está organizado da seguinte maneira: a Seção 2 apresenta os trabalhos relacionados; a Seção 3 define o que são *botnets*, suas principais características, seu relacionamento com o protocolo DNS e os aspectos gerais para sua detecção; a Seção 4 apresenta a arquitetura utilizada neste trabalho; a Seção 5 descreve a implementação e experimentos, resultados obtidos e análise dos mesmos; e, por fim, a conclusão na Seção 6.

2. Trabalhos Relacionados

[Holz et al. 2008] discutem uma técnica utilizada por *botnets* chamada Fast-Flux Service Network (FFSN). Esta pode ser resumida como uma versão maliciosa do Round-Robin DNS (RRDNS), na qual são utilizados valores de *Time-To-Live (TTL)* relativamente pequenos quando comparados aos valores utilizados para domínios não-maliciosos. Dessa forma, a lista de endereços IP presentes na resposta, os hospedeiros que fazem parte do ”fluxo”, é atualizada mais rapidamente [Salusky and Danford 2007].

[Stone-Gross et al. 2009] foram os primeiros a discutir outra técnica utilizada por *botnets* para o estabelecimento de sua comunicação: Algoritmos de Geração de Domínios (*Domain Generation Algorithms - DGA*). Com ela, *bots* geram uma lista de nomes de domínio candidatos a servidor de C&C a cada tentativa de comunicação ou dado um intervalo determinado pelo botmaster. A primeira máquina que responder a essas requisições será considerado um servidor de C&C válido.

[Bilge et al. 2011] destacam a importância do protocolo DNS para serviços não-maliciosos e maliciosos e, portanto, sua utilização no processo de detecção de atividades maliciosas. Sua principal hipótese é a de que, ao analisar grandes quantidades de dados do tráfego de uma rede, requisições DNS maliciosas e não-maliciosas exibirão padrões de comportamento diferentes e que poderão ser detectadas automaticamente. Sobre essa diferenciação de padrões na utilização do protocolo DNS, [Antonakakis et al. 2012] destacam que *bots* que utilizam DGAs geram um aumento na quantidade de respostas DNS negativas na rede e este comportamento pode ser utilizado para a geração de modelos de detecção.

Baseado na importância do protocolo DNS na realização de atividades maliciosas e nas técnicas utilizadas por *malwares*, este trabalho concentra sua análise no tráfego DNS

 © 2017 Sociedade Brasileira de Computação

de uma rede monitorada.

Trabalhos como [Schiavoni et al. 2014] e [Erquiaga et al. 2016] buscam identificar as ameaças através da modelagem de comportamentos maliciosos e não-maliciosos. Abordagens como estas quando aplicadas a ambientes muito heterogêneos podem exigir uma carga computacional grande para modelar todos os possíveis comportamentos, maliciosos e não-maliciosos, ali presentes. Portanto, diferente destes trabalhos, aqui é utilizado apenas o tráfego DNS que seria descartado pela rede, diminuindo assim a quantidade de informação a ser processada pelo sistema de detecção de ameaça e, consequentemente, apenas comportamentos tidos como maliciosos serão modelados.

3. Fundamentação Teórica

3.1. Botnets

Botnets são grupos de máquinas comprometidas por *malwares*, conhecidas como bots ou zumbis, controladas remotamente por um indivíduo chamado *botmaster* através de um canal de comunicação de comando e controle (C&C) [Cisco 2007, Antonakakis et al. 2012].

Um mesmo bot pode apresentar características de diferentes tipos de *malware*s tornando as *botnet*s capazes de realizar diversas atividades maliciosas tais como: ataques de negação de serviço, roubo de informações sensíveis, envio massivo de e-mails (spams) entre outras [Gu et al. 2008, Cavallaro et al. 2009, Tiirmaa-Klaar et al. 2013]. Além disso, *bots* podem infectar diferentes tipos de máquinas vulneráveis espalhadas em uma mesma rede ou em redes distintas, como mostrado na Figura 1.

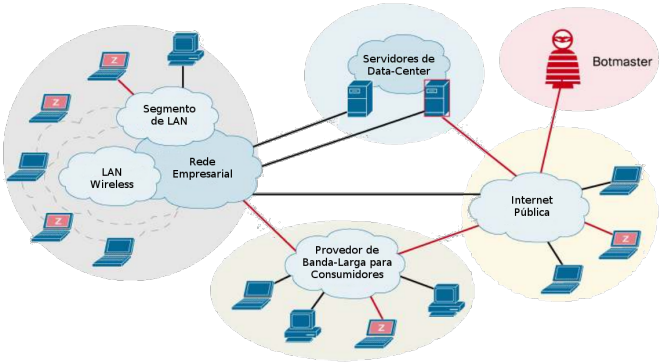


Figura 1. Ilustração de uma Botnet

A força de uma *botnet* está em possuir uma rede flexível de computadores que podem ser controlados remotamente [Karim et al. 2014], com isso, a escolha de uma arquitetura de comunicação é uma decisão importante do ponto de vista do botmaster, influenciando em sua capacidade de gerenciamento, resiliência e em alguns aspectos de seu funcionamento [Tiirmaa-Klaar et al. 2013].

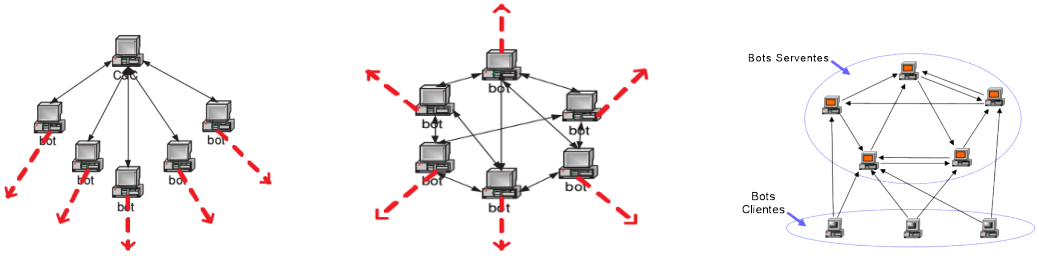
3.2. Arquitetura de uma Botnet

A busca por mecanismos que proporcionem uma infraestrutura flexível e confiável para sua *botnet* é uma questão importante enfrentada por desenvolvedores de *botnets*

© 2017 Sociedade Brasileira de Computação

[Bilge et al. 2011]. Sendo assim, diferentes arquiteturas foram utilizadas a fim de evitar contramedidas de bloqueio, mantendo a operabilidade da *botnet* pelo maior período possível [Tiirmaa-Klaar et al. 2013].

Quanto a sua estrutura de C&C, uma *botnet* pode ser projetada como uma arquitetura centralizada (Figura 2a), descentralizada (Figura 2b) ou híbrida (Figura2c).



(a) Centralizada. (b) P2P (c) Híbrida

Figura 2. Arquiteturas de uma *botnet*

3.3. Botnets e o Protocolo DNS

Devido a sua versatilidade, *bots* não sabem qual ação executar logo após a infecção, portanto, sua primeira ação é tentar estabelecer um canal de C&C com o botmaster a fim de registrar-se na *botnet* e então passar receber comandos, atualizações e trocar dados [Stone-Gross et al. 2009,].

Do ponto de vista do botmaster, é necessária alguma técnica que possibilite o estabelecimento da comunicação entre os membros da *botnet* e que os *bots* continuem invisíveis e portáveis [Choi et al. 2007]. Em resposta a essa necessidade, *bots* passaram a utilizar o protocolo DNS para localizar seus servidores de C&C e estabelecer este canal de comunicação entre estes [Stone-Gross et al. 2009, Bilge et al. 2011, Antonakakis et al. 2012].

O protocolo DNS, além de prover um mapeamento entre nomes de domínio e endereços IP, permite que um determinado hospedeiro possua mais de um nome (*host aliasing*) e que um mesmo domínio seja associado a mais de um endereço IP. Com isso, é possível que um determinado serviço seja distribuído entre servidores diferentes e, em caso de falha de algum destes servidores, outro poderá responder às requisições ao serviço.

Com essas carcaterísticas, a utilização do protocolo DNS para a comunicação entre membros da *botnet* agilizou o processo de migração de *botnet*, mantendo assim sua disponibilidade e a operabilidade ainda que um ou mais de seus servidores de C&C tenha sido descoberto e banido da rede [Choi et al. 2007, Antonakakis et al. 2010].

As principais técnicas utilizadas por *botnet*s envolvento o protocolo DNS são: Fast-Flux Service Networks e Algoritmos de Geração de Domínios, apresentadas em [Holz et al. 2008] e [Stone-Gross et al. 2009], respectivamente.

3.4. Detecção de Botnets

Independente da arquitetura utilizada, *botnets* compartilham uma característica: os *bots* pertencentes a uma mesma *botnet*, tendem a executar um mesmo comportamento, pré-determinado em seu código por seus criadores [Gu et al. 2008].

 © 2017 Sociedade Brasileira de Computação

Quando capturado, um bot pode ser analisado estática e dinamicamente a fim de extrair suas características comportamentais a nível de hospedeiro e/ou de rede [Sikorski and Honig 2012,].

Através da utilização de técnicas de engenharia reversa, são extratídas assinaturas dos exemplares capturados. Porém, nem sempre é viável fazer a engenharia reversa de um exemplar de um bot, uma vez que eles podem ser programados para se modificarem a cada infecção ou ainda, sofrerem mutações como resposta a comandos do botmaster [Stone-Gross et al. 2009, Antonakakis et al. 2012].

Recentemente, o processo de detecção de *botnets* concentrou-se na análise de comportamento dos *bots* nas redes [Cavallaro et al. 2009, Gu et al. 2008, Bilge et al. 2012]. Em uma rede infectada, podem ser observados comportamentos baseados em: protocolo utilizado, quantidade de mensagens trocadas em um determinado período, variações geradas no tráfego da rede, acessos a *sites* marcados como maliciosos, entre outros.

A identificação de comportamentos maliciosos pode ocorrer através: da busca de evidências da infecção de *bots* ou da comunicação entre as máquinas e servidores de C&C, também conhecida como correlação vertical; ou através da correlacão de eventos de rede a fim de identificar hospedeiros envolvidos em atividades maliciosas similares, também conhecida como correlação horizontal.

O processo de análise de comportamento pode ocorrer também seguindo uma abordagem passiva ou ativa [Stone-Gross et al. 2009]. Em uma abordagem passiva, são estudados os efeitos secundários gerados pela atividade das máquinas infectadas. Já em uma abordagem ativa, os pesquisadores de fato buscam infiltrar-se na *botnet* através da utilização de um exemplar do *malware* ou um cliente simulando um bot.

3.5. Modelagem do comportamento malicioso

Uma vez detectada a presença de uma ou mais *botnets* em uma rede, são gerados modelos do comportamento dos *bots* identificados e em seguida esses modelos são aplicados à rede a fim de mitigar a ameaça. Para automatizar o processo de modelagem desses comportamentos, têm-se utilizado técnicas de aprendizagem de máquina [de A Ribeiro et al. 2011, Bilge et al. 2012, Zhao et al. 2013].

Para a modelagem do comportamento, é necessária a identificação de características capazes de representar o comportamento em questão. As características utilizadas podem ser classificadas como:

- **Características do fluxo de rede:** possibilitam uma identificação global dos padrões de comunicação dos *bots* quanto aos hospedeiros que estão se comunicando e o protocolo utilizado por eles;
- **Características de *payload*:** diretamente ligadas ao protocolo utilizado pelos *bots*, permitem um melhor entendimento do comportamento dos *bots*;
- **Características temporais:** utilizadas a fim de identificar padrões temporais na comunicação entre os membros da *botnet*.

Uma *botnet* pode fazer o uso de protocolos diferentes em fases diferentes de seu ciclo de vida e até dependendo de sua arquitetura. Também deve ser levado em consideração que os *bots* podem utilizar técnicas para dificultar sua detecção como: mascaramento de IP, polimorfismo e criptografia.

© 2017 Sociedade Brasileira de Computação

4. Uma arquitetura para a reciclagem do tráfego DNS

A principal hipótese deste trabalho é que *bots* que utilizam DGAs geram um aumento na quantidade de respostas negativas do protocolo DNS, respostas do tipo NXDOMAIN. Com isso, é possível utilizar este tráfego para identificação de domínios maliciosos bem como quais máquinas estão infectadas em uma rede monitorada.

Com base nessa hipótese, a arquitetura (Figura 3) a seguir realiza uma correlação vertical baseada na análise passiva do comportamento de *bots* quanto ao protocolo DNS, em especial daqueles que utilizam DGAs. Essa análise é realizada sobre o tráfego DNS que seria descartado, ou seja, concentra-se nos pacotes resposta DNS do tipo NXDOMAIN.

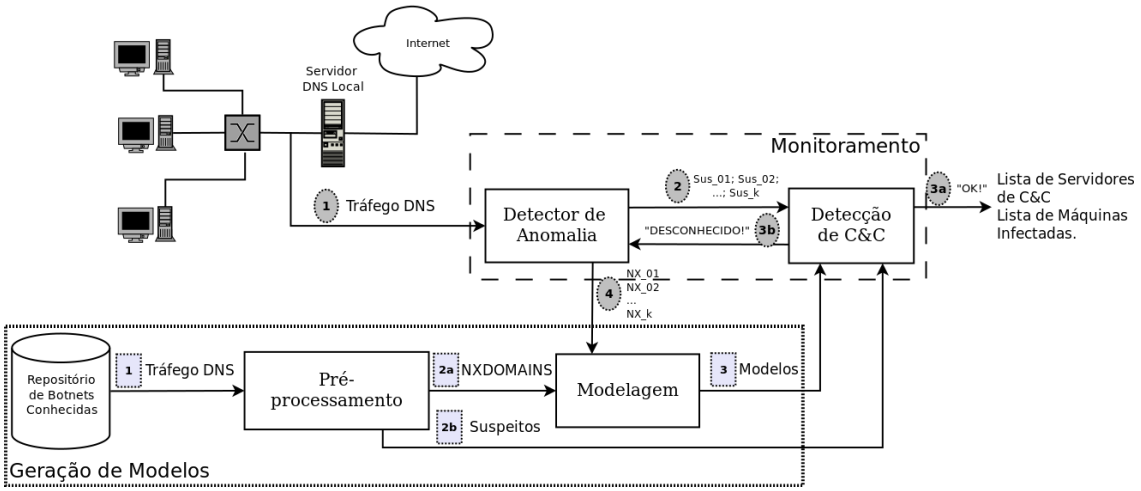


Figura 3. Visão geral e fluxo de informações na arquitetura

Na Figura 3, observa-se que a arquitetura está dividida em duas camadas: Monitoramento – composto por um módulo Detector de Anomalia e um módulo de Detecção de C&C; e Geração de Modelos, composto por uma base de conhecimento de tráfegos maliciosos previamente observados, um módulo de Pré-processamento e um módulo de Modelagem.

Primeiramente, o tráfego DNS é monitorado pelo Detector de Anomalia (Passo 1), este observa a rede monitorada a fim de identificar uma elevação nas taxas de tráfego NXDOMAIN, ao percerber tal comportamento anômalo, os domínios pertencentes a esse tráfego são agrupados e cada agrupamento é devidamente rotulado. As respostas de cada agrupamento são enviadas para a Detecção de C&C (Passo 2). Ao chegarem no módulo de Detecção de C&C, as respostas recebidas podem levar a um dos resultados a seguir:

- Ameaça detectada (”OK!”) – retorna os endereços IP dos servidores de C&C e a lista de máquinas infectadas (Passo 3a);
- Ameaça desconhecida (”DESCONHECIDO”) – padrão de comportamento ainda não modelado. O módulo envia um sinal para o módulo anterior (Passo 3b).

Seguindo o fluxo em caso de uma ameça ainda desconhecida, as respostas NXDOMAIN do agrupamento desconhecido são enviados para a camada de Geração de Modelos para que esta possa gerar um modelo para o novo comportamento (Passo 4).

 © 2017 Sociedade Brasileira de Computação

A camada de Monitoramento é executada na dentro da rede monitorada, dessa forma é possível identificar não somente a existência de infecções, mas também, quais máquinas foram contaminadas a fim de que estas sejam sanadas. A Figura 4 apresenta detalhes do módulo de Detecção de Anomalia e a Figura 5 do módulo de Detecção de C&C.

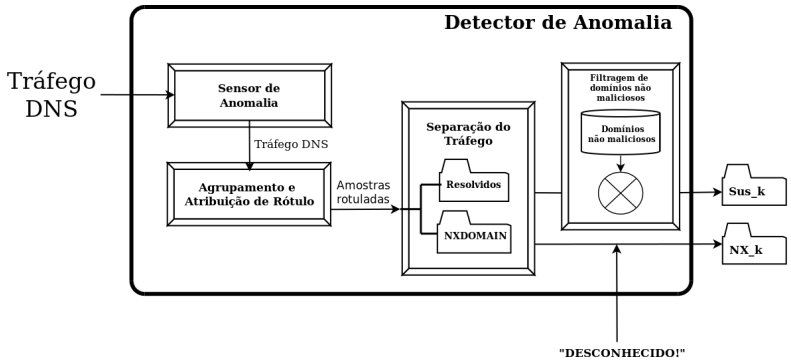


Figura 4. Módulo Detector de Anomalia

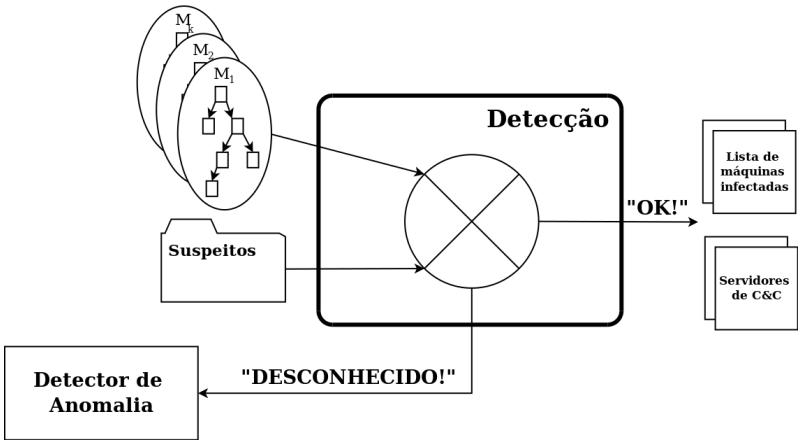


Figura 5. Módulo de Detecção de C&C

A camada de Geração de Modelos pode ser executada em um ambiente diferente do qual a camada anterior se encontra. Recomenda-se que este outro ambiente possua um poder computacional maior em relação ao Monitoramento tal que a geração dos modelos possa ocorrer o mais breve. O tráfego DNS obtido de um repositório de Botnets Conhecidas é enviado para uma fase de Pré-Processamento (Passo 1), onde será separado em respostas NXDOMAIN e respostas Suspeitas. O tráfego NXDOMAINS é enviado para a Modelagem (Passo 2a) e o tráfego Suspeito é enviado para a extração dos endereços IP dos servidores reportados (Passo 2b).

Quando a camada de Geração de Modelos recebe uma nova ameaça (Passo 4 da camada anterior), as requisições NXDOMAIN são enviadas direto para o módulo de Modelagem. Neste Módulo, são extraídas a características dos domínios requisitados e submetidos à técnicas de aprendizado de máquina para a geração dos modelos de detecção que serão enviados ao módulo de Detecção de C&C da camada de Monitoramento (Passo 3 da camada de Geração de Modelos).

© 2017 Sociedade Brasileira de Computação

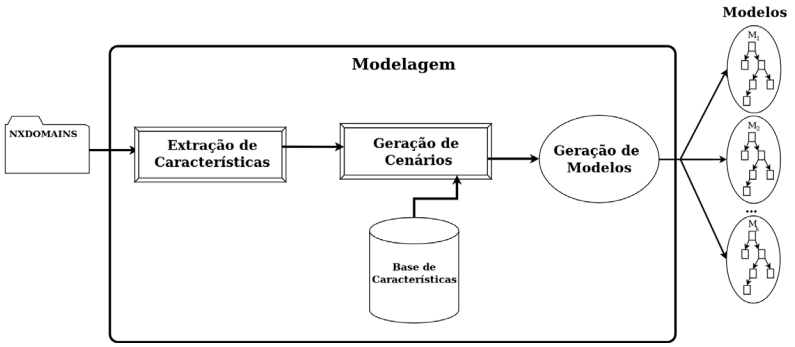


Figura 6. Estrutura interna no módulo Modelagem

A geração dos modelos de decteção leva em consideração o conhecimento observado anteriormente pela arquitetura. Nesta fase, um problema de classificação contendo *M* classes, é transformado em *M* problemas de classificação binária. Isto é alcançado através da Geração de Cenários (Figura 6), executado logo após a extração das características. Cada cenário possui uma série de cruzamentos binários das características de cada amostra repassada (Figura 7). Em seguida, os cenários são enviados para a geração de modelos através do uso de aprendizado de máquina.

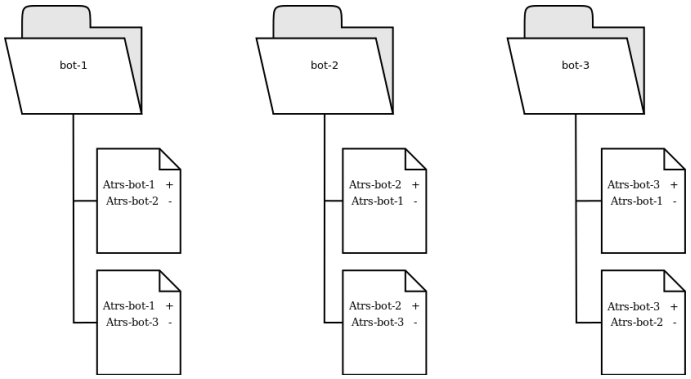


Figura 7. Representação dos cruzamentos binários que compõem cada cenário

5. Implementação e Análise dos Resultados

A implementação da arquitetura está dividida em duas fases, relacionadas as suas duas camadas: Monitoramento e Modelagem. Para este trabalho, a camada de Modelagem encontra-se concluída. Um outro trabalho, em andamento, trará a abordagem de monitoramento.

A base de *botnets* conhecidas foi montada a partir de dados disponibilizados pelo projeto *Malware Capture Facility Project* [Stratosphere 2015]. Este projeto possui um repositório responsável pela obtenção e manutenção de uma base de dados contendo o tráfego malicioso e não-malicioso.

Foram utilizadas 9 amostras de tráfego de rede. Dentre estas estavam 4 famílias de *bots*, onde uma família é composta pelas variantes de um *bot*. Dentre as famílias observadas, 3 utilizavam DGAs e 1 DGA+FFSN para o estabelecimento de comunicação com seus servidores de C&C.

 © 2017 Sociedade Brasileira de Computação

Os domínios gerados por DGAs não necessitam ser algo compreensível por seres humanos, tratam-se, basicamente, de cadeias pseudoaleatórias de caracteres alfanuméricos como: *aeaybikhawkftknjrtxd.org* e *g7ci7ek5ek1di.ad*.

Com base na estrutura dos domínios gerados, os atributos a seguir foram considerados fundamentais para capturar as características básicas dos domínios gerados por um DGA:

- Quantidade de níveis de domínio;
- Tamanho do 2LD;
- Número de caracteres distintos no 2LD;
- Número de dígitos no 2LD;
- Entropia do 2LD;
- Tamanho do 3LD;
- Número de caracteres distintos no 3LD;
- Número de dígitos no 3LD;
- Entropia do 3LD.

A extração das características foi feita através de um script em linguagem Python 2.7 e criação dos cenários através de um script em linguagem Bash.

Os cenários criados foram utilizados para a alimentação da ferramenta Weka [Hall et al. 2009] para a geração dos classificadores. Devido a sua alta versatilidade em problemas de classificação, os algoritmos selecionados para os testes foram: árvores de decisão [Quinlan 1993] e naive-Bayes [John and Langley 1995].

Os experimentos com a ferramenta Weka foram realizados utilizando a técnica de validação cruzada (*cross-validation*) para a geração dos modelos. Nessa técnica, o conjunto de amostras é dividido em k subconjuntos, dos quais k - 1 serão utilizados para treinar o algoritmo e o subconjunto remanescente para a validação. A cada repetição esses subconjuntos sofrem um rodízio no qual aquele que foi utilizado para validação será utilizado para treinamento na próxima iteração. Nos testes realizados nesse trabalho foi utilizado um valor $k = 10$.

O ambiente de execução utilizou um processador Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz, 8 GB de memória RAM (6 GB dedicados à ferramenta WEKA) e sistema operacional ArchLinux (kernel 4.8.13-1).

As métricas utilizadas foram:

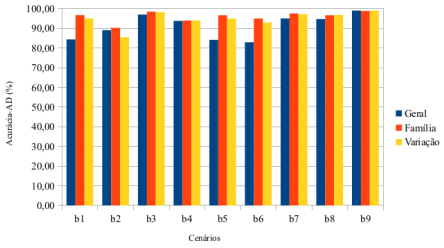
- **Acurácia:** expressa a proximidade entre o valor obtido no experimento e o valor real;
- **Taxa de Positivos Verdadeiros (TPR):** também conhecida como Recall ou taxa de detecção, ela expressa a quantidade de elementos que foram corretamente detectados;
- **Taxa de Positivos Falsos (FPR):** expressa a quantidade de dados negativos que foram classificados como positivos;
- **Área sob a curva ROC (AUC):** expressa a relação entre TPR e FPR. Pode ser utilizada para expressar a qualidade da classificação (valores abaixo de 0,6 a classificação falhou; entre 0,6 e 0,7 classificação pobre; entre 0,7 e 0,8 classificação razoável; 0,8 e 0,9 classificação boa e; acima de 0,9 classificação excelente).

© 2017 Sociedade Brasileira de Computação

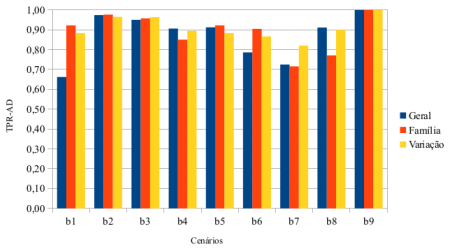
O comportamento dos classificadores foram medidos de acordo com as três visões a seguir:

- **Geral:** expressa o desempenho geral da classificação dentre todas as amostras analisadas, independente de qual família o *bot* pertence;
- **Família:** expressa o desempenho da classificação, levando-se em consideração o comportamento médio obtido com as demais famílias de *bots*;
- **Variação:** expressa o desempenho da classificação, sem considerar os membros da mesma família do cenário.

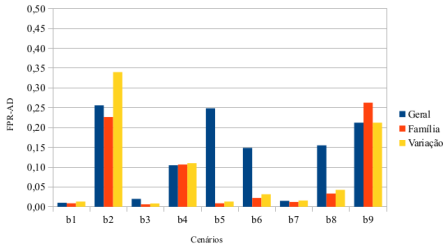
Dentre os algoritmos de aprendizado utilizados, Naive-Bayes e Árvores de Decisão, o algoritmo Naive-Bayes apresentou resultados absolutos abaixo daqueles obtidos com Árvores de Decisão. Entretanto, o comportamento geral da arquitetura para este algoritmo foi similar ao anterior. Sendo assim, serão mostrados apenas os gráficos relacionados ao algoritmo de Árvores de Decisão (Figura 8).



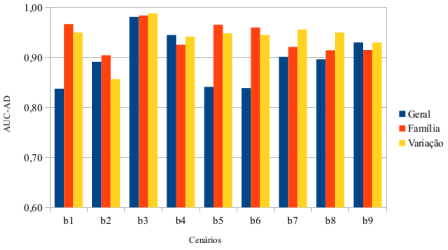
(a) Acurácia.



(b) TPR



(c) FPR



(d) AUC

Figura 8. Resultados para o algoritmo Árvores de Decisão

Como pode ser observado na Figura 8a, as características básicas sugeridas proporcionaram uma acurácia acima de 80% em todos os cenários e independente de considerar a qual família o *bot* pertence. Para a classificação levando em consideração a família dos *bots*, a acurácia passa a níveis acima dos 90% todos os cenários.

Através das Figuras 8b e 8c, observa-se mais detalhadamente os acertos na predição dos cenários. Ainda assim, a predição correta de elementos pertencentes a um dado cenário obteve um valor mínimo de 66%.

De acordo com a Figura 8d, todos os classificadores tiveram um desempenho, no mínimo, bom (entre 0,8 e 0,9) ainda que desconsidere a qual família o *bot* pertence. Quanto à classificação em relação à famílias diferentes, esta é considerada excelente (acima de 0,9) em todos os cenários observados.

© 2017 Sociedade Brasileira de Computação

5.1. O impacto do parentesco entre *bots*

Organizando as famílias de *bots* das amostras temos: as utilizam que DGAs – (b1-b5-b6), (b2-b3-b4) e (b7-b8); e que utiliza DGA e FFSN – (b9).

Durante os experimentos, para algumas famílias, observou-se uma dificuldade em diferenciar amostras de suas variações.

Como visto nas Tabelas 1a e 1c, a arquitetura apresentou uma dificuldade ao diferenciar os domínios gerados pelas variações destas famílias. Entretanto, com a Tabela 1b nota-se que a arquitetura consegue distinguir entre os elementos da família b2-b3-b4.

b1-b5-b6	Acurácia-AD(%)	TPR-AD	FPR-AD	AUC-AD
b1-b5	51,15	0,00	0,00	0,50
b5-b1	51,15	1,00	1,00	0,50
b1-b6	53,32	0,00	0,00	0,50
b6-b1	53,32	1,00	1,00	0,50
b5-b6	52,27	1,00	0,91	0,54
b6-b5	52,26	0,09	0,00	0,54

(a) Família b1-b5-b6

b2-b3-b4	Acurácia-AD(%)	TPR-AD	FPR-AD	AUC-AD
b2-b3	100,00	1,00	0,00	1,00
b3-b2	100,00	1,00	0,00	1,00
b2-b4	99,81	1,00	0,01	0,99
b4-b2	99,81	0,99	0,00	0,99
b3-b4	86,61	0,82	0,11	0,92
b4-b3	86,61	0,89	0,18	0,92

(b) Família b2-b3-b4

b7-b8	Acurácia-AD(%)	TPR-AD	FPR-AD	AUC-AD
b7-b8	80,04	0,06	0,01	0,52
b8-b7	80,04	0,99	0,94	0,52

(c) Família b7-b8

Table 1. Resultados agrupados por famílias de *bots* para o algoritmo Árvores de Decisão

Esse impacto sobre a predição de elementos da mesma família pode estar relacionado ao grau de semelhança entre suas variações sob o conjunto de características utilizado para a classificação. Ou seja, quando a metodologia erra ao classificar um elemento da classe-A como sendo classe-B, onde A e B representam variações de uma família de *bots*, isso significa que o conjunto de carcaterísticas vigente não possui um elemento que expresse a diferença no comportamento observado para estas variações, neste caso, entre os domínios gerados por estes *bots*.

Outra hipótese para o impacto percebido neste trabalho, é que as diferenças entre as variações não estejam relacionadas ao comportamento dos *bots* para o protocolo DNS. Ou seja, essas variações podem utilizar o mesmo DGA em sua comunicação e suas diferenças correspondem a outros comportamentos do *bot*. A utilização do mesmo DGA

© 2017 Sociedade Brasileira de Computação

entre variações de um *bot* justifica-se uma vez que todos os membros da família deverão contactar o centro de comando e controle daquela *botnet*.

Porém, nesta metodologia, como todas as classes referem-se a um bot e nenhuma a um tráfego não-malicioso, o impacto gerado está apenas na predição de qual *botnet* foi detectada e não no processo de detecção da infecção em si. Ou seja, um domínio identificado como bot X, independente de pertencer a classe X ou não, continua sendo um domínio envolvido em atividade maliciosa.

6. Conclusão e Trabalhos Futuros

A arquitetura apresentada é capaz de detectar a presença de *botnet*s que utilizam DGA para o estabelecimento de sua comunicação entre os *bots* e servidores de C&C. Destaca-se que, para a geração dos modelos, foram utilizadas apenas as informações contidas no tráfego que seria descartado na rede: as respostas NXDOMAIN do protocolo DNS.

Foi utilizado um conjunto de características consideradas fundamentais em relação aos nomes de domínio gerados pelos *bots*. Essas características apresentaram um nível de detecção razoável durante os experimentos. Entretanto, esse conjunto de atributos mostrou-se insuficiente na predição entre *bots* de uma mesma família, gerando um aumento no número de falsos positivos. Como a metodologia manipula apenas tráfego malicioso, esses falsos alarmes ainda são identificados como domínios gerados por *bots*.

A arquitetura apresenta os seguintes pontos positivos:

- Pelo fato de basear-se no comportamento DNS apresentado por *bots*, a metodologia aqui apresentada não depende da arquitetura da *botnet* (centralizada, P2P ou híbrida);
- Por não inspecionar a carga (*payload*) dos pacotes contendo os comandos e dados manipulados pela *botnet*, limitando-se apenas ao protocolo DNS, essa metodologia é imune à criptografia, polimorfismo e técnicas semelhantes.

A camada de Monitoramento está em processo de desenvolvimento. Durante este processo, estão sendo identificados os critérios de agrupamento de tráfego citados na Seção 4. Outros comportamentos relacionados à presença de *bots* podem ser incorporados à arquitetura através do tratamento de suas características para que estas possam ser utilizadas para gerar os respectivos cenários e modelos de detecção.

Para diminuir o problema que ocorreu na classificação entre variantes de uma mesma família de *bots*, pode ser utilizado um conjunto de características maior que o utilizado nos experimentos. Porém, como os modelos são gerados a partir de tráfego malicioso apenas, um domínio detectado por um destes modelos continua sendo malicioso.

Podem ser utilizadas também técnicas capazes de detectar erros de digitação originadas de usuários legítimos, a partir de erros de digitação por exemplo. Estes, na maioria dos casos, resultarão em respostas NXDOMAIN e, neste caso, representam ruídos na classificação. Por fim, pode-se utilizar técnicas de classificação em tempo real a fim de minimizar o tempo de detecção e resposta.

Referências

Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., and Feamster, N. (2010). Building a dynamic reputation system for dns. In *Proceedings of the 19th USENIX Confer-*

 © 2017 Sociedade Brasileira de Computação

ence on Security, USENIX Security'10, pages 18–18, Berkeley, CA, USA. USENIX Association.

Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., and Dagon, D. (2012). From throw-away traffic to bots: Detecting the rise of dga-based malware. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 24–24, Berkeley, CA, USA. USENIX Association.

Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., and Kruegel, C. (2012). Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 129–138, New York, NY, USA. ACM.

Bilge, L., Kirda, E., Kruegel, C., and Balduzzi, M. (2011). EXPOSURE: Finding malicious domains using passive dns analysis. In *Proceedings of the Network and Distributed System Security Symposium*, NDSS'11. Internet Society.

Cavallaro, L., Kruegel, C., and Vigna, G. (2009). Mining the network behavior of bots. Technical Report Tech. Rep. 2009-12, Department of Computer Science, University of California, Santa Barbara (UCSB), CA, USA.

Choi, H., Lee, H., Lee, H., and Kim, H. (2007). Botnet detection by monitoring group activities in dns traffic. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, CIT '07, pages 715–720, Washington, DC, USA. IEEE Computer Society.

Cisco (2007). Botnets: The new threat landscape. Technical report, Cisco Systems Inc, USA. White Paper.

de A Ribeiro, V., Filho, R., and Maia, J. (2011). Online traffic classification based on sub-flows. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 415–421, Dublin.

Erquiaga, M. J., Catania, C., and GarcÃa, S. (2016). Detecting dga malware traffic through behavioral models. In *2016 IEEE Biennial Congress of Argentina (ARGENCON)*, pages 1–6.

Gu, G., Perdisci, R., Zhang, J., and Lee, W. (2008). Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 139–154, Berkeley, CA, USA. USENIX Association.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

Holz, T., Gorecki, C., Rieck, K., and Freiling, F. C. (2008). Measuring and detecting fast-flux service networks. In *in NDSS. The Internet Society*.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo. Morgan Kaufmann.

Karim, A., Salleh, R. S., Shiraz, M., Shah, S. A. A., Awan, I., and Anuar, N. B. (2014). Botnet detection techniques: review, future trends, and issues. *Journal of Zhejiang University-SCIENCE C (Computers & Electronics)*, 15(11):943–483.

© 2017 Sociedade Brasileira de Computação

Khandelwal, S. (2017). Wannacry kill-switch(ed) its not over! wannacry 2.0 ransomware arrives.

Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA.

Salusky, W. and Danford, R. (2007). Know your enemy: FastÂflux service networks.

Schiavoni, S., Maggi, F., Cavallaro, L., and Zanero, S. (2014). *Phoenix: DGA-Based Botnet Tracking and Intelligence*, pages 192–211. Springer International Publishing, Cham.

Sikorski, M. and Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, 1st edition.

Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., and Vigna, G. (2009). Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 635–647. ACM.

Stratosphere, T. (2015). Malware capture facility project.

Tiirmaa-Klaar, H., Gassen, J., Gerhards-Padilla, E., and Martini, P. (2013). *Botnets*. Springer Publishing Company, Incorporated.

Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A., and Garant, D. (2013). Botnet detection based on traffic behavior analysis and flow intervals. *Comput. Secur.*, 39:2–16.

 © 2017 Sociedade Brasileira de Computação

Detecção de Intrusão Através de Aprendizagem de Fluxo no Ambiente do Adversário

Eduardo K. Viegas[1], Altair O. Santin[1], Vilmar Abreu[1], Luiz E. S. Oliveira[2]

[1]Pontifícia Universidade Católica do Paraná (PUCPR) – Escola Politécnica – Programa de Pós-Graduação em Informática (PPGIa) – Curitiba, PR – Brasil

[2]Departamento de Informática - Universidade Federal do Paraná (UFPR) – Curitiba, PR Brasil

{eduardo.viegas,santin,vilmar.abreu}@ppgia.pucpr.br,
lesoliveira@inf.ufpr.br

Abstract. *Despite the existence of several works that uses anomaly-based intrusion detection mechanisms, such approaches are hardly used in production environments. In general, the literature does not consider the adversarial environment properties, in which an attack will attempt to evade the detection mechanism. This paper proposes and evaluates an approach to reliably perform intrusion detection through stream learning techniques in the adversarial environment. The proposal relies in class-specific anomaly detectors and a rejection mechanism in order to reliably update the detection mechanism. The proposal evaluation showed that the proposal provides resiliency to both causative and exploratory attacks.*

Resumo. *Apesar da existência de diversos trabalhos que utilizam técnicas de detecção de intrusão baseada em anomalia, dificilmente tais técnicas são utilizadas em produção. Percebe-se que, em geral, a literatura não considera o ambiente do adversário, em que um atacante tenta evadir o mecanismo de detecção. Neste artigo é proposto e avaliado uma abordagem para efetuar a detecção de intrusão em fluxo de dados de forma confiável no ambiente do adversário. A proposta utiliza detectores de anomalia específicos as classes consideradas e um mecanismo de rejeição para permitir a atualização do sistema de forma confiável. A avaliação da proposta mostrou que a abordagem provém resiliência a ataques causais e exploratórios.*

1. Introdução

Sistema de Detecção de Intrusão (*Intrusion Detection System - IDS*) é uma abordagem amplamente referenciada pela literatura para detecção de ataques em nível de rede.. Um IDS permite a detecção de ataques, uso malicioso ou inadequado de um sistema computacional ou uma rede de computadores [García-Teodoro et al. 2009]. As detecções de intrusões são efetuadas, geralmente, através da utilização de técnicas de aprendizagem de máquina [Jyothsna, V. et al. 2011]. Nessa abordagem, o IDS é treinado como um conjunto limitado de perfis/comportamentos. Posteriormente, durante seu uso em produção, qualquer evento (e.g. um pacote da rede) que desvia significativamente dos perfis previamente conhecidos é classificado como uma tentativa de intrusão [Denning, D. E. 2012]. Essa abordagem é chamada detecção de intrusão baseada em anomalia, onde é possível detectar novos ataques.

 © 2017 Sociedade Brasileira de Computação

Apesar dos resultados promissores reportados na literatura, raramente a detecção de intrusão baseada em anomalia é utilizada em produção [Sommer, R. e Paxson, V. 2012]. Neste sentido, a abordagem baseada em assinatura que realiza a classificação através do confronto do evento com padrões conhecidos de ataques ainda é a mais utilizada [Axelsson, S. 2000]. Essa discrepância entre as abordagens da literatura e a utilização em produção ocorre principalmente devido ao alto número de falsos alertas emitidos [Sommer, R. e Paxson, V. 2012] e a necessidade de atualização do algoritmo de detecção de intrusão ao longo do tempo [Maggi, F. et al. 2009]. O algoritmo deve ser atualizado devido a mudanças de comportamentos no ambiente, tal como a ocorrência de novos tráfegos de rede e novos ataques [Axelsson, S. 2000].

Dessa maneira, nos últimos anos as técnicas de aprendizagem de fluxo (*stream learning*) [He, H. et al. 2011] estão ganhando destaque. Essas técnicas são geralmente utilizadas em cenários onde o conjunto de conceitos (classes) mudam ao longo do tempo [He, H. et al. 2011]. Por exemplo, no contexto de detecção de intrusão em rede, o comportamento legitimo (normal) muda ao longo do tempo com a implantação de novos serviços, enquanto o comportamento do atacante (intruso) também muda devido a geração de novos ataques [Sommer, R. e Paxson, V. 2012]. Nesse cenário em constante mudanças, a atualização do mecanismo de detecção é frequentemente efetuada em janelas de tempo deslizantes (*sliding windows*). Assim, os eventos mais recentes são identificados com maiores importâncias durante a etapa de classificação, enquanto eventos antigos são descartados [Bifet, A. et al. 2007].

Essa capacidade de lidar com ambientes que mudam ao longo do tempo, permite que diversos trabalhos argumentem que técnicas de *stream learning* possibilitam o uso de abordagens de detecção de intrusão baseada em anomalia em ambientes de produção [Jyothsna, V. et al. 2011]. Porém, na literatura, ainda existe carência na validação de tais abordagens [Sommer, R. e Paxson, V. 2012]. Em geral, os trabalhos na literatura não consideram o ambiente do adversário, onde um atacante pode tentar evadir o mecanismo de detecção, seja pela exploração das propriedades do mecanismo de detecção ou através da injeção de ataques durante a etapa de treinamento do sistema [Tygar, J. D. 2009].

Neste artigo é apresentado uma abordagem para a utilização de técnicas de detecção de anomalia para fluxos de dados que permite a detecção de intrusão de rede de maneira confiável. A proposta utiliza detectores de anomalias específicos a cada classe considerada para automaticamente e confiavelmente atualizar o mecanismo de detecção de intrusão ao longo do tempo. A proposta considera as características do ambiente do adversário através da rejeição de potenciais tentativas de evasão ou classificações não confiáveis.

2. Fundamentação

Esta seção apresenta duas abordagens consideradas para a detecção de intrusão baseada em anomalias: aprendizagem de máquina tradicional e *stream learning*.

2.1. Aprendizagem de Máquina Tradicional

A aprendizagem de máquina tradicional utiliza os algoritmos de reconhecimento de padrões tradicionais [Corona, I. et al. 2013], em que seu processo geralmente se baseia em um conjunto de dados fixo para treinamento. Nesse caso, o comportamento do usuário normal e do atacante são obtidos e armazenados em um conjunto fixo de dados. O

 © 2017 Sociedade Brasileira de Computação

conjunto de dados, geralmente referido como conjunto de dados de treinamento, é usado durante a etapa de treinamento do classificador (algoritmo de reconhecimento de padrões). Durante esse processo, uma taxa de falso positivo (FP) e falso negativo (FN) é estimada através de outro conjunto de dados denominado conjunto de dados de teste. A taxa de FP refere-se à taxa em que os eventos normais são incorretamente classificados como uma tentativa de intrusão, enquanto a taxa de FN refere-se à taxa em que eventos de intrusão são incorretamente classificados como uma atividade normal.

No processo de aprendizagem da máquina tradicional, quando o comportamento do atacante muda, o algoritmo de reconhecimento de padrões deve ser treinado novamente. Esse processo de treinamento normalmente é uma tarefa dispendiosa, uma vez que o ambiente deve ser monitorado para que os novos comportamentos sejam identificados (geralmente de forma manual). Assim, o classificador deve ser treinado e as taxas de FP e FN devem ser novamente estimadas. Além disso, identificar mudanças de comportamentos no ambiente é uma tarefa desafiadora, uma vez que a identificação é frequentemente baseada no aumento das taxas de FP e FN, que também são identificadas manualmente através da assistência de especialistas [Sommer, R. e Paxson, V. 2012].

A facilidade de atualização torna-se relevante no campo de detecção de intrusão baseado em anomalias. Novos ataques são descobertos diariamente, juntamente com novos serviços e seus conteúdos. Assim, independentemente do mecanismo de detecção de intrusão usado, o sistema deve ser frequentemente atualizado [Sommer, R. e Paxson, V. 2012].

2.2. Aprendizagem de Máquina para Fluxos de dados (*Stream Learning*)

As técnicas tradicionais de aprendizagem de fluxo (*stream learning*) visam lidar automaticamente com as mudanças do ambiente ao longo do tempo [He, H. et al. 2011]. Para isso, o algoritmo de aprendizagem de fluxo é executado em um ciclo repetido, no qual o algoritmo é atualizado de acordo com os eventos recebidos do fluxo de dados da rede [Bifet, A. et al. 2007]. Esse processo ocorre de acordo com os limites de memória e processamento. Nesse caso, os limites de memória são definidos de acordo com o número de eventos considerados durante o processo de classificação. Para esse propósito, a maioria das estratégias depende de uma abordagem de janela deslizante (*sliding window*), na qual uma janela (intervalo pré-definido) de eventos recentes é mantida e os eventos mais antigos são descartados de acordo com um conjunto de regras [Bifet, A. et al. 2007].

Devido à sua natureza adaptativa, um algoritmo de classificação de fluxo de dados deve apresentar as seguintes propriedades [Bifet, A. et al. 2007]: (*i*) processar e inspecionar um evento por vez; (*ii*) usar quantidade limitada de memória; (*iii*) classificar os eventos em uma quantidade restrita de tempo e (i*v*) prever novos eventos a qualquer momento.

2.3. Aprendizagem de Máquina no Ambiente do Adversário

Ao longo dos últimos anos, as técnicas tradicionais de aprendizagem de máquina e de fluxo foram aplicadas com sucesso em vários campos [Sommer, R. e Paxson, V. 2012]. Por exemplo, foram aplicadas em reconhecimento de imagem, sistemas de recomendação de produtos, tradução de linguagem natural [Oliveira, L. S. et al. 2002] entre outros. No entanto, em detecção de intrusão ainda existe falta de aplicabilidade em ambientes de produção [Sommer, R. e Paxson, V. 2012].

© 2017 Sociedade Brasileira de Computação

Um campo de pesquisa emergente, conhecido como aprendizagem de máquina no ambiente do adversário [Tygar, J. D. 2009] considera o uso de técnicas de aprendizagem de máquina nas configurações de um adversário - chamado configurações adversárias. Nesses casos, o adversário (atacante) tentará evadir o mecanismo de detecção de intrusão usando técnicas sofisticadas de ataques, denominadas de ataques causais e exploratórios [Tygar, J. D. 2009]. Os ataques causais referem-se a ataques que ocorrem durante o processo de treinamento [Tygar, J. D. 2009]. Neste caso, o atacante tenta injetar tentativas de intrusões, estas incorretamente classificadas, no conjunto de dados de treinamento como eventos normais. Por outro lado, os ataques exploratórios visam explorar as propriedades do algoritmo de aprendizagem de máquina [Tygar, J. D. 2009], através da manipulação de uma tentativa de intrusão, de forma que o mecanismo de detecção classifique o evento como uma atividade normal.

Um exemplo de ataque causal em um *stream learning* tradicional é exibido na Figura 1. Nesse cenário, o classificador classifica anomalias (*outliers*) ao longo do tempo. No Tempo 1 (Figura 1, Tempo 1), a janela deslizante é totalmente preenchida com eventos normais (*inliers*), desta maneira os ataques são classificados como *outliers*. No entanto, independentemente da classe atribuída, o ataque, inicialmente classificado como *outlier*, é adicionado na janela deslizante, enquanto um *inlier* (evento mais antigo na janela deslizante) é removido. O mesmo procedimento ocorre no Tempo 2 (Figura 1, Tempo 2). No entanto, o novo ataque ainda é classificado como *outlier*, pois não há eventos suficientes em sua vizinhança. Finalmente, no Tempo 3 (Figura 1, Tempo 3), o atacante é capaz de injetar ataques suficientes na janela deslizante, pois o número de vizinhos está próximo o suficiente para formar um grupo e ser classificado como *inlier*.

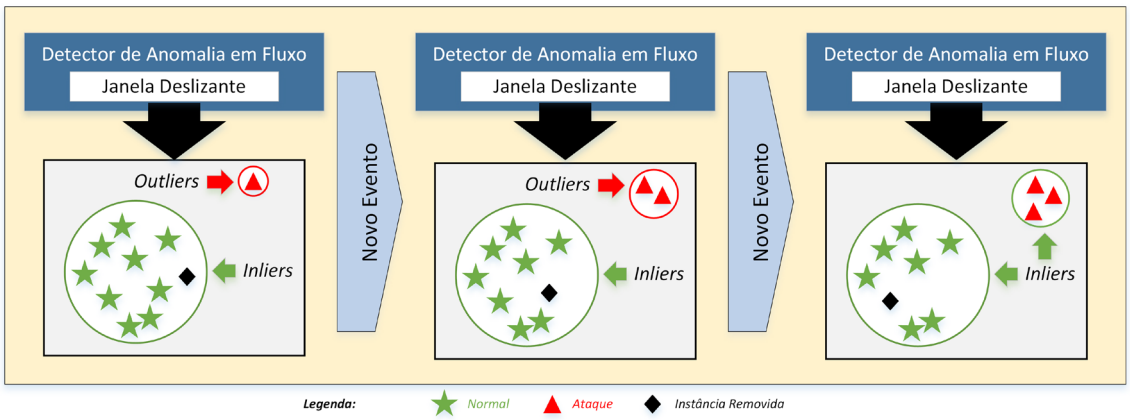


Figura 1. Comportamento da Janela Deslizante em detectores de anomalias em fluxo tradicionais.

Dessa maneira, o uso de técnicas de aprendizagem de máquina, especialmente técnicas de *stream learning*, na configuração do adversário visam a concepção de mecanismos de detecção que sejam capazes de resistir ou apresentar resiliência aos ataques (sofisticados) mencionados anteriormente.

3. Proposta

Esta proposta provê uma abordagem confiável de *stream learning* para detecção de intrusão baseada em anomalia, que permite a atualização automática do mecanismo de detecção ao longo do tempo. Para isso, a proposta utiliza detectores de anomalias

© 2017 Sociedade Brasileira de Computação

específicos a classe, provendo resiliência para ataques causais e exploratórios (seção 2.3). O método proposto é exibido na Figura 2 e descrito nas próximas subseções.

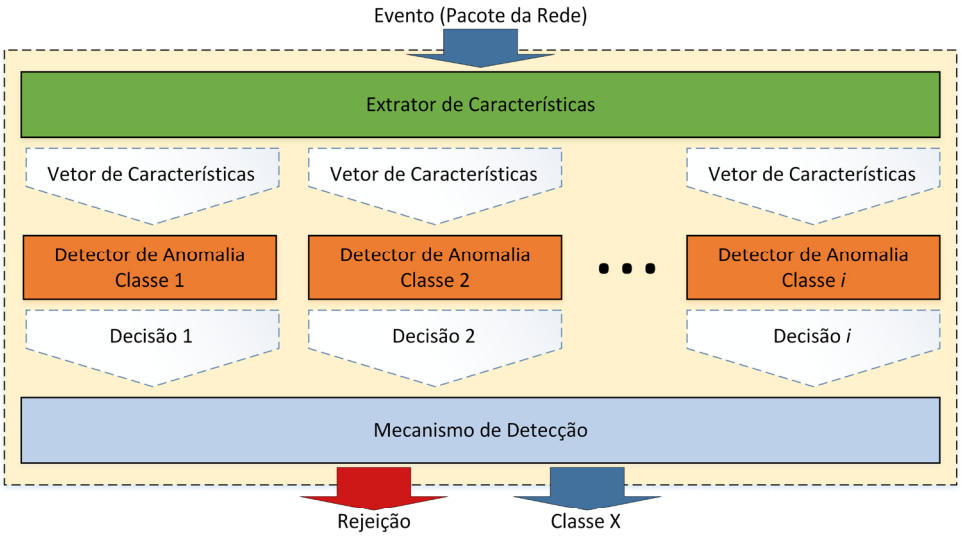


Figura 2. Método proposto para detecção de intrusão baseada em anomalia através de algoritmos de aprendizagem de fluxo no ambiente do adversário.

3.1. Esquema de Detecção

O método proposto considera que o esquema de detecção utiliza detectores de anomalias em fluxo específicos a classe. Por exemplo, um detector de anomalia para eventos normais e outro para ataques. A detecção é efetuada de acordo com a Figura 2. O processo é composto de 5 etapas: (*i*) o conjunto de características é extraído do evento a ser classificado, e.g. um pacote da rede; (*ii*) o vetor de características é fornecido a cada detector de anomalia; (*iii*) cada detector realiza a detecção atribuindo a classe ao evento como *inlier* (evento pertence ao grupo da classe do detector de anomalia) ou *outlier* (evento não pertence ao grupo do detector de anomalia); (*iv*) o mecanismo de detecção recebe a decisão de cada detector de anomalia e tenta encontrar um consenso entre as decisões; (*v*) se a decisão é unanime, a classe determinada é atribuída ao evento, caso contrário, a decisão sobre o evento é rejeitada.

Quando o mecanismo de detecção recebe uma decisão de evento, decide se a classificação do evento é confiável ou não. A confiabilidade da classificação de um evento (representado como "Classe X" na Figura 2) é provida da nulidade da interseção da decisão de todos os classificadores. O cálculo de confiabilidade é exibido na Fórmula 1, sendo que $Decisão_i$ determina cada saída dos detectores de anomalias.

$$\cap_{i=1}^{n}(Decisão_i) = \emptyset \qquad \text{(Fórmula 1)}$$

Como exemplo, considere dois algoritmos de detecção de anomalias, um para eventos normais e outro para ataques. Um evento que é classificado como *inlier* para normal e *outlier* para ataque é confiável, uma vez que a decisão é uma unanimidade. Ou seja, não há interseção entre classes de classificação nos diferentes mecanismos de detecção. No entanto, um evento que é classificado como *inlier* para mais de um detector de anomalia deve ser rejeitado, pois a decisão não é confiável. As classificações rejeitadas indicam que uma possível tentativa de evasão ou um falso alarme podem estar ocorrendo. Dessa maneira, outro mecanismo de detecção deve ser usado, por exemplo, um mecanismo de detecção de intrusão baseado em assinatura ou inspeção manual.

© 2017 Sociedade Brasileira de Computação

3.2. Provendo Resiliência a Adversários – Exploratório

Ao contrário dos algoritmos tradicionais de aprendizagem de fluxo, a abordagem proposta fornece resiliência a ataques exploratórios através do comportamento imutável de cada algoritmo de detecção de anomalias. O comportamento imutável é definido por uma restrição que não permite que um *outlier* se torne um *inlier* no algoritmo de detecção de anomalias ao longo do tempo. A abordagem proposta considera que para detecção de intrusão baseada em anomalias, um evento inicialmente classificado como *outlier* não se tornará *inlier* em nenhum momento do tempo. Por exemplo, um ataque que foi classificado como *outlier* (ataque) pelo algoritmo de detecção de anomalia da classe normal não deve ser classificado como um evento normal, mesmo que a sua ocorrência aumente na janela deslizante ao longo do tempo.

Ao usar o comportamento imutável, o atacante não poderá explorar o intervalo da janela deslizante para perverter (poluir) a classificação de eventos que estão sendo analisados por um detector de anomalia. É importante notar que os eventos classificados como *inlier* continuam a serem adicionados à janela deslizante de aprendizagem de fluxos. Portanto, o algoritmo ainda é capaz de se adaptar às mudanças no fluxo. Porém, a proposta mitiga um possível ataque de evasão, em que a quantidade de eventos *outliers* se tornam predominantes em uma janela deslizante, desta maneira, tornando um evento *outlier* em *inlier*.

3.3. Provendo Resiliência a Adversários – Causais

Para proporcionar resiliência a ataques causais, a proposta baseia-se tanto no comportamento imutável (seção 3.2) quanto nos detectores de anomalias específicos da classe. A proposta considera que a resiliência a ataques causais deve ser fornecida em duas etapas: treinamento inicial e readaptação contínua (atualização).

O treinamento inicial está relacionado a população inicial da janela deslizante do detector de anomalias. Nesta fase, as janelas deslizantes dos detectores de anomalias ainda estão sendo preenchidas, ou seja, estão suscetíveis a ataques causais. Desta maneira, a abordagem proposta pressupõe que há, pelo menos, uma população inicial que permita a classificação correta em pelo menos um detector de anomalias, uma vez que a janela deslizante será atualizada de acordo com os eventos iniciais. Preencher as janelas deslizantes iniciais com um número predominante de cópias do mesmo evento *inlier* é uma maneira de garantir a confiabilidade. Dessa maneira, o detector de anomalias é confiável, sendo que as saídas das classificações podem ser confiáveis se a unanimidade de classificação for alcançada, caso contrário a classificação é rejeitada.

A fim de fornecer readaptação contínua, a proposta depende do mecanismo de detecção de anomalia específico a classe e do comportamento imutável (Seção 3.2). O mecanismo de detecção de anomalia específico a classe fornece resiliência à manipulação do comportamento do evento. Por exemplo, o atacante deve manipular o comportamento do evento de modo em que o evento se comporte como um evento normal, enquanto também é um evento anômalo para o mecanismo de detecção de anomalias de ataque. Desta maneira, o comportamento imutável dificulta o ataque sobre a janela deslizante, uma vez que o atacante deve ter habilidades para manipular os eventos de maneira que perverta todos os detectores de anomalias.

 © 2017 Sociedade Brasileira de Computação

4. Avaliação

4.1. Cenário de testes

Devido às limitações conhecidas das abordagens atuais em relação as bases de dados para avaliação de IDSs [Sommer, R. e Paxson, V. 2012], uma abordagem utilizada em trabalho anterior foi adotada [Viegas et al. 2017]. Desta forma, duas classes de tráfego de rede foram utilizadas para compor o conjunto de dados: normal e ataque. O ambiente de testes é composto por 100 máquinas clientes, 3 hosts atacantes e um único servidor.

Para gerar o tráfego normal, os serviços fornecidos no cenário de teste foram: HTTP, SSH, SMTP, SNMP e DNS. Foram gerados três conjuntos de ataques: SYNFlood, UDPFlood e ICMPFlood. O cenário de testes foi executado durante 30 minutos. A quantidade total de tráfego de rede gerada para cada serviço e ataque é relatada na Tabela 1. Para cada pacote de rede, são extraídos 23 atributos [Viegas et al. 2017].

Para fins de teste, o algoritmo de detecção de *outliers* em fluxo Micro-cluster (MCOD) [Kontaki, M. et al. 2011] foi considerado no método proposto (Figura 2, Seção 3). Para fins de comparação, foram consideradas duas outras abordagens: a Aprendizagem de Máquina Tradicional (Seção 2.1) e Aprendizagem de Fluxo Tradicional (Seção 2.2).

Tabela 1. Distribuição do tráfego no ambiente de testes

Classe	Tráfego Gerado	Quantidade de Pacotes
Normal	HTTP	20.238.802
	SMTP	2.298.222
	SSH	1.048.482
	SNMP	3.017.731
	DNS	135.188
Ataque	SYNFlood	471.288
	UDPFlood	121.645
	ICMPFlood	130.698

4.2. Obtenção do Modelo

Para o método proposto (Seção 3), duas classes foram consideradas: normal e ataque. Para cada teste, foram utilizados dois detectores de anomalias: um para o normal e outro para o ataque. Uma janela deslizante de 10.000 eventos foi considerada. Um total de 50 eventos foram estabelecidos como a quantidade de vizinhos próximos (k). Cada detector de anomalias específico a classe tem seu próprio parâmetro de raio. Uma série de testes foram efetuados para estabelecer os parâmetros do raio, sendo que o critério escolhido era minimizar o valor de objetivo da Fórmula 2.

$$objetivo = erro_{taxa} + rejeição_{taxa} \quad \text{(Fórmula 2)}$$

O $erro_{taxa}$ e $rejeição_{taxa}$ foram definidos através da detecção dos 10.000 eventos normais iniciais seguidos pela detecção de 10.000 eventos de ataque do conjunto de dados de treinamento (Tabela 1). Os valores de raio para cada detector de anomalia de classe, normal e ataque, foram variados em um intervalo de 0,01 entre 0,00 a 2,00.

O classificador k-Vizinhos Próximos (*k-Nearest* Neighbors - kNN) foi utilizado para a aprendizagem de máquina tradicional (Seção 2.1). Para permitir a comparação, um total de 5000 eventos para cada classe (normal e ataque) foram usados. Os 5000 eventos de cada classe são definidos pelo algoritmo de agrupamento *k-means* [Alsabti, K. et al. 1997] usando o conjunto de dados de treinamento (25% dos eventos escolhidos

© 2017 Sociedade Brasileira de Computação

aleatoriamente da Tabela 1). Os vizinhos do kNN configurados não são atualizados durante o processo de classificação. Finalmente, para a aprendizagem de fluxo tradicional (Seção 2.2), o MCOD é usado. No entanto, apenas a classe normal é considerada, conforme comumente é realizado em trabalhos relacionados [Kontaki, M. et al. 2011]. Finalmente, o processo de obtenção do raio foi estabelecido apenas pela minimização do erro.

4.3. Avaliação tradicional

Inicialmente, o processo de avaliação tradicional foi considerado para as abordagens avaliadas. Para a avaliação tradicional, as configurações do adversário (Seção 2.3) não são consideradas.

Para o classificador kNN, o conjunto de dados foi dividido em: treinamento, validação e teste, sendo que o conjunto de dados (Tabela 1) foi distribuído como 25% para treinamento, 25% para validação e 50% para teste. Por efetuar a sua aprendizagem em fluxo, o MCOD foi utilizado todo o conjunto de dados durante a avaliação tradicional. Neste caso, os eventos são reproduzidos exatamente na mesma ordem em que aparecem no conjunto de dados original (Tabela 1). A Tabela 2 exibe as taxas de acurácia em relação a cada uma das abordagens avaliadas. Sendo que a coluna do método se refere à abordagem utilizada durante a fase de detecção. Cada abordagem é testada com um conjunto diferente de ataques utilizados durante a etapa de treinamento, exibidos entre parênteses na coluna do método.

Tabela 2. Avaliação da abordagem proposta e das abordagens tradicionais.

Método	Acurácia (Rejeição)			
	Acurácia Normal (Rejeição)	Acurácia SYNFlood (Rejeição)	Acurácia UDPFlood (Rejeição)	Acurácia ICMPFlood (Rejeição)
Abordagem Proposta MCOD (SYNFlood)	100,00 (0.04)	100,00 (0,00)	- (100,00)	- (100,00)
Aprendizagem de Máquina Tradicional kNN (SYNFlood)	99,83 (N.A.)	100,00 (N.A.)	100,00 (N.A.)	0,01 (N.A.)
Abordagem Proposta MCOD (UDPFlood)	100,00 (0.98)	- (100,00)	100,00 (0,10)	- (100,00)
Aprendizagem de Máquina Tradicional kNN (UDPFlood)	99,93 (N.A.)	49,97 (N.A.)	100,00 (N.A.)	0,01 (N.A.)
Abordagem Proposta MCOD (ICMPFlood)	100,00 (0.97)	- (100,00)	- (100,00)	100,00 (0,12)
Aprendizagem de Máquina Tradicional kNN (ICMPFlood)	100,00 (N.A.)	3,23 (N.A.)	100,00 (N.A.)	100,00 (N.A.)
Aprendizagem de Fluxo Tradicional MCOD	99,19 (N.A.)	0,81 (N.A.)	0,69 (N.A.)	0,22 (N.A.)

É possível notar que tanto a abordagem proposta, como a aprendizagem de máquina tradicional (kNN) conseguem detectar o mesmo conjunto de ataques com uma taxa de precisão significativamente alta. Em relação à detecção de ataques, tanto a abordagem proposta como o kNN apresentaram uma taxa de FN de zero por cento, ao detectarem o mesmo conjunto de ataques ao qual o sistema foi treinado. Considerando a taxa de FP, o classificador kNN alcançou 0,17, 0,07 e zero por cento quando treinados com ataques SYNFlood, UDPFlood e ICMPFlood, respectivamente. A abordagem proposta alcançou uma taxa de FP de zero por cento em todos os casos testados. No entanto, a abordagem proposta rejeitou classificações potencialmente erradas. Nesse caso,

 © 2017 Sociedade Brasileira de Computação

0,04, 0,98 e 0,97 por cento dos eventos normais foram rejeitados pelos ataques SYNFlood, UDPFlood e ICMPFlood, respectivamente. Pode-se observar que a abordagem proposta apresenta uma precisão de detecção semelhante quando comparada à abordagem tradicional de aprendizagem por máquina. No entanto, a abordagem proposta rejeita classificações potencialmente erradas, que podem ser observadas comparando a taxa de FP do kNN e a taxa de rejeição da abordagem proposta.

Considerando a abordagem tradicional de aprendizagem de fluxo, foi possível notar que quando os eventos são reproduzidos exatamente na mesma ordem que são apresentados no conjunto de dados original (Tabela 1), o método é capaz de detectar apenas os ataques iniciais - quando a janela deslizante é quase totalmente preenchida com eventos normais. No entanto, à medida que a ocorrência de ataques aumenta, os eventos de ataque adicionais são classificados como *inliers* (normal). Tal propriedade ocorre devido à natureza adaptativa dos algoritmos de aprendizagem de fluxo, que permitem que um evento que foi inicialmente classificado como *outlier* (ataque), seja adicionado na janela deslizante. Permitindo assim que um ataque se torne um *inlier* ao longo do tempo, pervertendo o comportamento dos detectores de anomalias. Desta forma, os algoritmos tradicionais de aprendizagem de fluxo devem considerar essa propriedade no campo de detecção de intrusão, que é tratado neste trabalho, através do comportamento imutável (Seção 3.2).

4.4. Ambiente do adversário – Ataques exploratórios

Dois tipos de ataques exploratórios foram avaliados: a evasão tradicional e a exploração da janela deslizante.

4.4.1. Evasão tradicional

A evasão tradicional refere-se à detecção de ataques que possuem tipo diferente de comportamento para o ataque no qual o sistema foi treinado, no entanto produzem o mesmo resultado ao alvo. Por exemplo, ataques que objetivam gerar uma quantidade significativa de tráfego de rede na vítima, independentemente do protocolo considerado, e.g. inundação UDP, TCP ou ICMP. Desta forma, cada uma das abordagens consideradas também foi avaliada com ataques de inundação diferentes do qual o sistema foi treinado. A precisão obtida é exibida na Tabela 2.

Em relação a aprendizagem de máquina tradicional (kNN), o atacante é capaz de evadir o sistema ao mesmo tempo em que gera um ataque que produz o mesmo resultado no alvo. Quando o kNN foi utilizado, a possibilidade de evasão foi evidenciada para todos os ataques avaliados: SYNFlood, UDPFlood e ICMPFlood. Por exemplo, quando o sistema foi treinado com ataques SYNFlood, o invasor ainda é capaz de evadir o mecanismo de detecção através da geração de ataques ICMPFlood. Por outro lado, a abordagem proposta aceitou apenas os resultados das classificações em relação aos ataques que sistema foi treinado. A elevada taxa de rejeição e o aumento da confiabilidade ocorreram devido ao possível aumento da taxa de erro, no qual ocorreu devido à falta de decisão unanime entre os detectores de anomalia que realiza a rejeição das classes atribuídas aos eventos.

© 2017 Sociedade Brasileira de Computação

4.4.1. Exploração da Janela Deslizante

O segundo ataque exploratório avaliado é chamado de exploração da janela deslizante. Esse ataque visa avaliar a precisão do algoritmo de aprendizagem de fluxo tradicional de acordo com a ocorrência de ataques em sua janela deslizante. Conforme observado na Seção 4.2, o aumento na frequência de ataques na janela deslizante torna o algoritmo de aprendizagem de fluxo não confiável. A Figura 3 (gráfico inferior) exibe a taxa de erro em relação a cada um dos ataques avaliados, durante os 8 a 9 milhões de pacotes no conjunto de dados criado. A taxa de erro é avaliada em um intervalo de 1.000 pacotes.

É possível observar que durante a detecção de eventos normais, a taxa de erro do algoritmo de aprendizagem de fluxo tradicional continua a ser semelhante à taxa obtida durante a avaliação tradicional (seção 4.2, 0,81 por cento). No entanto, à medida que os ataques começam a ocorrer (em torno do pacote 8,2 milhões), a taxa de erro para a detecção dos ataques aumenta devido ao aumento da ocorrência dos ataques. Desta forma, o atacante é capaz de explorar a janela deslizante do algoritmo de aprendizagem de fluxo tradicional. Isso é possível aumentando a ocorrência do ataque (Figura 3, gráfico superior), fazendo com que os ataques sejam classificados como *inliers* (normais) devido ao aumento de frequência na janela deslizante.

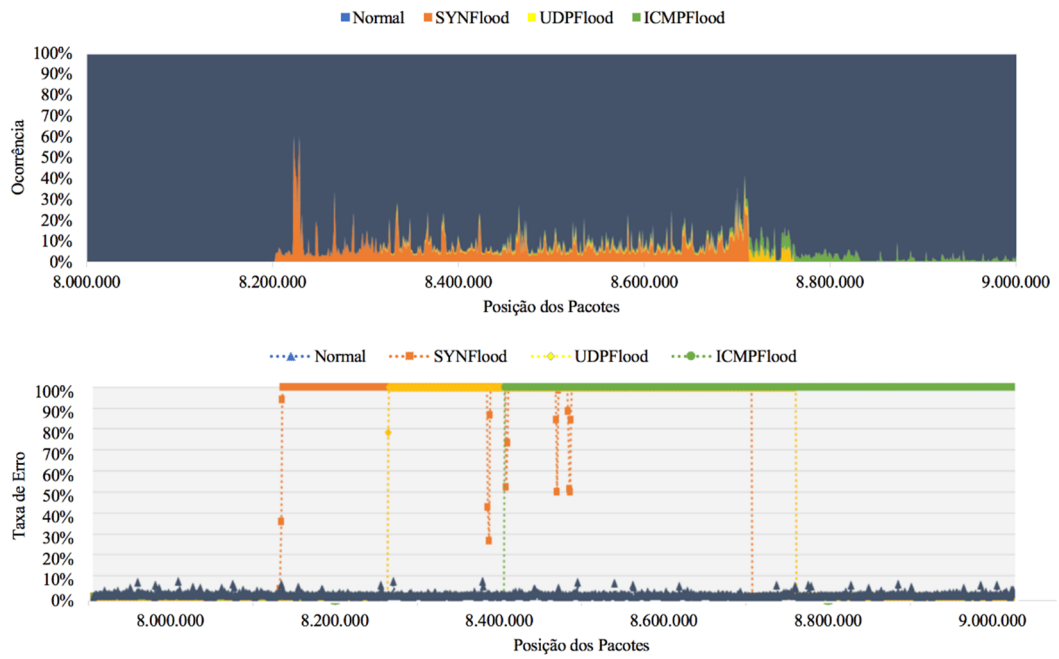


Figura 3. Comportamento da aprendizagem de fluxo tradicional durante a ocorrência de ataques de inundação; o gráfico superior exibe a ocorrência dos ataques; o gráfico inferior exibe a taxa de erro durante a ocorrência dos ataques.

A exploração da janela deslizante não ocorre na abordagem proposta devido ao comportamento imutável (Seção 3.2) e ao mecanismo de detecção de anomalia específico a classe (Seção 3.1). Os resultados são mostrados na Tabela 2. O invasor não é capaz de adicionar ataques na janela deslizante do detector de anomalia da classe normal devido ao comportamento imutável. Embora, se o mecanismo de detecção classificar erroneamente um evento e, assim, adicioná-lo em sua janela deslizante, o evento será rejeitado. Isso ocorre porque não será possível estabelecer uma unanimidade entre os outros detectores de anomalias.

© 2017 Sociedade Brasileira de Computação

4.4. Ambiente do adversário – Ataques Causais

Finalmente, para avaliar a resistência a ataques causais, foi adotada uma abordagem de manipulação da base de dados de treinamento. A aprendizagem de máquina tradicional e a abordagem proposta foram avaliados quanto à influência que os ataques que foram inicialmente injetados no conjunto de dados de treinamento como eventos normais, têm sobre a precisão final dos algoritmos. Assim, o objetivo foi avaliar cada um dos métodos considerados em relação à sua resiliência aos ataques de manipulação da base de dados de treinamento. A Figura 4 exibe a relação entre a taxa de detecção de ataques e o percentual de controle do atacante sobre o conjunto de dados de treinamento, enquanto injeta com sucesso ataques erroneamente classificados como eventos normais.

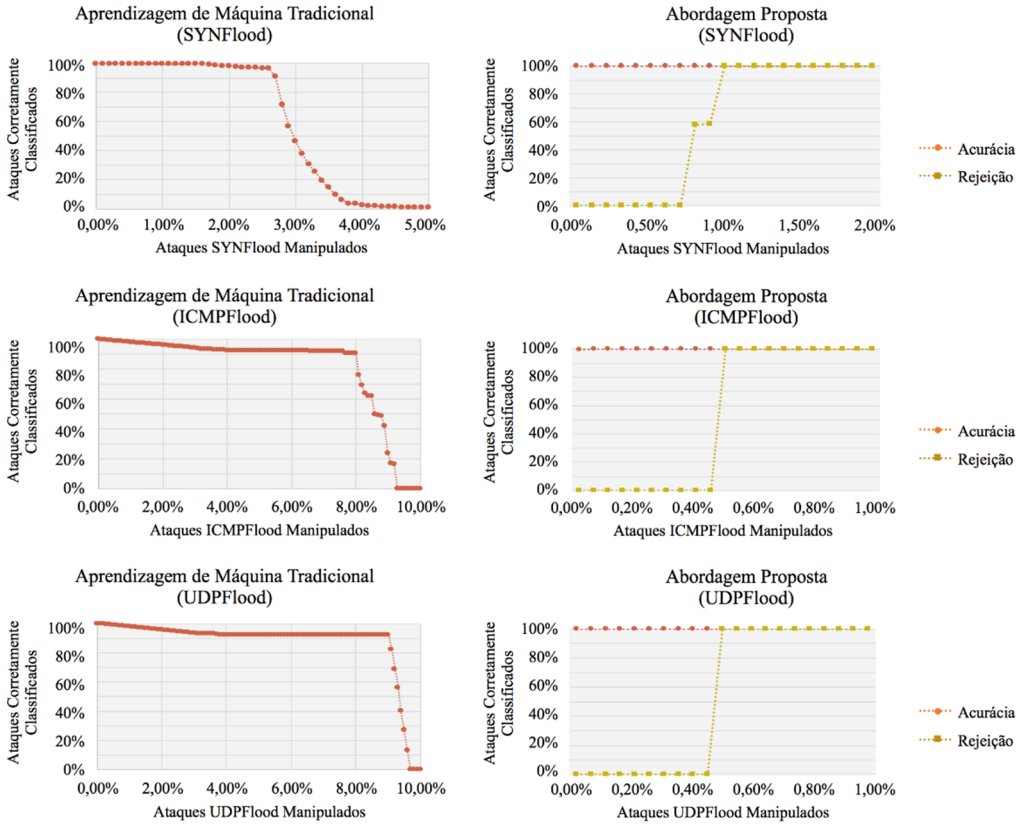


Figura 4. Avaliação da resiliência a ataques causais da abordagem de aprendizagem de máquina tradicional e da abordagem proposta.

Em relação a aprendizagem de máquina tradicional, é possível observar que para os três ataques avaliados, a evasão do mecanismo de detecção é possível. A taxa de precisão dos ataques de SYNFlood caiu para 50% quando apenas 3% dos eventos de ataques foram injetados como eventos normais na base de treinamento. Enquanto, para ICMPFlood e UDPFlood, o atacante poderia evadir o mecanismo de detecção quando apenas 9% dos ataques fossem manipulados. Por outro lado, a abordagem proposta poderia detectar quando os ataques foram manipulados no conjunto de dados de treinamento e rejeitar as classificações adicionais. Tal característica ocorre devido a utilização de detectores de anomalias específicos a classe. Assim, os ataques injetados no conjunto de dados de treinamento como eventos normais resultam na falta de unanimidade durante o processo de classificação, implicando que os eventos sejam rejeitados.

© 2017 Sociedade Brasileira de Computação

5. Trabalhos Relacionados

A baixa utilização de métodos de detecção de intrusão baseados em anomalias em ambientes de produção foi observada nos últimos anos por vários trabalhos [Sommer, R. e Paxson, V. 2012]. Essa lacuna pode ser causada por vários aspectos. No entanto, existe um consenso que o método de detecção deve ser pelo menos confiável e de fácil atualização [Sommer, R. e Paxson, V. 2012].

A confiabilidade da detecção é muitas vezes considerada em outras áreas [Oliveira, L. S. et al. 2002]. Para esse fim, em geral, os autores [Cavalcanti, G. D. C. et al. 2016] utilizaram a probabilidade da classe de saída para rejeitar ou não as decisões. Outras abordagens utilizam um conjunto de classificadores para estabelecer a confiabilidade de classificação através de uma abordagem de votação por maioria [Nelson, B. et al. 2008]. Apesar de ser muitas vezes considerado em outras áreas, até o melhor de nosso conhecimento, a confiabilidade da classificação ainda não foi considerada no campo de aprendizagem de fluxo. Alguns autores, no entanto, consideraram o ambiente do adversário na detecção de intrusão baseada em anomalia.

Tygar e seus colegas [Tygar, J. D. 2009] definiram uma taxonomia utilizada em seu trabalho para classificar possíveis ataques adversários contra o sistema de aprendizagem de máquina. Os autores também avaliaram o impacto que um conjunto de dados de treinamento manipulado incorrem na precisão final do classificador. Em todos os casos avaliados, o classificador tornou-se pouco confiável quando o conjunto de dados de treinamento teve ataques incorretos injetados.

No cenário de detecção de spam, Blaine Nelson e seus colegas [Nelson, B. et al. 2008] avaliaram o impacto da manipulação do conjunto de dados de treinamento na acurácia do sistema. Os autores relataram um aumento na taxa de erro da classificação de até 36% quando o atacante tem controle de apenas 1% do conjunto de dados de treinamento. Os autores também avaliaram uma abordagem de resistência a ataques causais, através da identificação de melhorias na acurácia, quando uma nova instancia era adicionada a base de treinamento. Apesar dessa abordagem ser efetiva, os autores utilizaram um conjunto de dados supervisionado (quando todas as instâncias são previamente classificadas). Essa abordagem não pode ser utilizada em produção, pois as instâncias não são rotuladas previamente e a precisão não pode ser estimada em tempo real. No cenário de detecção de PDF malicioso, Srndic e Laskov [N. Srndic e Laskov, P. 2014] avaliaram um conjunto de ataques contra uma conhecida ferramenta de classificação de PDF maliciosos. Os autores conseguiram reduzir a precisão da classificação de quase 100% para 28%. Os autores também sugeriram que um sistema de classificação de múltiplos classificadores deveria fornecer mais resiliência a tais ataques de adversários, devido à necessidade de evadir vários classificadores complementares.

Poucos autores abordam ataques causais no campo de detecção de intrusão de rede [Wang, G., et al. 2014]. Joseph e seus colegas [Joseph, A. D. e Taft, N. 2009] desenvolveram o ANTIDOTE, que se baseia em um PCA robusto e um limiar de Laplace que é menos susceptível aos ataques de manipulação. No entanto, sua abordagem permanece exposta a ataques exploratórios.

Para o melhor de nosso conhecimento, este é o primeiro trabalho a abordar ataques causais e exploratórios em algoritmos de aprendizagem de fluxo para o campo de detecção de intrusão. Nossa abordagem permanece confiável durante ambos os ataques,

 © 2017 Sociedade Brasileira de Computação

causal e exploratório através de um mecanismo de rejeição e detectores de anomalias específicos a classe.

6. Conclusão

A detecção de intrusão baseada em anomalia tem sido amplamente estudada nos últimos anos. No entanto, apesar de resultados promissores, essa abordagem é pouco utilizada em ambientes de produção. Isso ocorre principalmente devido à ausência de métodos de detecção confiáveis e atualizáveis. O principal problema encontrado é o uso da aprendizagem de máquina em configurações adversárias, em que um atacante tenta evadir o mecanismo de detecção.

Este artigo apresentou um novo método de detecção de intrusão baseado em anomalia, que aborda o uso de aprendizagem de fluxo em configurações adversárias. A abordagem proposta utiliza detectores de anomalias específico a classe. Permitindo assim, a detecção de possíveis tentativas de evasão, e ao mesmo tempo que fornece um mecanismo de detecção confiável e atualizável. A confiabilidade é alcançada rejeitando classificações potencialmente erradas ou tentativas de evasão. Através do conjunto de avaliações efetuadas, o método proposto apresentou sua resistência para ataques causais e exploratórios.

Para trabalhos futuros, busca-se a redução da taxa de rejeição, enquanto ainda resistimos aos ataques adversários. Para este fim, planeja-se empregar uma abordagem híbrida, que se baseie em algoritmos de aprendizagem de fluxo e algoritmos de aprendizagem de máquinas tradicionais.

Referências

Alsabti, K., Ranka, S. e Singh, V. (1997). An efficient k-means clustering algorithm. Electical engineering and Computer Science, v. 43 p. 2-7.

Axelsson, S. (2000). The Base-Rate Fallacy and the Difficulty of Intrusion Detection. ACM Transactions on Information and System Security, v. 3, n. 3, p. 186–205.

Bifet, A., Gavalda, R. e Gavaldà, R. (2007). Learning from Time-Changing Data with Adaptive Windowing. Sdm, v. 7, p. 2007.

Cavalcanti, G. D. C., Oliveira, L. S., Moura, T. J. M. e Carvalho, G. V. (2016). Combining diversity measures for ensemble pruning. Pattern Recognition Letters, v. 74, p. 38–45.

Corona, I., Giacinto, G. e Roli, F. (2013). Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. Information Sciences, v. 239, p. 201–225.

Denning, D. E. (1987). An intrusion-detection model. Proceedings - IEEE Symposium on Security and Privacy, n. 2, p. 118–131.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Ferná Ndez, G. e Vá Zquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, v. 28, p. 18–28.

He, H., Chen, S., Li, K. e Xu, X. (2011). Incremental learning from stream data. IEEE Transactions on Neural Networks, v. 22, n. 12, p. 1901–14.

© 2017 Sociedade Brasileira de Computação

Joseph, A. D. e Taft, N. (2009). ANTIDOTE: Understanding and Defending against. Traffic, SIGCOMM, p. 1–14.

Jyothsna, V., V Rama Prasad, V. e Munivara Prasad, K. (2011). A Review of Anomaly based Intrusion Detection Systems. International Journal of Computer Applications, v. 28, n. 7, p. 26–35.

Kontaki, M., Gounaris, A., Papadopoulos, A. N., Tsichlas, K. e Manolopoulos, Y. (2011). Continuous Monitoring of Distance-Based Outliers over Data Streams.

Maggi, F., Robertson, W., Kruegel, C. e Vigna, G. (2009). Protecting a moving target: Addressing web application concept drift. Lecture Notes in Computer Science, v. 5758 LNCS, p. 21–40.

Nelson, B., Barreno, M., Chi, F. J., et al. (2008). Exploiting machine learning to subvert your spam filter. In Proceedings of the First Workshop on Large-scale Exploits and Emerging Threats (LEET), n. April, p. Article 7.

N. Srndic e Laskov, P. (2014). Practical Evasion of a Learning-Based Classifier: A Case Study, IEEE Symposium Security and Privacy, p. 197-211.

Oliveira, L. S., Sabourin, R., Bortolozzi, F. e Suen, C. Y. (2002). Automatic recognition of handwritten numerical strings: A Recognition and Verification strategy. IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 24, n. 11, p. 1438–1454.

Sommer, R. e Paxson, V. (2010). Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. 2010 IEEE Symposium on Security and Privacy, v. 0, n. May, p. 305–316.

Tygar, J. D. (2011). Adversarial machine learning. IEEE Internet Computing, v. 15, n. 5, p. 4–6.

Viegas, E. K., Santin, A. O. e Oliveira, L. S. (2017) Toward a reliable anomaly-based intrusion detection in real-world environments. Computer Networks, v. 127, p. 200-216.

Wang, G., Barbara, S., Wang, T., Zheng, H. e Zhao, B. Y. (2014). Man vs. Machine: Practical Adversarial Detection of Malicious Crowdsourcing Workers. the 23rd USENIX Security Symposium, p. 239–254.

 © 2017 Sociedade Brasileira de Computação

Atraindo, conhecendo e repudiando atacantes VoIP

Daniel Bauermann[1,2], **Giovani Luís Emmert**[1]

[1]Faculdade IENH (IENH)
Rua Frederico Mentz, 526 – 93.525-360S – Novo Hamburgo – RS – Brasil

daniel.b@ienh.com.br, giovaniemm@hotmail.com

[2]Universidade Feevale
ERS-239, 2755 – 93.525-075 – Novo Hamburgo – RS – Brasil

danielbauermann@feevale.br

Abstract. *The expansion of the use of voice over IP systems has attracted a growing number of people who try to find vulnerabilities that may cause harm to organizations. To understand how these people exploit the systems, it is crucial know their way of acting. For this, different works uses honeypot technology with the objective of investigating the caracteristics of many kinds of possible attacks. However, few works present a system that can repulse attackers interested in exploiting vulnerable systems, especially for VoIP services. In this work, we propose a framework capable of compiling attack data to increase the security of an IP telephony network.*

Resumo. *A expansão do uso de sistemas de voz sobre IP tem atraído um número cada vez maior de indivíduos que buscam vulnerabilidades que podem causar prejuízos às organizações. Para entender de que forma estes indivíduos exploram os sistemas, é fundamental conhecer seu modo de agir. Para isto, diferentes trabalhos fazem uso da tecnologia de honeypots com o objetivo de investigar as características dos diversos ataques possíveis. Entretanto, poucos trabalhos apresentam um sistema capaz de repelir malfeitores interessados em explorar sistemas vulneráveis, principalmente para serviços VoIP. Neste trabalho propomos um arcabouço capaz compilar dados de ataques buscando incrementar a segurança de uma rede de telefonia IP.*

1. Introdução

O uso de sistemas de telefonia Voice over Internet Protocol (VoIP) há muito deixou de ser uma promessa e uma tendência de crescimento para se tornar uma realidade para muitas empresas e organizações. A popularização destes sistemas tem atraído atacantes interessados em usufruir de sistemas vulneráveis ou expostos. A evidência deste crescimento bem como do interesse neste tipo de sistemas foi observada pelo CERT.br[1], originando um estudo de aprofundamento do interesse destes atacantes [Steding-Jessen 2016], no qual se registrou um elevado número de abusos em servidores Session Initiation Protocol (SIP).

[1]O CERT.br é o Grupo de Resposta a Incidentes de Segurança para a Internet brasileira, mantido pelo NIC.br, do Comitê Gestor da Internet no Brasil. É responsável por tratar incidentes de segurança em computadores que envolvam redes conectadas à Internet brasileira. Fonte: http://www.cert.br/sobre/

 © 2017 Sociedade Brasileira de Computação

Este cenário tem motivado diversos pesquisadores a investigar as fragilidades e possíveis proteções para sistemas de telefonia VoIP, gerando obras extensas sobre o assunto, como por exemplo, [Endler e Collier 2007]. Em Nassar [Nassar et al. 2011] podemos encontrar diferentes possibilidades de ataques, além disso, o autor destaca que tais possibilidades exigem diferentes tratamentos de defesa.

Dentre as tecnologias para aprofundamento do conhecimento dos métodos e das técnicas de um atacante, o uso de *honeypots* e *honeynets* tem se apresentado como uma alternativa bastante eficaz, uma vez que simulam ambientes reais e armazenam diversos dados sobre os atacantes, permitindo um estudo detalhado de um ambiente atacado.

O conhecimento aprofundado sobre ataques SIP permite criar medidas para mitigar ataques. Porém, quando existe a necessidade de expor sistemas de telefonia VoIP em uma rede pública, ainda que se implementem tais medidas, os riscos aumentam, já que a central de telefonia se torna exposta a todos. Neste cenário, o uso de *blacklists* com endereços de atacantes é uma técnica que aumenta significativamente a segurança de um sistema de telefonia. No entanto, a atualização de seus registros bem como seus critérios de inclusão nem sempre são claros. Além disso, não se tem conhecimento de listas que representem o cenário nacional.

Diferente de outros trabalhos nos quais o uso de *honeynets* é um meio para levantar dados de atacantes e analisar seu comportamento, no presente trabalho, o uso desta técnica é o meio para criar uma *blacklist* em tempo real para proteger um sistema de telefonia exposto em uma rede pública.

Assim, o objetivo do presente trabalho é desenvolver uma estrutura de proteção para um sistema de telefonia VoIP, fazendo uso de dados coletados através de uma *honeynet*, a fim de incrementar o nível de segurança de perímetro existente em um ambiente real de telefonia.

O restante deste trabalho está organizado como segue. A seção 2 apresenta um visão geral de trabalhos relacionados. Uma breve descrição de VoIP, bem como os ataques possíveis ao sistema são apresentados na seção 3. Na seção 4 descreve o cenário de implementação deste trabalho e descreve a arquitetura criada no decorrer deste estudo para incrementar o nível de segurança da rede onde este trabalho foi desenvolvido. Os resultados encontrados e análises são detalhados na seção 5. A seção 6 encerra com conclusões e perspectivas de trabalhos futuros.

2. Trabalhos relacionados

A questão de segurança em VoIP tem estimulado muitas investigações sobre o assunto, gerando um série de trabalhos que classificam os ataques a sistemas de telefonia conforme seus objetivos ([Thermos e Takanen 2008] dedica um capítulo inteiro sobre o assunto). O trabalho de [Hoffstadt et al. 2012] serviu de base para classificação de ataques SIP usados na presente pesquisa.

Apesar destas e outras tantas fontes contribuírem para o entendimento e classificação de agressores a sistemas VoIP, para Gruber [Gruber et al. 2011] existia a necessidade de entender o comportamento dos atacantes, buscando padrões e assinaturas de ataques. Isto gerou um extenso trabalho investigativo [Gruber et al. 2015] no qual se validou comportamentos semelhantes, mesmo em redes diferentes.

 © 2017 Sociedade Brasileira de Computação

Ainda que Gruber [Gruber et al. 2015] considere que o comportamento de atacantes em diferentes redes seja semelhante, buscou-se para esta pesquisa dados que apontassem o comportamento dos ataques em redes nacionais. Embora o trabalho desenvolvido pelo CERT.br no projeto honeyTARG [CERT.br 2016] apresente dados interessantes sobre o ambiente nacional, não existem dados específicos sobre ataques VoIP em sua coleção. Através da análise de seus dados relativos às tentativas de exploração de portas User Datagram Protocol (UDP), fica explícito que a porta 5060 (padrão do protocolo SIP) lidera as tentativas dos atacantes que objetivam serviços executando em UDP. No entanto, não existem outros detalhes sobre estes atacantes disponibilizados de forma pública. Assim sendo, o presente trabalho faz uso de *honeynets* para coletar tais informações dos agressores.

Assim como em [Nassar et al. 2011] e [Gruber et al. 2015], Carmo [Carmo 2011] também faz uso de *honeynets* para coletar informações de atacantes. Diferentemente destes e outros trabalhos cujo objetivo é compreender padrões de ataques, o presente trabalho tem como objetivo repelir atacantes que tenham intencionado explorar a rede na qual a *honeynet* implementada foi instalada.

Para o controle de *spammers* é comum a adoção de *blacklists* que impendem o recebimento de conteúdo indesejado oriundo de fontes duvidosas. A consulta de uma lista pública onde constam registrados alguns destes endereços Internet Protocols (IPs) ajuda a minimizar o recebimento deste tipo de conteúdo. Há muito tempo é possível consultar diferentes fontes que mantêm este tipo de lista com apoio da comunidade da internet, como por exemplo, [Spa 2016] e [bar 2016], entre outras.

Ainda que existam *blacklists* específicas para VoIP, que auxiliam na construção de uma proteção para centrais telefônicas expostas em redes públicas, seus registros nem sempre atendem aos objetivos desta pesquisa. [Sco 2016], por exemplo, oferece uma lista aberta de IPs que podem ser usados para bloqueio. Mas sua lista não possuía nenhum registro do Brasil até a conclusão deste trabalho, ao contrário do que encontramos em nosso cenário desenvolvido. Já a lista disponibilizada em [Net 2016] oferece abertamente apenas um número limitado de registros, sendo que a lista completa é disponibilizada apenas para clientes da empresa mantenedora da lista. Além disso, não constam detalhamentos dos métodos utilizados para criação desta lista.

Alinhado com os objetivos de Nassar [Nassar et al. 2011], o presente trabalho também se buscou criar estratégias de defesa sem complicação e sem isolamento, a fim de defender um sistema de telefonia IP exposto em uma rede pública.

3. Voice over Internet Protocol

Assim como na telefonia tradicional, também na telefonia digital, ao se tentar conectar com alguém, é necessário ser informado se a pessoa de destino está com o telefone ocupado ou se está sendo chamado. Da mesma forma, a pessoa que está recebendo a chamada deve ser notificada. Este procedimento é conhecido como sinalização [Callado et al. 2007].

Existem diferentes protocolos para o processo de sinalização, sendo os mais comuns: H.323, SIP, Inter-Asterisk eXchange v2 (IAX2), Skinny Client Control Protocol (SCCP), Media Gateway Control Protocol (MGCP) e H.248 ou MEGACO. Ainda que o

© 2017 Sociedade Brasileira de Computação

H.323 tenha sido um dos primeiros protocolos apresentado e utilizado pela indústria, sua complexidade deu espaço para o avanço do protocolo SIP, sendo este um dos mais populares. Assim sendo, na próxima seção é apresentado um detalhamento deste protocolo.

3.1. Session Initiation Protocol

O SIP é um protocolo da camada de aplicação baseado em texto (semelhante ao protocolo Hypertext Transfer Protocol (HTTP)) desenvolvido para estabelecer, manter e terminar ligações telefônicas, videoconferência e outras aplicações multimídia [Callado et al. 2007]. A arquitetura SIP é formada por dois componentes, o agente de usuário e o servidor de rede.

O agente de usuário é responsável por iniciar e terminar as requisições SIP. Já o servidor de rede, nesta arquitetura, se divide em outros três tipos. O (1) servidor de registro, responsável por receber as requisições dos agentes de usuários e manter um banco de dados com informações dos usuários. O (2) servidor *proxy*, responsável por receber pedidos e atendê-los diretamente ou encaminhá-los para outro servidor, trazendo mais desempenho no processo de encaminhamento. E o (3) servidor de redirecionamento, responsável por receber requisições e informar ao cliente o endereço do próximo servidor. Assim, o cliente contata diretamente o próximo servidor [Ghafarian et al. 2016]. No contexto deste trabalho são utilizados apenas servidores de registro.

Os agentes se comunicam com servidores através de transações (ou pedidos). Cada transação gera uma ou mais respostas. A figura 1 ilustra a troca de mensagens em uma comunicação SIP, já considerando que os agentes dos usuários Marcos e João estejam devidamente registrados em um servidor de registro. Marcos inicia a comunicação enviando uma mensagem de requisição para o servidor SIP, solicitando contato com João. O servidor verifica em sua base se João existe e, existindo, avisa a Marcos que é possível comunicar-se com seu destino, informando-lhe o caminho até João. Após estabelecida a sessão, há o tráfego de pacotes de voz (sessão de mídia), que são transportados sob o protocolo Realtime Transport Protocol (RTP). Por fim, a comunicação se encerra através das mensagens representadas pela transação 3 na figura.

Conforme a RFC 3261 [RFC 2016] existem seis tipos de mensagens SIP: `INVITE`, `ACK`, `BYE`, `CANCEL`, `REGISTER` e `OPTIONS`. As mensagens ainda são classificadas em mensagens de requisição e de resposta. Uma transação é formada por, pelo menos, uma mensagem de requisição e uma mensagem de resposta.

3.2. Ataques SIP

Hoffstadt [Hoffstadt et al. 2012], através de experimentos com *honeypots*, caracterizou ataques SIPs em quatro estágios distintos, todos executados através de mensagens do protocolo SIP, conforme descritos a seguir:

3.2.1. Varredura de servidores e dispositivos SIP

Assim como em outros tipos de ataques, a primeira fase é a descoberta de alvos. Isto é feito através de varreduras de redes, buscando informações e portas abertas.

O fato de um dispositivo SIP responder a pacotes `OPTIONS` permite ao atacante usar um "`ping`" para um único IP, ou até mesmo para uma rede inteira, com pacotes

 © 2017 Sociedade Brasileira de Computação

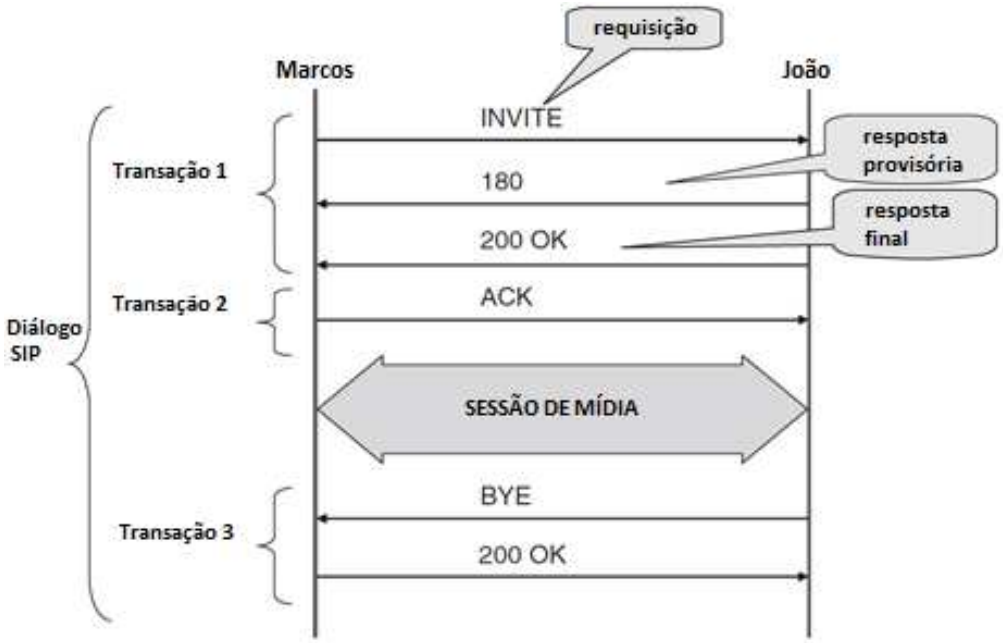


Figura 1. Fluxo de comunicação SIP

`OPTIONS` buscando identificar dispositivos SIP. Em caso de agentes que não implementem o protocolo SIP padrão e só respondam pacotes `OPTIONS` de fontes conhecidas, ainda assim uma varredura é possível. Neste caso, o atacante pode utilizar solicitações do tipo `REGISTER`.

3.2.2. Varredura de ramais

Uma vez identificado um alvo, o atacante passa a executar uma varredura mais aprofundada, buscando ramais no servidor. Isto é feito através de tentativas sistemáticas de registros de ramais sem senhas. Um ramal consiste em uma sequência de dígitos e/ou série de letras. Se um ramal existir, o servidor normalmente responde com uma mensagem `401 UNAUTHORIZED` (não autorizado) ou `403 FORBIDDEN` (senha errada). Se o ramal não existir, uma mensagem `404 NOT FOUND` (não encontrado) é retornada. O resultado desta fase é uma lista completa de ramais existentes.

3.2.3. Registro de ramais

Para registrar um ramal, o atacante tenta adivinhar a senha. Para isto são enviadas múltiplas mensagens `REGISTER` com diferentes senhas para um determinado ramal. Se uma senha é adivinhada, a informação é salva para um posterior registro.

© 2017 Sociedade Brasileira de Computação

3.2.4. Fraude

A fraude acontece sempre que uma pessoa gerar custos usando um ramal ”sequestrado”. O atacante pode, por exemplo, fazer chamadas internacionais, usando as funcionalidades do VoIP. Outro benefício de um ramal sequestrado está no fato do atacante permanecer anônimo.

Em termos de mensagens SIP, o atacante primeiro envia uma mensagem `REGISTER` com a senha correta. Após a mensagem `200 OK` do servidor, o invasor pode iniciar chamadas usando mensagens `INVITE`.

4. Implementação

Esta seção descreve como foi implementada a *honeynet* usada para coleta de dados de atacantes que alvejavam a rede em que este trabalho foi exposto. Na sequência é descrito o arcabouço desenvolvido para compilar o conjunto de dados coletados e banir IPs mal intencionados que tentaram explorar os serviços propositalmente expostos.

4.1. *Honeynet*

O software de código aberto utilizado neste trabalho foi o Dionaea[2], um *honeypot* de baixa interação [Weissheimer Júnior 2008] com capacidade de emular serviços de telefonia VoIP, além de possuir módulos para outros serviços.

Foram utilizados dois servidores virtuais rodando Ubuntu Linux 12.04 LTS 32 *bits*[3] nos quais o Dionaea foi instalado. Cada servidor possui 4 GB de memória RAM, um processador de 2 núcleos com *clock* de 1,2 GHz e 100 GB de espaço em disco.

Ainda que o uso de *honeypots* se apresente como uma alternativa excelente para coletar dados e formas de ataque, conforme [Safarik et al. 2012], um atacante tende a perder o interesse por um alvo ao perceber que está atacando um ambiente simulado. Por isto, diferentes trabalhos fazem uso de *honeynets* a fim de maximizar a atração de malfeitores para sua rede.

Assim sendo, objetivando maximizar a atração de malfeitores à rede na qual este trabalho foi desenvolvido, cada *honeypot* foi instalado em um segmento de rede, sendo o primeiro servidor disponibilizado com um IP real da faixa 200.x.y.z e o segundo na faixa 177.x.y.z.

4.2. Arquitetura do *Honey To Block*

Uma vez que a implementação da *honeynet* foi superada, se fez necessário o desenvolvimento de um sistema capaz de concentrar os dados de forma única. Para isto, foi criado em Python[4] um software denominado *Honey To Block*, com o objetivo de receber os dados dos *honeypots*, agrupá-los de forma única e disponibilizá-los para a consulta de um *firewall*. A figura 2 apresenta a arquitetura criada.

O `H2B-server` é o servidor responsável por atender as requisições dos demais componentes deste arcabouço. Atualmente seu papel se concentra em duas atividades

[2]https://github.com/rep/dionaea
[3]https://www.ubuntu.com/
[4]https://www.python.org/

 © 2017 Sociedade Brasileira de Computação

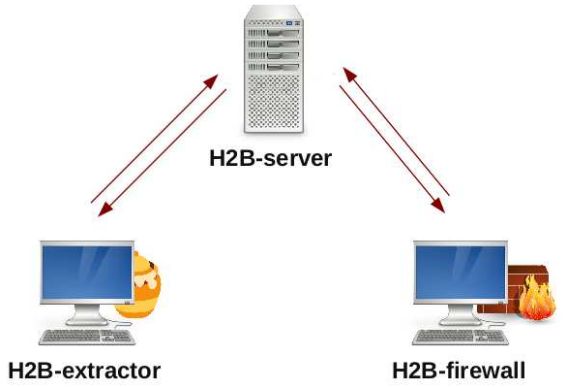


Figura 2. Arquitetura Honey to Block

principais, que são: (1) receber a lista de novos atacantes coletadas em cada *honeypot*, organizando-os de forma única; e (2) informar ao *firewall* quais são os novos atacantes registrados. Estas interações consideram o tempo entre um contato e outro a fim de minimizar o volume de tráfego entre clientes e servidor.

Já o `H2B-extactor` tem o papel de extrair o endereço IP dos atacantes que tentaram explorar o módulo VoIP do *honeypot*, informando-os ao servidor. Isto é feito através da análise dos *logs* do Dionaea em ciclos de tempo. Entre cada execução são extraídos somente os *logs* registrados naquele intervalo de tempo. Assim, podem existir registros duplicados de IPs, como o caso de um atacante que está tentando fazer uma nova exploração. Caso isto ocorra, o servidor será o responsável por manter a unicidade destes registro, embora mantenha também armazenada a informação de todas as notificações do *honeypot*, permitindo análises futuras.

Por fim, o `H2B-firewall` é o cliente que está rodando em uma estrutura de *firewall* com a responsabilidade de proteger um sistema de telefonia IP. Ele tem o papel de consultar o servidor (`H2B-server`) para buscar uma lista de quais atacantes serão banidos da rede, evitando qualquer tentativa de interação destes com o sistema de telefonia. Esta consulta ocorre em períodos de tempo determinados e, também, busca somente os novos malfeitores registrados desde a última consulta. Uma vez obtida a lista de quais atacantes deverão ser banidos, o `H2B-firewall` aplica as regras sobre o filtro de pacotes em execução, carregando somente as regras necessárias para tal negação, não alterando as demais regras criadas no ambiente.

5. Resultados

Desde que foram implementados os *honeypots*, juntos, já somam 15.114 horas de execução, sendo 8.193 horas no primeiro *honeypot* e 6.921 horas no segundo. Neste período foram registrados 2.121.660 ataques em diferentes serviços. Especificamente para o serviço de VoIP foram contabilizados 308.631 tentativas de exploração do serviço, até o fechamento deste artigo.

Para validar os dados coletados em nossos *honeypots*, utilizamos as estatísticas [CERT.br 2017] disponibilizadas pelo projeto honeyTARG [CERT.br 2016], coordenado pelo CERT.br. Estas estatísticas são disponibilizadas diariamente, sendo os dados apresentados neste trabalho referente ao dia 09/07/2017.

© 2017 Sociedade Brasileira de Computação

Na tabela 1 podemos observar que o principal protocolo atacado nos serviços disponibilizados pelo honeyTARG é o protocolo Transmission Control Protocol (TCP). A diferença entre o primeiro protocolo (TCP) e o segundo (UDP), em volume de dados trafegados, é significativa. O mesmo comportamento foi encontrado em nossos *honeypots*.

Tabela 1. Protocolos atacados em 09/05/2017 no honeyTARG.

#	Protocol	Total		Max	Avg
01	TCP	1.23 GB	85.45 %	108.63 KB/s	14.28 KB/s
02	UDP	209.20 MB	14.49 %	18.35 KB/s	2.42 KB/s
03	ICMP	929.58 KB	0.06 %	121.49 B/s	10.76 B/s
04	Others	0.00 B	0.00 %	0.00 B/s	0.00 B/s

A figura 3 apresenta o número de requisições registradas nos *honeypots* expostos em nosso trabalho. Cada gráfico nesta figura representa o comportamento coletado em um *honeypot* exposto. No eixo *x* do gráfico são apresentados os números das portas atacadas, sendo portas TCP representadas em vermelho e portas UDP representadas em verde. O eixo *y* representa o número total de conexões recebidas em cada porta. Para facilitar o entendimento, o número de conexões por porta é apresentado no topo de cada barra. São apresentadas apenas as 20 primeiras portas atacadas, ordenadas de forma decrescente, para facilitar a visualização.

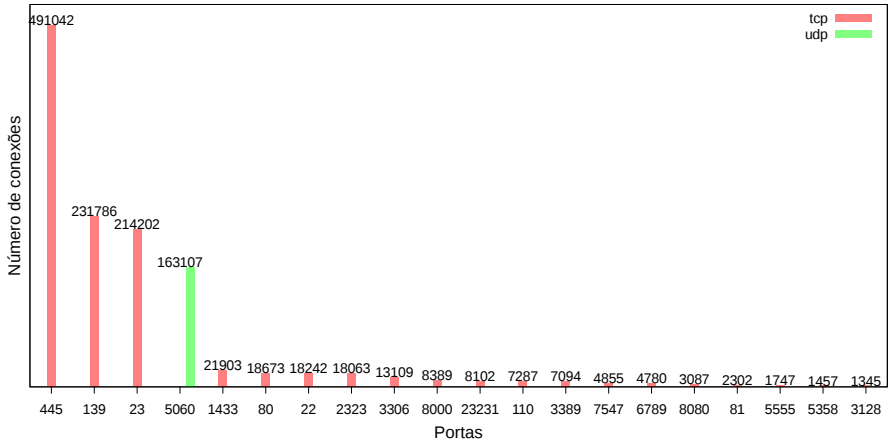
Observa-se que as portas TCP atacadas nos alvos expostos no presente trabalho são as mesmas encontradas nas estatísticas do projeto honeyTARG [CERT.br 2017] (exceto a porta 110 que não possui registros no *honeypot 2*, dentre as 20 primeiras portas apresentadas). Ainda que a distribuição do volume de ataques destas portas dentre nossos *honeypots* divirjam do projeto honeyTARG, pode-se perceber que os serviços buscados pelos atacantes são os mesmos. O volume e distribuição de ataques varia conforme dias analisados no honeyTARG.

Já para as portas UDP, encontramos registros significativos em nossos *honeypots* apenas para a porta 5060, sendo inexpressivos os números de ataques registrados nas demais portas UDP de nossos alvos, em relação ao total de ataques registrados. Este comportamento é o mesmo observado no dia 09/07/2017 das estatísticas do projeto honeyTARG, onde o tráfego SIP representa quase 98% do total de tráfego UDP. Considerando que o objetivo do `H2B-extactor` é coletar especificamente os dados dos ataques VoIP, os demais serviços UDP eventualmente explorados pelos atacantes não são significativos em nosso contexto.

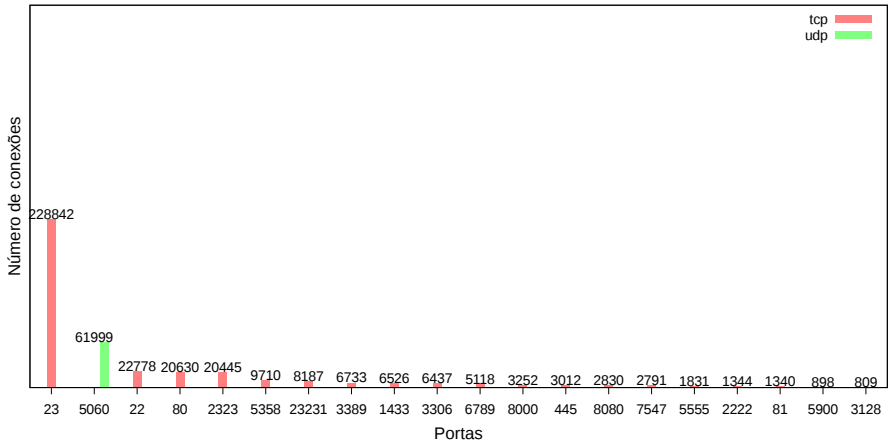
O segundo *honeypot* acumula um número menor de ataques em relação ao primeiro, conforme podemos observar na figura 3. Pode-se observar que os serviços atacados nos dois alvos expostos são um pouco diferentes. Enquanto que a porta 5060 UDP é o segundo serviço mais atacado no *honeypot* 2, demonstrado na figura 3(b), este mesmo serviço é apenas o quarto mais desejado no *honeypot* 1, conforme a figura 3(a).

A figura 4 apresenta em cada um de seus eixos o total acumulados de IPs por número de ataques. Na primeira barra da esquerda, por exemplo, podemos observar que 1.566 endereços IPs diferentes geraram um único ataque em nossos *honeypots*. No outro extremo do figura, o eixo da direita representa um único atacante com 176 tentativas catalogadas em nosso banco de dados.

 © 2017 Sociedade Brasileira de Computação



(a) Honeypot 1



(b) Honeypot 2

Figura 3. Número de ataques por portas.

Destacamos a representatividade de atacantes com um baixo número de entradas em nossas observações. De um total de 3.109 endereços IPs coletados, aproximadamente 84%, ou 2.624 endereços, fizeram até 5 ataques apenas. Este baixo número de tentativas de ataques não despertaria suspeita sobre estes atacantes e os mesmos possivelmente não seriam denunciados para uma *blacklist*, por exemplo.

Para comprovar a eficácia dos bloqueios de atacantes registrados na *honeynet* e barrados em nosso servidor *firewall* que protege nosso ambiente de produção real, geramos *logs* para cada tentativa de exploração oriunda de algum dos endereços IPs catalogados previamente.

O volume destes *logs* é bastante grande, pois além de nossos registros, são gravadas outras informações características do ambiente e necessárias para o dia a dia das operações dos serviços que servem como barreira à rede de produção. Por isto, não é mantido um histórico extenso destes registros. Analisamos dados provenientes de 28 dias de proteção, compreendidos entre os dias 29/01/2017 e 26/02/2017.

Neste período, de um total de 343.173 ataques de IP catalogados pelo servidor de telefonia em produção e banidos com o apoio das informações da *honeynet*, 76.319

© 2017 Sociedade Brasileira de Computação

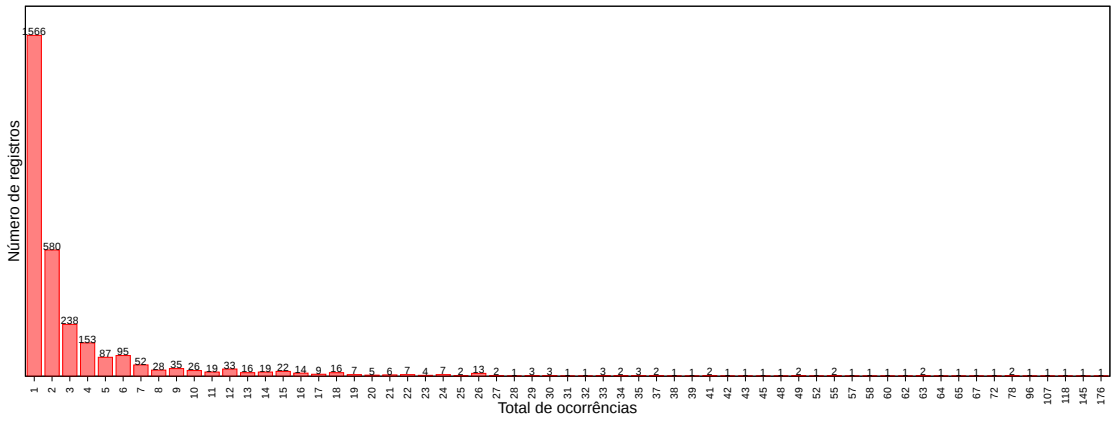


Figura 4. Quantidade de IPs por número de ataques

registros tentaram explorar especificamente a porta 5060 disponibilizada em nosso ambiente de produção, ou seja, cerca 22% dos ataques tinham como objetivo a exploração dos serviços de VoIP. Vale ressaltar que embora nosso objetivo tenha sido proteger especificamente o serviço de telefonia, com o banimento destes atacantes em nossa rede, conseguimos ainda proteger outros serviços que os atacantes buscavam, ou seja, 267.034 tentativas de ataques não eram direcionadas especificamente para VoIP.

Destacamos na figura 5 os 20 IPs com maior número de ataques bloqueados. A primeira barra da esquerda apresenta o IP 51.15.143.238 com 67.485 tentativas de ataque no período analisado. Chama a atenção que os 5 IPs com maior número de tentativas, juntos, somam 305.066, ou seja, quase 89% dos ataques barrados;

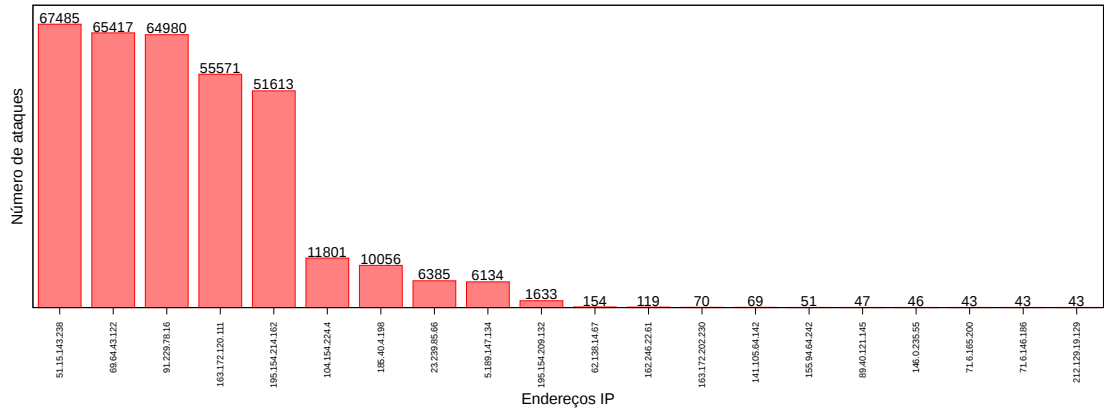


Figura 5. Quantidade de ataques barrados por endereços IPs

No período que os *logs* foram analisados, contávamos com 2.425 IPs únicos em nossa base de dados. Estes atacantes tentaram explorar os serviços de VoIP, ”ofertados” em nossos *honeypots*, desde o início de nosso projeto. Desta base, no período de 28 dias analisados nos *logs*, 204 endereços únicos foram barrados, graças à coleta e filtragem prévia destes atacantes em nossa *honeynet*. Ou seja, 8% dos atacantes que tentaram explorar os serviços VoIP dos *honeypots*, também tentaram explorar os serviços de um servidor de telefonia de produção, expostos na Internet. Ainda assim, 343.173 tentativas de ataques que foram bloqueadas.

 © 2017 Sociedade Brasileira de Computação

6. Conclusões e trabalhos futuros

A popularidade do serviço de telefonia IP tem atraído atacantes que buscam explorar os recursos disponibilizados por esta tecnologia em benefício próprio, causando prejuízos para as organizações que proveem este serviço.

Diferentes pesquisas tem sido desenvolvidas para criar mecanismos de defesas contra atacantes de VoIP. Dentre estes mecanismos, o uso de *honeypots* e *honeynets* tem se apresentando como uma alternativa para conhecer detalhes dos atacantes. Neste trabalho utilizamos os dados coletados de *honeypots* para incrementar o nível de proteção de uma estrutura de *firewall* que protege um sistema de telefonia exposto de forma pública.

Através da implementação de um arcabouço de proteção, conseguimos atrair, conhecer e, posteriormente, repudiar atacantes que tentaram explorar serviços VoIP em nosso ambiente. O registro e a análise de *logs* mostrou-nos a eficiência do mecanismo de proteção criado.

Embora nosso objetivo de proteger o sistema de telefonia exposto publicamente tenha sido plenamente atingido, surpreendeu-nos descobrir que o repúdio dos possíveis atacantes VoIP de nossa rede também contribuiu para a proteção de outros serviços pelos quais os atacantes também buscavam explorar. Por isto, podemos afirmar, que tivemos benefícios adicionais em nossos mecanismos de proteção de rede.

Como trabalhos futuros, pretendemos investigar o motivo do baixo número de ataques por IP registrado em nossos servidores. Será perda de interesse como apontado por [Safarik et al. 2012] ou será o nível de interatividade do *honeypots* expostos?

Referências

(2016). Barracuda reputation block list (brbl). http://www.barracudacentral.org/rbl.

(2016). Protect you asterisk voip server from hackers and voip hijackers. http://www.networksystemssolutions.eu/voipblocklist.php.

(2016). Sip: Session initiation protocol. https://www.ietf.org/rfc/rfc3261.txt.

(2016). The spamhaus project. https://www.spamhaus.org/.

(2016). Voip blacklist. http://www.voipbl.org/.

Callado, A., Fernandes, G., Silva, A., Barbosa, R., Sadok, D., e Kelner, J. (2007). Construção de redes de voz sobre ip. 25^o *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 11–58.

Carmo, Rodrigo do e Nassar, M. e. F. O. (2011). Artemisa: an open-source honeypot back-end to support security in voip domains. *IFIP/IEEE International Symposium on Integrated Network Management*, páginas 361–368.

CERT.br (2016). Projeto honeypots distribuídos. http://honeytarg.cert.br/honeypots/index-po.html.

CERT.br (2017). Distributed honeypots project. https://honeytarg.cert.br/stats/flows/current/.

Endler, D. e Collier, M. (2007). *Hacking Exposed VoIP*. McGraw-Hill, New York.

© 2017 Sociedade Brasileira de Computação

Ghafarian, A., Seno, S. A. H., e Dehghani, M. (2016). An empirical study of security of voip system. *SAI Computing Conference*, páginas 1031–1036.

Gruber, M., Fankhauser, F., Taber, S., Schanes, C., e Grechenig, T. (2011). Security status of voip based on the observation of real-world attacks on a honeynet. *The Third IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT)*, páginas 1041–1047.

Gruber, M., Hoffstadt, D., Aziz, A., Fankhauser, F., Schanes, C., Rathgeb, E., e Grechenig, T. (2015). Global voip security threats - large scale validation based on independent honeynets. *IFIP Networking Conference (IFIP Networking), 2015*, páginas 1–9.

Hoffstadt, D., Marold, A., e Rathgeb, E. (2012). Analysis of sip-based threats using a voip honeynet system. *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, páginas 541–548.

Nassar, M., State, R., e Festor, O. (2011). Voip honeypot architecture. *2011 10th IFIP/IEEE International Symposium on Integrated Network Management*, páginas 109–118.

Safarik, J., Voznak, M., Rezac, F., e Macura, L. (2012). Malicious traffic monitoring and its evaluation in voip infrastructure. *Telecommunications and Signal Processing*, páginas 259–262.

Steding-Jessen, Klaus e Ceron, J. a. M. e. H. C. (2016). Anatomia de ataques a servidores sip. http://www.cert.br/docs/palestras/certbr-ctir2013-1.pdf.

Thermos, P. e Takanen, A. (2008). *Securing VoIP Networks*, capítulo 3, páginas 53–125. Addison-Wesley.

Weissheimer Júnior, Carlos Alfredo e Bastos, E. L. (2008). Honeynet - estudo teórico e experimentação. *Seminário de Pós-Graduação*.

 © 2017 Sociedade Brasileira de Computação

Santos': Algoritmo para Detecção de Ataques do Tipo *Stepping-stones*

Amanda S. Santos[1], **Marcus A.R Tenorio**[1], **Adriano A. Santos**[1], **Andrey Brito**[1]

[1]Departamento de Sistemas e Computação – Universidade Federal de Campina Grande
58.429-140 – Campina Grande – PB – Brasil

amandasouza@lsd.ufcg.edu.br, mart@lsd.ufcg.edu.br

adriano@copin.ufcg.edu.br, andrey@computacao.ufcg.edu.br

Abstract. *Currently, detection of incidents related to cyber threats is considered a challenging task. The increasing number of threats in the network makes information security as a significant topic for discussion in the security community, conducting studies that propose or implement solutions for privacy and protection of users on the network. Despite the methods to carry out attacks, the stepping-stones method stands out because it is an intrusion technique that allows maintaining the anonymity of the invaders and that uses a chain of intermediate machines, which are interconnected, through remote connections. In the present research, an algorithm for detecting and classifying intrusions by stepping-stones is proposed to sort the remote connections as coming from intruders or from legitimate users. The experiments were carried out in three stages: The first one being designated in the definition of a classification profile of the connections; The second to apply the algorithm to an international database; in addition, the third for validation with actual attacks. The results indicated a statistical significance for the profile classification (*97.5% *accuracy), in the verification process as the attack detection method (*100% *accuracy) and validation in a real environment (*95% *accuracy).*

Resumo. *Atualmente, a detecção de incidentes relacionados às ameaças cibernéticas é considerada uma tarefa desafiadora e o progressivo número de ciberataques na rede acentua a importância da segurança da informação como um tema para discussões na comunidade de segurança, conduzindo estudos que proponham ou implementem soluções de privacidade e proteção aos usuários na rede. Dentre os métodos para a realização de ataques, destaca-se o* stepping-stones*, por se tratar de uma técnica de intrusão que permite manter o anonimato dos invasores e que utiliza uma cadeia de máquinas intermediárias, e interligadas entre si, por conexões remotas. É proposto, na presente pesquisa, um algoritmo de detecção e classificação de invasões por* stepping-stones*, com a finalidade de classificar as conexões remotas como provenientes de intrusos ou de legítimos usuários. Os experimentos foram realizados em três etapas, sendo a primeira designada na definição de um perfil de classificação das conexões; a segunda para aplicação do algoritmo a uma base de dados internacional; e a terceira para validação em ataques reais. Os resultados apontaram uma significância estatística na classificação de perfis (precisão de* $97,5\%$*), no processo de verificação do método de detecção dos ataques (precisão de* 100%*) e validação em um ambiente real (precisão de* 95%*).*

 © 2017 Sociedade Brasileira de Computação

1. Introdução

O avanço do poder computacional e a popularidade da Internet acentuam a importância da segurança da informação como um tema para discussões na comunidade científica. Independente do desenvolvimento das pesquisas para implementação de mecanismos de segurança cibernética, milhões de ciberataques são bem sucedidos todos os anos e, dentre esses, muitos não são reportados ou mesmo acontecem sem o conhecimento da vítima [Herd and Kriendler 2013]. Em uma ação maliciosa, os intrusos na Internet lançam os ataques às vítimas de forma indireta e sofisticada, com o propósito de diminuir as hipóteses de serem descobertos. Um método usualmente utilizado para atingir o anonimato na rede e burlar os recursos de segurança é denominado por *stepping-stones* [Kuo et al. 2010].

A intrusão por *stepping-stones* é uma técnica que mantém o anonimato do cibercriminoso, através do envio de ataques por uma cadeia de máquinas interligadas entre si por acessos remotos [Ding and Huang 2011]. Esse ataque, propicia ao intruso, a habilidade de acessar o sistema comprometido, anonimamente, para roubar informações confidenciais e vendê-las na rede, ou mesmo criptografar os dados do computador alvo e reivindicar um resgate para a concessão de acesso aos dados pela vítima.

O primeiro procedimento citado na literatura para detectar intrusões por *stepping-stones* foi desenvolvido por [Staniford-Chen and Heberlein 1995], e conforme alguns trabalhos sugerem [Daud et al. 2015, Kumar and Gupta 2016, Huang et al. 2016], as estratégias desenvolvidas na detecção de intrusos por *stepping-stones* estão em constante aprimoramento. Entretanto, a proteção do computador da vítima é restrita às limitações dos algoritmos na composição de ataques sofisticados, fazendo necessário o surgimento de técnicas elaboradas para proteger os recursos tecnológicos dos diferentes níveis de ameaças atuais [Global 2013].

Nesse cenário, o principal objetivo deste artigo é propor um algoritmo para detectar invasões por *stepping-stones* para computadores caracterizados como alvos finais durante um ciberataque em tempo de resposta. Em comparação aos principais trabalhos relacionados, como os propostos em [Ding and Huang 2011, Zhang 2014], a abordagem atual aplica um limite de erro de [Gosset 1908] para a incidência de alerta falso-positivo ao resultado calculado, dispensando o monitoramento do tráfego de todas as máquinas envolvidas na cadeia de *stepping-stones*.

Como contribuições principais desse artigo, pode-se citar: (i) a confirmação da finalidade das técnicas propostas nos trabalhos de [Ding and Huang 2011, Zhang 2014], aplicadas em tempo real do ataque; (ii) a proposta da aplicação da margem de erro de [Gosset 1908] para suprimir os alertas atípicos provenientes de alterações na rede; (iii) a inclusão do intervalo de confiança para ampliar o índice de detecção proposto nos algoritmos anteriores; e (iv) o desenvolvimento de um algoritmo e validação em ambientes reais, por três arquiteturas distintas, e dentre estas, uma técnica configurada em diferentes países.

O artigo está organizado da seguinte forma: a Seção 2 detalha os trabalhos relacionados. A Seção 3 aborda os conceitos aplicados ao método proposto no atual estudo. A Seção 4 descreve o algoritmo proposto para a construção de um modelo de detecção de intrusos. Na Seção 5 é abordado configuração do ambiente e o detalhamento do planejamento dos experimentos. A Seção 6 apresenta considerações finais e conclui o artigo.

 © 2017 Sociedade Brasileira de Computação

2. Trabalhos Relacionados

O tema *stepping-stones* foi abordado em inúmeras técnicas descritas na literatura. Entretanto, no presente artigo, como ponto principal, foram descritos apenas os trabalhos que colaboraram diretamente para o desenvolvimento do método de detecção de intrusos.

O trabalho realizado em [Zhang and Paxson 2000], desenvolveu um método para detectar intrusos em *links* na Internet, por meio dos terminais interativos. O método introduziu a ideia de *Inter-gaps* e *Intra-gaps* por intermédio da correlação do tráfego ON/OFF, gerados pelo início ou pausa da digitação do usuário. O algoritmo desenvolvido pelos autores, considera como importantes, as informações contidas nos cabeçalhos dos pacotes, tais como tempo e tamanho, e apresentou um nível de precisão relevante em seus resultados para conexões estabelecidas em uma rede local. No entanto, o algoritmo evidenciou limitações na distinção entre *stepping-stones* para *hosts* configurados a longa distância ou em diferentes países.

A proposta apresentada em [Ding and Huang 2011], sugeriu um algoritmo para estimar a distância das conexões a partir do cálculo de ida e volta dos pacotes. Os autores introduziram os conceitos de *u*RTT, com abordagem ao conceito inicial para um método de detecção baseado nas informações coletadas no computador da vítima. A distância entre as conexões eram calculadas conforme a mensuração do *gap* existente entre a máquina do intruso e o computador a ser invadido. O índice de precisão máximo para a categorização das longas conexões foi cerca de 80% na classificação de intrusos, e a presença de 15% de valores falso positivos. Entre os restrições apresentadas pelos autores, são citadas, o número limitado de longas conexões e experimentos realizados, como a ausência de métodos para detectar *scripts* automáticos durante os ataques de *stepping-stones*.

Em [Zhang 2014], foi definido uma abordagem de detecção por meio do cruzamento de pacotes durante a comunicação das conexões. A análise de modo *offline*, decorreu da coleta de dados de cada computador envolvido em uma longa cadeia de *stepping-stones*. Embora os resultados apresentem um alto índice de detecção de intrusos e uma pequena incidência de falso positivos, o algoritmo tem limitações consideráveis. A ausência de resposta ao ataque em tempo real através da coleta de informações em todos os computadores envolvidos em uma longa conexão, apresentou-se inviável, pois, de fato, não há o conhecimento legítimo da localização de todos os *hosts* comprometidos em uma invasão por *stepping-stones*.

O presente trabalho avança nas contribuições dos algoritmos propostos em [Ding and Huang 2011, Zhang 2014] ao apresentar um limite de falha na detecção de intrusos por *stepping-stones* em uma cadeia de máquinas configuradas em diferentes países.

Este limite adiciona uma limiar aceitável de erro de [Gosset 1908], para desconsiderar os dados com valores incomuns, causados por fatores externos como, por exemplo, o *jitter* intrínseco da rede. A proposta atual assume as fragilidades dos trabalhos anteriores e alerta a incidência de longas cadeias por *stepping-stones* em tempo de ataque,através de informações coletadas exclusivamente no computador da vítima.

A proposta abordada nesse trabalho, apresenta um alto índice de precisão, mesmo em ataques realizados por *scripts* ou interação humana.

© 2017 Sociedade Brasileira de Computação

3. Detectando *Stepping-Stones*

As invasões por intermédio do método de *stepping-stones* são consideradas como uma técnica popularmente utilizada para atingir o anonimato na rede. Conforme [Kuo et al. 2010], a metodologia objetiva realizar cibercrimes mediante o uso de conexões entre máquinas previamente vulneráveis e interligadas por inúmeras comunicações remotas.

O estudo realizado em [Omar et al. 2008], definiu dois termos importantes para classificar os tipos de conexões remotas na execução de um ataque por *stepping-stones*. As longas conexões ($C_1 ... C_n$) são caracterizadas pelo estabelecimento de uma conexão remota proveniente de um intruso, estendido da máquina 1 até a máquina n, por meio de inúmeras conexões em cadeia, conforme pode ser apresentado na Figura 1 pela comunicação entre os computadores $i-1$, i e $i+1$. As curtas conexões são definidas por uma conexão remota resultante de um usuário normal C_1, estabelecida entre a máquina 1 até o computador seguinte ($i-1$).

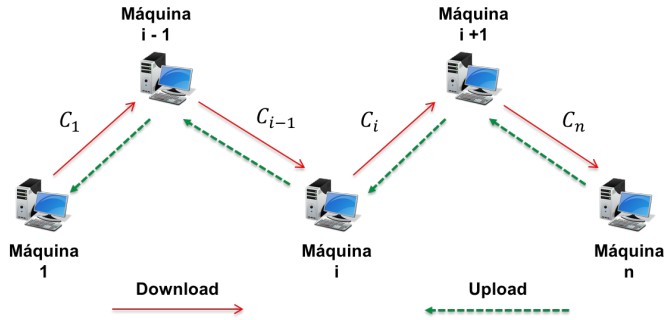


Figura 1. Conexão por *stepping-stones*

As comunicações entre os computadores em uma longa conexão, conforme apresentados na Figura 1, apresentam duas definições distintas, classificadas pelo direcionamento dos dados entre as máquinas. Para o tráfego do tipo *download*, a comunicação, por exemplo, de uma conexão por *stepping-stones*, é direcionada do *host* 1 para a máquina n. Em contrapartida, a comunicação por *upload*, é sucedida do *host* n para a máquina 1.

Para realizar uma ação maliciosa, o intruso de posse a uma coleção de contas remotas, conecta-se individualmente a computadores previamente comprometidos em outros ciberataques. Uma vez estabelecida uma longa cadeia de máquinas, entre as múltiplas seções remotas conectadas ao alvo final, conforme apresentado na Figura 2, o endereço virtual visivelmente reconhecido está diretamente ligado ao computador da vítima, dificultando o rastreamento da origem do ataque.

O exemplo contido na Figura 2, classifica as características para uma rede constituída entre as máquinas visíveis. As conexões remotas, denotam um perfil anônimo do verdadeiro endereço do cibercriminoso para a vítima, caracterizado pela visibilidade do *host* antecessor, que está conectado diretamente ao computador passível a ataques, e a omissão das demais máquinas participantes da cadeia de *stepping-stones* [Wu and Huang 2010].

 © 2017 Sociedade Brasileira de Computação

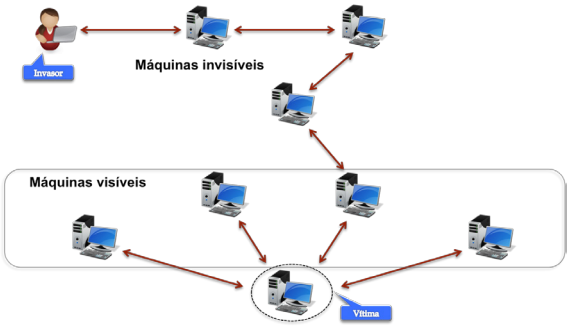


Figura 2. Proteção da vítima no final da cadeia.

3.1. Distribuição *u*RTT

O protocolo TCP possui características para a definição de mecanismos que influenciem a integridade das mensagens trocadas entre as máquinas no canal de comunicação. A nomenclatura que define os atributos na troca de mensagens em um ataque por *stepping-stones* são definidos por RTT e *u*RTT.

O atributo *Round-Trip Time* ou RTT é definido pelo tempo calculado entre a disponibilidade do envio de uma informação na rede para um destinatário, até a entrega do pacote de confirmação de recebimento ao remetente. O cálculo do RTT é estimado para definir a média de tempo que o emissor pode esperar pela confirmação de entrega da mensagem ao destinatário [Mittal et al. 2015].

A distribuição *Upload Round Trip Time* ou *u*RTT é o cálculo do tempo entre o recebimento do primeiro e o segundo pacote de requisição para o servidor. Em um ataque por *stepping-stones* estima-se um gradativo número de perda de pacotes, devido ao longo tempo para o encaminhamento da mensagem até o destinatário. Assim, o tempo da mensagem no canal de comunicação, entre as máquinas envolvidas no ataque, será maior que o tempo de ida e volta do pacote calculado pelo algoritmo do RTT [Zhang 2014].

Como um exemplo, na Figura 3 é ilustrada a troca de pacotes entre computadores envolvidos em uma cadeia de máquinas por *stepping-stones*. O computador cliente envia um pacote de requisição para a máquina-servidor. O servidor recebe o pacote requisição e envia um pacote de resposta para o cliente. O tempo calculado entre o envio do pacote requisição e o recebimento do pacote resposta é definido como RTT da conexão. O RTT pode ser apenas calculado por meio da coleta dos pacotes no computador do cibercriminoso.

Ainda na Figura 3, é definido o cálculo do tempo por *u*RTT. O cliente envia um pacote de requisição para o servidor e, este, ao receber o pacote de requisição, inicia a contagem do tempo *u*RTT e envia como confirmação, um pacote de resposta. O cliente ao receber o pacote de resposta, envia novamente um pacote de requisição para o servidor. O servidor ao considerar uma segunda requisição recebida do cliente, finaliza o cálculo do *u*RTT, determinando o valor em microssegundos. A sequência do cálculo do *u*RTT é reiniciada, e a contagem dos pacotes de requisição é zerada. O *u*RTT é calculado no computador da vítima e indispensável para detectar intrusos em um ataque por *stepping-stones*.

© 2017 Sociedade Brasileira de Computação

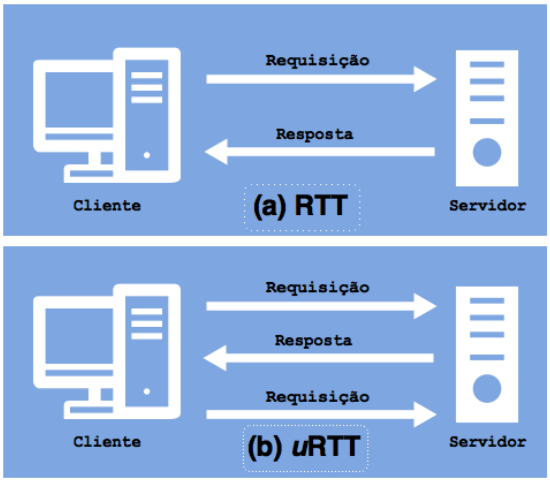


Figura 3. Distribuição RTT e *u*RTT.

3.2. Os Limites *Intra-gap* e *Inter-gap*

Para o presente estudo de detecção de intrusos por *stepping-stones*, é necessário classificar a ocorrência de ataques pelo tempo computado no *upload Round-Trip Time*, e categorizá-los em duas especificações distintas, denominadas por limites *intra-gap* e limites *inter-gap*.

A técnica proposta em [Ding and Huang 2011], permite especificar os intervalos de tempo, computados em microssegundos, para separar as informações irrelevantes dos pacotes de rede, ao qual não venham a contribuir para o algoritmo proposto.

O limite *intra-gap* está associado ao intervalo de tempo existente entre o pressionamento de teclas individuais de um comando, Unix ou Windows, e, portanto, responsáveis pela formação de um pacote de dados que será enviado como requisição para o computador da vítima.

O limite *inter-gap* consiste no espaço de tempo computado entre dois limites consecutivos. Conforme [Zhang 2014], o limite *inter-gap* consiste no intervalo de tempo entre o fim de uma sequência de caracteres determinando pela tecla *enter* e o primeiro caractere do próximo comando. Os limites *inter-gap* são importantes para estimar a existência das longas conexões, porque possui um relacionamento direto ao tamanho da cadeia de máquinas e, portanto, o *u*RTT os computadores envolvidos no ataque será progressivamente proporcional ao tamanho do número conexões definidas em uma longa cadeia de *stepping-stones*.

O exemplo apresentado na Figura 4 determina a definição de limites *intra-gap* e *inter-gap*, por meio da digitação dos comandos por um intruso. O cibercriminoso pressiona uma tecla e um pacote de requisição criptografado é criado como conteúdo para a vítima. O primeiro computador participante da cadeia *stepping-stones* recebe o pacote de requisição e, ao verificar o destinatário da mensagem, este reenvia o pacote para a próxima máquina até a informação ser recebida pela vítima. O computador da vítima, ao descriptografar o pacote recebido, infere que a informação é uma tecla inicial de um comando conhecido pelo sistema operacional, e, continua a processar os demais pacotes,

 © 2017 Sociedade Brasileira de Computação

até receber um pacote de definição para fim de comando, representado pela tecla enter. A máquina da vítima ao processar um comando, envia um pacote do tipo resposta para o computador antecessor ao ataque, e esta informação trafega o caminho inverso na cadeia de *stepping-stones*, até chegar ao computador do intruso.

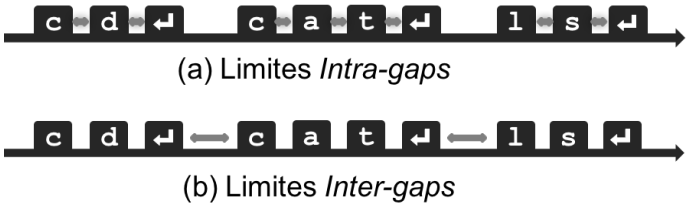


Figura 4. Os Limites *Inter-Gap* e *Intra-Gap*.

4. Algoritmo Proposto

O propósito do algoritmo é detectar a ocorrência de longas conexões a partir da observação dos dados trafegados no computador da vítima. O algoritmo, escrito na linguagem Python, adota uma ordem sequencial de passos, descritos formalmente, e combina propostas validadas em [Ding and Huang 2011, Zhang 2014], aos algoritmos matemáticos de [Gosset 1908, Pearson 1913], analisado detalhadamente abaixo.

O estágio é iniciado na captura do tráfego de rede, e para cada iteração de um experimento, o algoritmo leu os arquivos em *pcap* ou coletou os pacotes via SSH em tempo real e os classificou conforme duas categorias: pacote de requisição ou pacote de resposta. A determinação dos limites *inter-gaps* e *intra-gaps* foram calculados pelo valor de x_i, para $i = [1, ..., n]$, tal que i é o número de iterações de 1 até o n-ésimo valor da distribuição *u*RTT, ordenados em um vetor, crescentemente, pela função $sort$, tal que os valores de x_i são determinados pelo cálculo do intervalo de tempo entre primeiro t_1 e o segundo pacote de requisição t_2 na comunicação cliente e servidor, dada por

$$x_i = sort(t_2 - t_1) \tag{1}$$

onde, x_i é um vetor com valores i ordenados entre $[n, ..., 1]$.

A classificação de x_i é determinado pela exclusão dos limites *intra-gaps*, conforme as abordagens propostas em [Ding and Huang 2011, Zhang 2014]; a distinção entre os valores relacionados para o cálculo da limiar $\lambda(x)$, foram determinados pela seleção dos limites *inter-gaps* atuais x_i, combinado aos valores dos *inter-gaps* anteriores x_{i-1}, tal que $x_i \neq 0$ e $x_{i-1} > x_i$. O valor de n, determina o número atual, e é importante determinar o valor médio entre x_i e x_{i-1}, e pode ser representado por

$$\lambda(x) = \sum_{i=1}^{n} \frac{|n - x_i|}{x_{i-1}} \begin{cases} x_i \neq 0 \\ x_i < x_{i-1} \end{cases} \tag{2}$$

onde, n é número atual de valores x_i.

A distribuição mediana dos valores *inter-gaps* $D(\overline{X})$ é dado pelo cálculo da média $\overline{X}$, de um total de n valores contínuos de x_i, tal que o resultado do algoritmo possua um

© 2017 Sociedade Brasileira de Computação

valor influente acerca dos resultados discrepantes dos pacotes coletados. A equação 3 determina a distribuição da média *inter-gaps* $D(\overline{X})$, respectivamente,

$$D(\overline{X}) = \left(1 + \frac{x_{\overline{1}}^{n} + x_{\overline{2}}^{n}}{2}\right) \times \lambda(x), \tag{3}$$

sendo, x o valor atual de x_i e $\lambda(x)$ a limiar atual de x_i.

Os valores de confiança $XC_n(\lambda(x); D(\overline{X}))$ dos limites *inter-gaps* são resultados com valores padrões e aproximados ao valor exato na detecção de intrusos, e é dado por

$$XC_n(\lambda(x); D(\overline{X})) = \overline{X} \left(\sum_{i=1}^{n} \frac{|x_i \times D(\overline{X}) - x_{i-1}|}{n}\right) \begin{cases} x_i \neq 0 \\ x_i < x_{i-1} \end{cases} \tag{4}$$

O intervalo de confiança IC_f estabelece uma estimativa de valores prováveis na detecção de intrusos por *stepping-stone*, para um nível de confiança de 99% na distribuição *T-test* de [Gosset 1908], determinado por $\frac{\sigma}{\sqrt{n}}$ para erro da média $\mu(XC_n)$, dado por,

$$IC_f = \left(\mu(XC_n) - 2,5 * \frac{\sigma}{\sqrt{n}},\ \mu(XC_n) + 2,5 * \frac{\sigma}{\sqrt{n}}\right), \tag{5}$$

tal que, μ representa a média de XC_n; n o valor de $count(XC_n)$; $2,5$ a confiança em 99% na tabela de distribuição gaussiana (Z); e σ o desvio padrão de [Pearson 1913] dos valores XC_n.

Nas seções seguintes, são descritos o processo da configuração dos experimentos e coleta de dados, a análise dos resultados, e as considerações finais sobre os resultados.

5. Validação e Análise dos Resultados

A definição dos experimentos, aplicadas a três categorias de recursos distintos, em um ambiente de ataques por *stepping-stones* foi atribuída aos seguintes requisitos mínimos:

- a configuração de dois computadores para exercer as funções de cliente e servidor, e as demais máquinas reservadas para assumir o papel de *stepping-stone* no ataque;
- a interconectividade de todas as máquinas, uma a uma, entre si por conexões remotas por *SSH*;
- a execução do algoritmo em Python e o uso do analisador de protocolos *Wireshark* para captura do tráfego de pacotes na rede.

A primeira etapa do experimento consistiu da análise dos dados coletados na pesquisa em [Zhang 2014], para a investigação de um perfil de detecção. Para o desenvolvimento desta etapa, foram configuradas as seguintes máquinas: (i) um computador cliente, atribuído como atacante; (ii) um segundo como parte integrante da cadeia de *stepping-stones*; e (iii) uma terceira máquina, exercendo dupla função de servidor e vítima final nos ataques. A quarta máquina desempenhou a atribuição de *botnet* nos testes de invasão por *stepping-stones*. Enquanto às três primeiras máquinas estavam na rede do campus

 © 2017 Sociedade Brasileira de Computação

na Universidade de Houston, Texas, a quarta estava situada na cidade de Pittsburg, Pensilvânia.

A classificação dos dados, para a primeira etapa, sucedeu conforme a captura dos pacotes em cada uma das quatro máquinas da cadeia de *stepping-stones*. Os dados foram disponibilizados em planilhas do tipo CSV, e divididas entre longa e curtas conexões, contendo informações dos limites *inter-gaps*, em um total de quarenta experimentos.

A segunda etapa definiu a validação do algoritmo, ao investigar o comportamento do perfil de detecção definido na primeira etapa, por meio dos testes realizados em dados coletados da central de supercomputadores da Universidade de San Diego e disponibilizados pela . A base de dados do CAIDA[CAIDA 2016], dispõe de um tráfego de pacotes classificados entre dados coletados de usuários normais e pacotes capturados de conexões com ataques de negação de serviço, *botnets*, e exploração de vulnerabilidades.

A validação dos dados, para a base de dados definida por *Anonymized 2016 Internet Traces*, foi caracterizada pelo tráfego de dados passivo em tempo real com diferentes categorias de ataque. Os dados foram disponibilizados em formato *pcap*, classificados pela data da coleta e os arquivos continham entre 1 e $2,5\ GB$ de informações dos diferentes categorias de protocolo de rede.

A terceira etapa compreendeu da aplicação de um ataque no mundo real, por meio do uso de máquinas na nuvem da AWS. A elaboração de um modelo de ataques por *stepping-stones*, constitui de conexões formadas por cinco computadores e distribuídas da seguinte forma: (i) uma máquina cliente para o envio de ciberataques, localizada no Laboratório de Sistemas Distribuídos na UFCG; (ii) a configuração de três recursos da nuvem da AWS, localizadas respectivamente na Califórnia (EUA), Sidney (Austrália), e uma terceira em Singapura, atribuídas como parte integrante da cadeia de *stepping-stones*; e (iii) um servidor, atuando no papel de vítima, situado em Dublim, Irlanda.

A análise dos dados foi iniciada a partir do estabelecimento das conexões aos serviços da AWS. Um total de quarenta iterações de experimentos foram realizados, e para cada iteração, mil pacotes foram selecionados em tempo real, para definir um padrão dos dados, e os resultados foram salvos em arquivos de texto e em arquivos do tipo *pcap*, para fins de comprovação dos dados. A primeira etapa dos experimentos decorreu da análise dos dados coletados na pesquisa de [Zhang 2014].

Para às três etapas, houve uma necessidade de selecionar os pacotes relevantes, os que transportavam dados das conexões SSH, desconsiderando os demais.

O desenvolvimento do processo dos experimentos, consistiu em um modelo de fatores controláveis composto das informações de entrada, combinados aos fatores incontroláveis, determinados pelos limites *inter-gaps* discrepantes. O princípio básico do planejamento dos ensaios teve ênfase na possibilidade de replicação das iterações, sob as mesmas condições experimentais dos fatores aplicados no processo, conforme a ilustração da Figura 5.

Os fatores incontroláveis foram determinados pelos valores atípicos entre os quartis de um fluxo de informações para os limites *inter-gaps*. Os quartis representaram a separação dos limites *inter-gaps* divididos entre: (i) o quartil inferior Q_i que delimitou os 25% dos limites inter-gaps menores a Q_i e 75% dos limites *inter-gaps* maiores a Q_i; (ii)

© 2017 Sociedade Brasileira de Computação

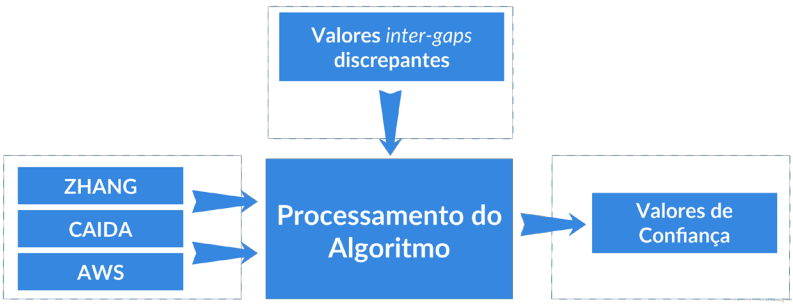


Figura 5. Planejamento dos Experimentos

o quartil central Q_c que correspondeu a 50% dos limites *inter-gaps* maiores e menores a Q_i e Q_s, respectivamente; (iii) e o quartil superior Q_s correspondente a 75% dos limites *inter-gaps* menores a Q_s e 25% dos limites *inter-gaps* maiores a Q_s.

A distribuição das informações para os valores discrepantes abrangeram os pontos extremos dos limites *inter-gaps* calculados em microssegundos (μs). Na primeira etapa dos experimentos, a frequência dos dados incomuns acima de Q_s foram contínuos em conexões estabelecidas por um intruso, e apresentaram uma variação nos quartis de $\Delta Q \cong 2,5\ \mu s$ valores atípicos por iteração. As curtas conexões ocasionalmente apresentaram dados discrepantes, retratando uma assiduidade entre um valor discrepante para cada três ensaios.

Para a segunda etapa, as distribuições dos fatores incontroláveis para amostras provenientes de ataques, foram contínuas, devido à presença de pontos discrepantes maiores que no quartil superior Q_s. A média individual para a presença de valores atípicos presentes na amostra foi de $\Delta Q \cong 18,33\ \mu s$. As amostras para arquivos provenientes de um tráfego de rede normal, apresentaram uma média de $\Delta Q \cong 1,33\ \mu s$ pontos discrepantes por rodada de experimento. A ausência de dados atípicos menores que o quartil inferior Q_i, para às duas etapas, determinou que os elementos discrepantes inferiores a Q_i não seriam considerados como fatores incontroláveis.

A variação dos valores discrepantes inferiores ΔQ_i e superiores ΔQ_s são apresentados pela Tabela 1, e detalham os resultados com os números máximos e mínimos dos valores de confiança, sobre às duas perspectivas de classificação das conexões.

Tabela 1. Resultado dos Valores Discrepantes em microssegundos (μs)

Base de Dados	Variação para Ataques	Variação para Usuários Normais
Zhang	$\Delta Q_i \cong 2,32$ e $\Delta Q_s \cong 7,84$	$\Delta Q_i \cong 0,36$ e $\Delta Q_s \cong 1,06$
CAIDA	$\Delta Q_i \cong 0,62$ e $\Delta Q_s \cong 2,01$	$\Delta Q_i \cong 0,003$ e $\Delta Q_s \cong 0,011$
AWS	$\Delta Q_i \cong 0,62$ e $\Delta Q_s \cong 7,84$	$\Delta Q_i \cong 0,43$ e $\Delta Q_s \cong 1,20$

Um ponto importante a ser destacado, na Tabela 1, foi referido ao número de valores discrepantes observados nos experimentos realizados na base de dados do CAIDA. O fator de detecção identificado nessa etapa, apresentou um número inferior, para às duas categorias de conexões, e divergiu dos resultados apresentados na base de dados de Zhang e AWS.

Os valores de confiança XC_n foram definidos como as variáveis de resposta do

 © 2017 Sociedade Brasileira de Computação

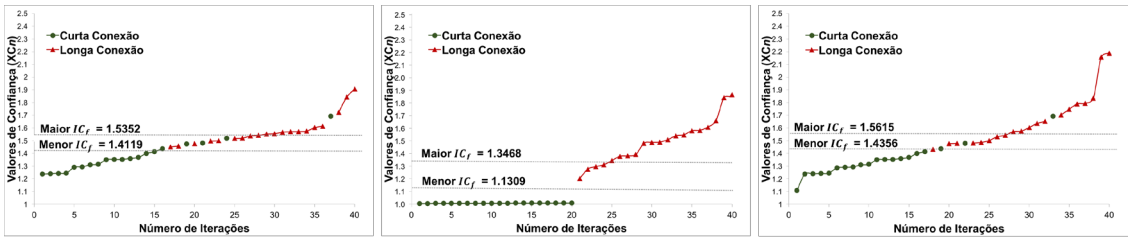
cálculo do risco relativo atribuído aos valores *inter-gaps* discrepantes. O propósito estabelecido aos valores de confiança XC_n foi determinar a estimativa de pontos aproximados de um verdadeiro valor, para definir um intervalo de confiança IC_f que delimite as informações entre ataque ou usuário real. O nível de confiança, para os resultados do algoritmo propostos neste trabalho, aplicados ao intervalo de confiança IC_f foi de 99% de confiança e uma incidência de erros em 1%. Na Tabela 2 é possível identificar os resultados para os valores mínimos e máximos do intervalo de confiança IC_f.

Tabela 2. Resultado dos Intervalo de Confiança

Base de Dados	Mínimo IC_f	Máximo IC_f
Zhang	1,4119 μs	1,5352 μs
CAIDA	1,1309 μs	1,3468 μs
AWS	1,4356 μs	1,5615 μs

Com base na Tabela 2, o intervalo de confiança para a base de dados CAIDA foi totalmente distinto da primeira e terceira etapa. Diante desse fato, foi possível inferir que a disposição anormal de valores discrepantes no segundo passo, pode indicar uma influência negativa nos resultados apresentados, e uma notável diferença na conclusão dos experimentos.

O intervalo de confiança IC_f foi determinado por meio da incidência de um risco relativo delimitado a partir dos valores de confiança XC_n. A delimitação do intervalo de confiança, a partir dos valores confiáveis, é ilustrada na Figura 6.



(a) Base de dados Zhang (b) Base de dados CAIDA (c) Base de dados AWS

Figura 6. Resultado do Intervalo de Confiança (IC_f) para as três etapas

Conforme apresentado na Figura 6, os resultados referentes a base de dados de Zhang, para curtas conexões foi de 95% e uma classificação de 19 verdadeiros positivos do 20 valores de confiança, e para longas conexões o resultado ficou com um índice de 100% de detecção, classificando os 20 verdadeiros negativos dos 20 valores de confiança XC_n. Os mesmos ensaios, aplicados a base de dados do CAIDA, viabilizou um índice de detecção de intrusos em 100% de acertos para às duas categorias de classificação entre as conexões. A aplicação do algoritmo a base de dados AWS, viabilizou a identificação de 19 dos 20 casos aplicados para longas conexões, e a detecção de 19 em 20 amostras, para conexões provenientes de um usuário real.

As métricas para avaliação de desempenho e mensuração dos resultados, foram calculadas com base na matriz de confusão, considerando as seguintes nomenclaturas: (i) curtas conexões positivas (CCP), como resultados provenientes de verdadeiros usuários;

© 2017 Sociedade Brasileira de Computação

(ii) as curtas conexões negativas (CCN), como valores de curtas conexões consideradas como ataque; (iii) as longas conexões positivas (LCP), considerados como conexões provenientes de intrusos; (iv) as longas conexões negativas (LCN), como valores provenientes de conexões estabelecidas por cibercriminosos que são classificadas como usuários reais; (v) a taxa de precisão (IPr), para determinar a taxa de acerto na classificação das conexões; (vi) a incidência de erros (IFa), calculada pela quantidade de alarmes falsos negativos produzidos pela classificação das longas conexões; e (vii) o índice de exatidão do algoritmo (IEt), determinado pelo número de longas e curtas conexões que foram corretamente classificadas. Na Tabela 3 é listado os resultados obtidos para cada etapa dos experimentos.

Tabela 3. Matriz de Confusão do resultado obtido para as três etapas

Base de Dados	CCP	CCN	LCP	LCN	IPr	IFa	IEt
Zhang	100%	0%	95%	5%	100%	5%	97,5%
CAIDA	100%	0%	100%	0%	100%	0%	100%
AWS	95%	5%	95%	5%	95%	5%	95%

Os resultados apresentados na Tabela 3, evidenciaram uma maior taxa de detecção, e, ao mesmo tempo, uma diminuição no índice de alerta falsos positivos. O índice de precisão dos resultados no trabalho proposto em [Zhang 2014] foi de 85% de detecção para longas conexões, em contrapartida, a aplicação do algoritmo a base de dados Zhang, nas mesmas condições de testes, apresentaram uma taxa de detecção de intrusos por *stepping-stones* em 97,5% na primeira etapa dos experimentos. Para a segunda, o processamento do algoritmo adotado para uma base de dados reconhecida internacionalmente no meio científico, retratou uma taxa de precisão de 100% para os casos testados. Na última etapa, o índice de precisão para um ambiente de máquinas configuradas em diferentes países foi de 95% de detecção de intrusos.

Em vista dos resultados obtidos neste trabalho, é possível evidenciar a superioridade do algoritmo proposto em relação às técnicas realizadas em [Ding and Huang 2011, Zhang 2014]. A redução na incidência de alarmes falso positivos comparados às técnicas anteriores indica a relevância do algoritmo na detecção de ataques do tipo *stepping-stones*.

6. Considerações Finais

O estudo acerca da detecção de intrusões por *stepping-stones* mostrou-se complexo desde as primeiras técnicas apresentadas em [Staniford-Chen and Heberlein 1995].

Na busca de atingir os objetivos direcionados na identificação de invasões por *stepping-stones*, o método de detecção proposto na presente pesquisa, aplicou o desenvolvimento de um algoritmo para classificar as conexões a partir da limitação dos limites *inter-gaps* aplicados a distribuição de informações com 99% de confiança. O diferencial da metodologia proposta em relação às técnicas existentes, está na identificação das tentativas de ataques a partir da análise dos dados coletados no computador da vítima, em tempo real de ataque, uma vez que os métodos anteriores desenvolveram soluções baseadas nas informações provenientes dos computadores intermediários envolvidos a uma cadeia de *stepping-stones*.

 © 2017 Sociedade Brasileira de Computação

A validação da solução proposta foi realizada por experimentos divididos em três etapas. A primeira etapa consistiu na aplicação do algoritmo aos dados coletados na pesquisa em [Zhang 2014], apresentando a proposta de criar um perfil de detecção a partir de informações já validadas e apresentou um índice de detecção de 97,5% para 5% de falsos positivos. O segundo passo, foi fundamentado na verificação do modelo aplicado a uma base de dados internacional reconhecida no meio científico e classificou as conexões a uma taxa de 100% de detecção para nenhum alerta falso positivo. A terceira etapa abrangeu uma incidência de 5% de falsos positivos e um índice de detecção em 95% para conexões configuradas em diferentes países.

Dentre as dificuldades encontradas, é destacado a alta concentração dos valores discrepantes ao longo da segunda fase dos experimentos e a influência negativa nos resultados para a base de dados do CAIDA. Outra dificuldade está relacionada na aplicação de um número reduzido de fatores incontroláveis produzidos pela ausência de métodos como *jitter* e *crossover* na organização dos experimentos.

É importante ressaltar que a finalidade estabelecida no atual trabalho é melhorar as soluções propostas na pesquisa de [Ding and Huang 2011, Zhang 2014] e contribuir cientificamente para a comunidade de pesquisadores na definição de novos conceitos e explicações a respeito da temática *stepping-stone*. Assim, as contribuições apontadas neste trabalho são evidenciadas na validação das pesquisas propostas por [Ding and Huang 2011, Zhang 2014], aplicadas ao algoritmo capaz de identificar intrusões por *stepping-stone* aplicados à base de dados para três arquiteturas de rede. Adicionalmente, a atribuição de diferentes recursos estatísticos, permite a aplicação do algoritmo proposto a qualquer ambiente que envolva ataques por *stepping-stones*.

Como trabalhos futuros, pretende-se incrementar uma solução que aplique as técnicas de *jitter*, *crossover* e perda de pacotes ao algoritmo proposto. Adicionalmente, incluir uma base de dados composta por informações pré-existentes dos padrões associados às longas e curtas conexões, viabilizando bloquear as intrusões por *stepping-stone* a partir da inferência comportamental das invasões. Por fim, implementar uma solução de processamento de *streams* ao algoritmo, para modelar um sistema de detecção de anomalias em tempo real.

7. Agradecimentos

Este trabalho foi parcialmente financiado pelo projeto EU-BRA SecureCloud (MCTI/RNP 3a Chamada Coordenada) e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Referências

CAIDA (2016). The CAIDA UCSD Anonymized Internet Traces 2016. `http://www.caida.org/data/passive/passive_2016_dataset.xml`.

Daud, A. Y., Ghazali, O., and Omar, M. N. (2015). Stepping-stone Detection Technique for Recognizing Legitimate and Attack Connections. In Jamaludin, Z., ChePa, N., Ishak, W. H. W., and Zaibon, S. B., editors, *5th International Conference on Computing and Informatics*, number 189, pages 440–446, Istanbul, Turkey. School of Com-

© 2017 Sociedade Brasileira de Computação

puting, University Utara Malaysia.

Ding, W. and Huang, S.-H. S. (2011). Detecting Intruders Using a Long Connection Chain to Connect to a Host. In *2011 IEEE International Conference on Advanced Information Networking and Applications*, pages 121–128, Biopoles, Singapure. IEEE.

Global, P. (2013). Na mira dos ataques cibernéticos - Pesquisa Global. Technical report, Ernest & Young, Rio de Janeiro, RJ.

Gosset, W. S. (1908). The Probable Error of a Mean. *Biometrika*, 6(1):1.

Herd, G. P. and Kriendler, J. (2013). *Understanding NATO in the 21st Century: Alliance Strategies, Security and Global Governance*. Contemporary Security Studies. Taylor & Francis, Abingdon, UK, 1 edition.

Huang, S.-H. S., Zhang, H., and Phay, M. (2016). Detecting Stepping-Stone Intruders by Identifying Crossover Packets in SSH Connections. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 1043–1050. IEEE.

Kumar, R. and Gupta, B. (2016). Neural Network Based Approach for Stepping Stone Detection under Delay and Chaff Perturbations. *Procedia Computer Science*, 85(Cms):155–165.

Kuo, Y.-W., Huang, S.-H. S., Ding, W., Kern, R., and Yang, J. (2010). Using Dynamic Programming Techniques to Detect Multi-hop Stepping-Stone Pairs in a Connection Chain. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 198–205. IEEE.

Mittal, R., Lam, V. T., Dukkipati, N., Blem, E., Wassel, H., Ghobadi, M., Vahdat, A., Wang, Y., Wetherall, D., and Zats, D. (2015). Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 537–550, New York, NY, USA. ACM.

Omar, M. N., Siregar, L., and Budiarto, R. (2008). Hybrid stepping stone detection method. In *2008 First International Conference on Distributed Framework and Applications*, pages 134–138. IEEE.

Pearson, K. (1913). On the Probable Error of a Coefficient of Correlation as found from a Fourfold Table. *Biometrika*, 9(1-2):22–27.

Staniford-Chen, S. and Heberlein, L. (1995). Holding intruders accountable on the Internet. In *Proceedings 1995 IEEE Symposium on Security and Privacy*, pages 39–49. IEEE Comput. Soc. Press.

Wu, H.-C. and Huang, S.-H. S. (2010). Neural networks-based detection of stepping-stone intrusion. *Expert Systems with Applications*, 37(2):1431–1437.

Zhang, H. (2014). *Detecting Network Intruders by Examining Packet Crossovers in Connections*. Dissertação (mestrado em ciências da computação), Dissertação (Mestrado em Ciências da Computação) – University of Houston, Texas.

Zhang, Y. and Paxson, V. (2000). Detecting stepping stones. In *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, SSYM'00, pages 13–13, Berkeley, CA, USA. USENIX Association.

 © 2017 Sociedade Brasileira de Computação

Analysis, Anti-Analysis, Anti-Anti-Analysis: An Overview of the Evasive Malware Scenario

Marcus Botacin[1], **Vitor Falcão da Rocha**[1], **Paulo Lício de Geus**[1], **André Grégio**[2]

[1]Instituto de Computação (IC)
Universidade Estadual de Campinas (Unicamp)
Campinas – SP – Brasil

[2]Departamento de Informática (DInf)
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

Abstract. *Malicious programs are persistent threats to computer systems, and their damages extend from financial losses to critical infrastructure attacks. Malware analysis aims to provide useful information to be used for forensic procedures and countermeasures development. To thwart that, attackers make use of anti-analysis techniques that prevent or difficult their malware from being analyzed. These techniques rely on instruction side-effects and that system's structure checks are inspection-aware. Thus, detecting evasion attempts is an important step of any successful investigative procedure. In this paper, we present a broad overview of what anti-analysis techniques are being used in malware and how they work, as well as their detection counterparts, i.e., the anti-anti-analysis techniques that may be used by forensic investigators to defeat evasive malware. We also evaluated over one hundred thousand samples in the search of the presence of anti-analysis technique and summarized the obtained information to present an evasion-aware malware threat scenario.*

1. Introduction

Malicious software, also known as malware, is a piece of software with malicious purposes. Malware actions can vary from data exfiltration to persistent monitoring, causing damages to both private and public institution, either on image or financial aspects. According to CERT statistics [Cert.br 2015], malware samples may account by more than 50% of total reported incidents.

Given this scenario, analysts are required to analyze malicious samples in order to provide either defensive procedures or mechanisms to prevent/mitigate the infection, as well as to perform forensic procedures on already compromised systems. The set of techniques used for such kind of inspection is known as malware analysis. Analysis procedures can be classified into static, where there is no need to run the code, and dynamic, where code runs on controlled environment [Sikorski and Honig 2012]. The scope of this work is limited to static procedures, since they are the first line of detection against evasive malware.

Considering malware analysis capabilities and peculiarities, criminals started to protect their artifacts from being analyzed, equipping them with so-called anti-analysis (or anti-forensics) techniques. This way, their infection could last longer since they could make their samples stealth. Recent studies, such as [Branco et al. 2012], present scenarios in which 50% of samples contain at least one anti-analysis technique, and this number has been growing constantly.

In order to keep systems protected from such new armored threats, we need to understand how these anti-analysis techniques work so as to develop ways to effectively detect evasive samples before

 © 2017 Sociedade Brasileira de Computação

they can act. This is called anti-anti-analysis. In this paper, we present the *modus operandi* behind such kind of techniques, as well as possible detection methods in details. We evaluated the developed solution against over a hundred thousand samples, benign and malicious, which allowed us to build an evasive scenario panorama. We also compared evasive techniques used on different contexts (distinct countries), which can help analysts to be ahead of the next coming threats.

This work is organized as follows: Section 2 introduces basic concepts related to anti-analysis techniques and discusses related work and tools aimed at detecting anti-analysis techniques; Section 3 describes an study of how distinct evasion techniques work, and presents our detection framework; Section 4 shows the results obtained from applying our solution to distinct datasets; finally, Section 5 presents concluding remarks and future work.

2. Background and Related Work

In this section, we present the concepts related to anti-analysis and their detection counterparts as well as introduce the current state-of-the-art solutions.

2.1. Anti-analysis

The main idea of anti-analysis techniques is to raise the bar of counteraction methods. It can be done in many ways, e.g., leveraging theoretical hard-to-compute constructions. In this Section, we provide an overview of such anti-analysis techniques.

One common approach is to fingerprint the analysis environment. Known analysis solutions expose regular patterns, such as fixed IP addresses, host names, and serial numbers. Evasive samples can detect those patterns and suspend their execution [Yokoyama et al. 2016]. This type of approach was successfully used against Cuckoo [Ferrand 2015] and Ether [Pék et al. 2011] sandboxes.

Another approach is to evade analysis by detecting execution side effects of virtual machines and emulators, which has been the most used environment for malware analysis. Those systems may exhibit a differing behavior when compared to their bare-metal counterparts, such as instructions not being atomic [Willems et al. 2012]. Currently, there are automated ways of detecting these side effects [Paleari et al. 2009]. Virtual Machines can also be detected by the changes that hypervisors perform on system internals (e.g., table relocations). Many tables, such as the Interrupt Descriptor Table (IDT), have their addresses changed on VMs when compared to bare-metal systems. These addresses can then be used as an indicator of a virtualized environment [Ferrie 2007].

There also approaches based not on evading the analysis itself, but on hardening the post-infection reverse engineering procedure. One noticeable technique is the anti-disassembly, a way of coding where junk data is inserted among legitimate code to fool the disassembler tool. Another variation of anti-disassembly techniques is to use opaque constants [Kruegel et al. 2007], constructions that cannot be solved without runtime information. Static attempts to guess resulting values of these expressions tend to lead to the path explosion problem [Xiao et al. 2010].

Finally, there are samples that make use of time measurement for analysis detection, since any monitoring technique imposes significant overheads [Lindorfer et al. 2011]. Although some solutions try to mitigate this problem by faking time measures, either on system APIs [Singh 2014], or on the hardware timestamp counter [Hexacorn 2014], the problem is unsolvable in practice, since an advanced attacker can make use of an external NTP server over encrypted connections.

 © 2017 Sociedade Brasileira de Computação

A notable example of anti-analysis tool is `pafish` [Pafish 2012], which consists of a series of modules that implement many of mentioned detection techniques, such as virtual machines detection and environment fingerprints. The tool's intention is to be used as verifier for any attempt of transparent solution, as well as to allow for a better understanding of practical malware evasion techniques.

2.2. Anti-Anti-Analysis

As well as the general analysis techniques, the anti-anti-analysis ones may also be classified as static or dynamic approaches. Static approaches can be applied in the form of pattern matching detectors of known anti-analysis constructions, for instance, address verification and locations. However, due to its known limitations, some constructions can only be solved during runtime, which is accomplished when they run inside dynamic environments.

Dynamic solutions, in a general way, are based on faking answers for known anti-analysis checks, such as in COBRA [Vasudevan and Yerraballi 2006]. These approaches, however, turn into an arms-race, since new anti-analysis techniques are often released and these systems need to be updated. To minimize the impact of this issue, transparent analysis systems have been proposed, such as Ether [Dinaburg et al. 2008] and MAVMM [Nguyen et al. 2009]. These systems, however, impose high overheads and development costs.

In the following sections, we review the anti-anti-analysis techniques for the above presented anti-analysis classes and present static detectors for these techniques. We left dynamic detectors for future work, since they are not part of this work's scope.

2.3. State-of-the-art of anti-anti-analysis

Our work is related to many detection solutions. Two noticeable ones are `pyew` [Pyew 2012] and `peframe` [Peframe 2014], which aim to detect the evasive technique itself, and not whether a tool/system/environment may be evaded or not. They work by statically looking for known shellcodes and library imports related to analysis evasion. In this work, we have expanded theses detectors in order to provider a broader coverage.

In addition to the aforementioned tools, our work relates to the one presented by [Branco et al. 2012], which implemented several anti-anti-analysis detectors and analyzed evasive samples. In this work, we have implemented both the anti-analysis techniques as well as the presented static detectors, applying them against our distinct datasets, and enriching their analysis with the discussion of the working flow of the mentioned techniques. We also proceed in the same way regarding the work by [Ferrie 2008].

At the time we were writing this article, we have noticed a related work implementing similar techniques [Oleg 2016]. Such work, however, is limited to implementation issues whereas we present a comprehensive discussion and results evaluation.

Other related approaches, although more complex, are those which rely on using intermediate representations (IR) [Smith et al. 2014] or interleaving instructions [Saleh et al. 2014], cases not covered by this work. This work also does not cover obfuscation techniques based on encryption. This issue was addressed by other work, such as [Calvet et al. 2012].

3. Anti-Analysis Techniques and Detection

In this section, we summarize the anti-analysis techniques, their operation, and how they can be detected. The techniques were originally described in the previously presented

© 2017 Sociedade Brasileira de Computação

works [Branco et al. 2012, Ferrie 2008, Pyew 2012, Peframe 2014, Pafish 2012, Oleg 2016] and are here classified according to their purpose: anti-disassembly, anti-debugging, and virtual machine detection. The complete discussion of each trick is presented on the appendix[1], due to space constraints.

3.1. Anti-disassembly

To understand how disassembly can turn into a hard task, we first introduce how current disassemblers work. After that, we present known tricks to detect evasion.

In general, disassemblers can be classified into `linear sweep` and `recursive traversal` approaches [Schwarz et al. 2002]. In the former, the disassembly process starts at the first byte of a given section and proceeds sequentially. The major limitation of this approach is that any data embedded in the code is interpreted as an instruction, leading to a wrong final disassembled code.

The latter approach takes into account the control flow of the program being disassembled, following the possible paths from the entry point, which solves part of problems presented by the linear approach, such as identifying `jmp`-preceeded data as code. The major assumption of this approach is that it is possible to identify all successors of a given branch, which is not always true, since any fail on identifying the instruction size can lead to incorrect paths and instructions.

3.1.1. Tricks

Table 1 shows a summary of anti-disassembly techniques and their detection methods[2].

Table 1. Anti-disassembly techniques and their detection methods.

Technique	Description	Detection
PUSH POP MATH	`PUSH` and `POP` a value on/from the stack instead of using a direct `MOV`	Detect a sequence of `PUSH` and `POP` on/from a register.
PUSH RET	`PUSH` a value on the stack and `RET` to it instead of the ordinary return.	Detect a sequence of `PUSH` and `RET`
LDR address resolving	Get loaded library directly from the PEB instead of using a function call	Check memory access referring the PEB offset.
Stealth API import	Manually resolving library imports instead of directly importing them.	Check for a sequence of access/compares of PEBs offsets.
NOP sequence	Breaks pattern matching by implanting NO-OPerations	Detect a sequence of `NOPs` within a given window
Fake Conditional	Create an always-taken branch	Check for branch-succeeded instructions which set branch flags
Control Flow	Changing control flow within an instruction block	Check for the `PUSH-RET` instruction sequence
Garbage Bytes	Hide data as instruction code	Check for branch-preceded data

[1] `https://github.com/marcusbotacin/Anti.Analysis/tree/master/Whitepaper` [2] Described by Branco et al. 2012

 © 2017 Sociedade Brasileira de Computação

3.2. Anti-Debug

In order to understand how anti-debug techniques work, we firstly introduce the basic idea of most tricks: using direct memory checks instead of function calls. Secondly, we present the tricks themselves.

3.2.1. Known API x Direct call

Most O.S. provide support for debugging checks. Windows, for instance, provides the `IsDebuggerPresent` API [Microsoft 2016]. Most anti-debug tricks, however, do not rely on these APIs, but perform direct calls instead. The main reason behind such decision is that APIs can be easily hooked by analysts, thus faking their responses. Internal structures, in turn, such as the process environment block (PEB) [Microsoft 2017c], are much harder to fake — some changes can even break system parts.

3.2.2. Tricks

Table 2 presents a summary of anti-debug techniques and their detection counterparts[3][4][5].

Table 2. Anti-debug techniques and their detection methods.

Technique	Description	Detection
Known Debug API	Call a debug-check API	Check for API imports
Debugger Fingerprint	Check the presence of known debugger strings	Check known strings inside the binary
NtGlobalFlag	Check for flags inside the PEB structure	Check for access on the PEB offset
IsDebuggerPresent	Check the debugger flag on the PEB structure	Check access to PEB on the debugger flag offset
Hook Detection	Verify whether a function entry point is a `JMP` instruction	Check for a `CMP` instruction having `JMP` opcode as an argument
Heap Flags	Check for heap flags on the PEB	check for heap checks involving PEB offsets
Hardware Breakpoint	Check whether hardware breakpoint registers are not empty	Check for access involving the debugger context
SS Register	Insert a check when interruptions are disabled	Check for SS register's `POP`s
Software Breakpoint	Check for the `INT3` instruction	Check for `CMP` with `INT3`
SizeOfImage	Change code image field	Check for PEB changes.

3.3. Anti-VM

A summary of anti-analysis tricks used by attackers to identify and evade virtualized environments is shown in Table 3[6][7].

[3] API check implemented by Pyew and Peframe [4] SizeOfImage implemented by Ferrie 2008 [5] Other techniques implemented by Branco et al. 2012 [6] VM fingerprint implemented by Pafish [7] Other techniques by Branco et al. 2012

© 2017 Sociedade Brasileira de Computação

Table 3. Anti-vm techniques and their detection methods.

Technique	Description	Detection
VM Fingerprint	Check for known strings, such as serial numbers	Check for known strings inside the binary
CPUID Check	Check CPU vendor	Check for known CPU vendor strings
Invalid Opcodes	Launch hypervisor-specific instructions	Check for specific instructions on the binary
System Table Checks	Compare IDT values	Look for checks involving IDT
HyperCall Detection	Platform specific feature	Look for specific instructions

3.4. Detection Framework

Given the presented detection mechanisms, we have implemented them by using a series of Python scripts[8]. They work by iterating over `libopcodes`-disassembled instructions, and performing a pattern matching on these, according the trick we are looking for. Our pipeline is able to provide the information whether a given technique was found on a binary or not, the number of occurrences per binary, and the section the trick was found.

Unlike Branco et al. 2012 approach, which considered the `RET` instruction as a code block delimiter, we have implemented a variable-size window delimiter to evaluate whether the tricks may have been implemented by making use of multi-block constructions.

4. Results

In this section, we present the results of applying our set of detectors to distinct datasets and discuss how anti-analysis tricks have been applied in practice.

4.1. Binary sections

Here we show the binary section influence on the trick detection. In order to perform this evaluation, we considered a dataset of 70 thousand worldwide crawled samples.

Figure 1 shows the detection distribution along the binary sections. It is worth to notice that the usual instruction section (`text`) is only the 5^{th} more prevalent section. The presence of other sections can be due to samples moving their tricks to distinct sections in order to not be detected by anti-virus (AV). This fact can only be exactly determined through dynamic analysis. The presence of some section such as `.aspack`, for instance, is due to the presence of a packer to obfuscate the code.

Figure 2 shows that the tricks contained in the `.text` section correspond to half of the total tricks detected. The most prevalent techniques, such as `PushPop` and `PushRet`, are the most simple ones.

4.2. Packer influence

In the last section, we could see that sections related to packer obfuscation were identified. In this section, we discuss the packer influence on trick detection. The first noticeable situation is that the tricks detected on packed samples are not equally distributed among sections, as shown in Figure 3. We can observe that the `C++` compiler and the `PIMP` packer exhibit tricks on the `.rsrc` section, whereas the `UPX` packer presents tricks on distinct sections. A similar situation

[8] `https://github.com/marcusbotacin/Anti.Analysis`

 © 2017 Sociedade Brasileira de Computação

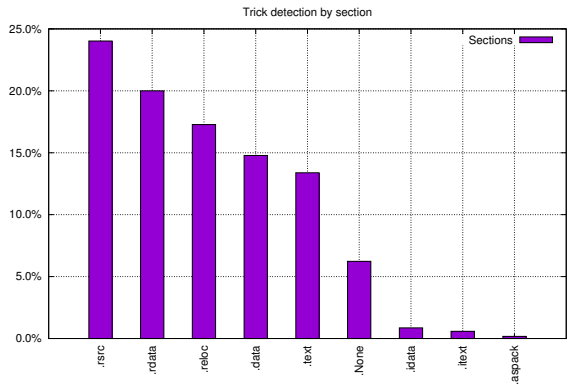


Figure 1. Tricks by section.

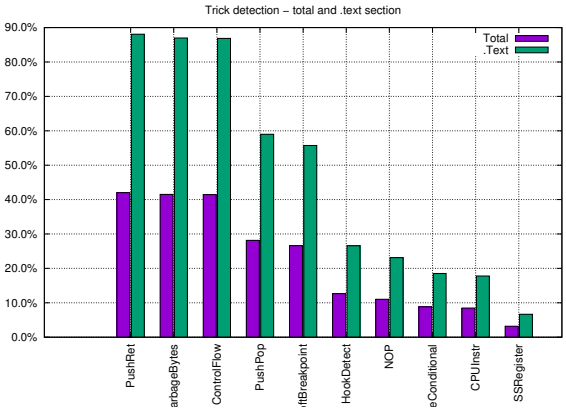


Figure 2. Tricks - total and `.text` section.

happens when considering the detected tricks, as shown in Figure 4. The `C++` compiler and the `PIMP` packer presents similar rates of tricks while the `UPX` packer presents distinct tricks. Finally, in order to evaluate the packer influence on trick detection, we unpacked all samples for which there are known unpackers (6 thousand samples), thus obtaining the results shown in Figure 5. We could confirm our expectations that the majority of the tricks are present on the packer, not on the original code. This fact is mostly due to the usage of malware kit generators.

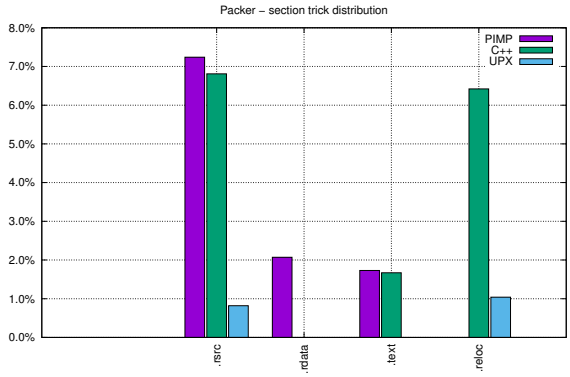


Figure 3. Packer distribution across binary sections.

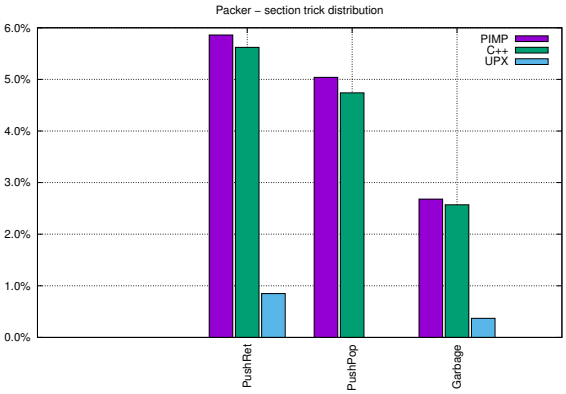


Figure 4. Tricks detected on distinct packers.

4.3. Malware and Goodware

Some of the presented tricks are widely used, so they can be found either on benign programs (goodware) and malicious ones. To verify whether the detection of those aforementioned tricks could be used as a malicious program indicator, we compared the trick incidence on both program classes, as shown in Figure 6. We performed our tests using as a benign dataset the binaries and DLLs from a clean Windows installation (binaries from the `System32` directory). We can observe that some general tricks (CPU identification) can also be found on system DLLs, but these are not present on the binaries. This fact is explained by the Windows architecture, which relies on DLLs for userland-kernel communication. This indicates that we need to employ distinct approaches when developing heuristics for executables and DLLs. We aim to extend this evaluation for general binaries, despite system ones, however, it is hard to ensure internet-downloaded binaries are not trojanized in any way, thus biasing the results.

© 2017 Sociedade Brasileira de Computação

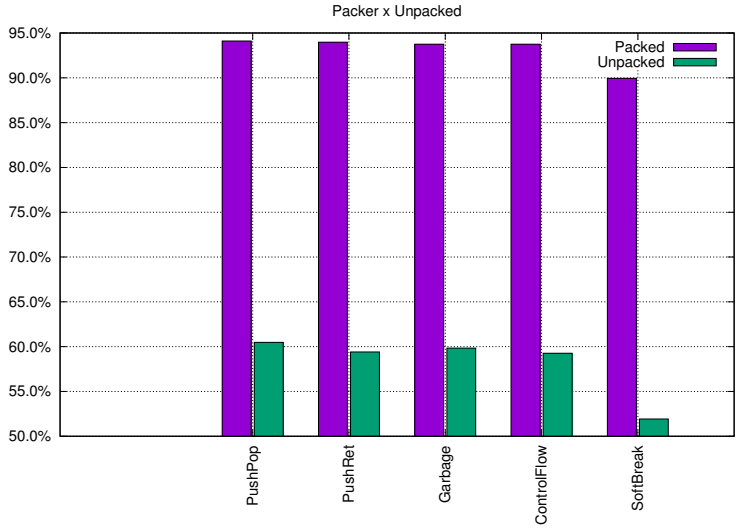


Figure 5. Packer influence on trick detection.

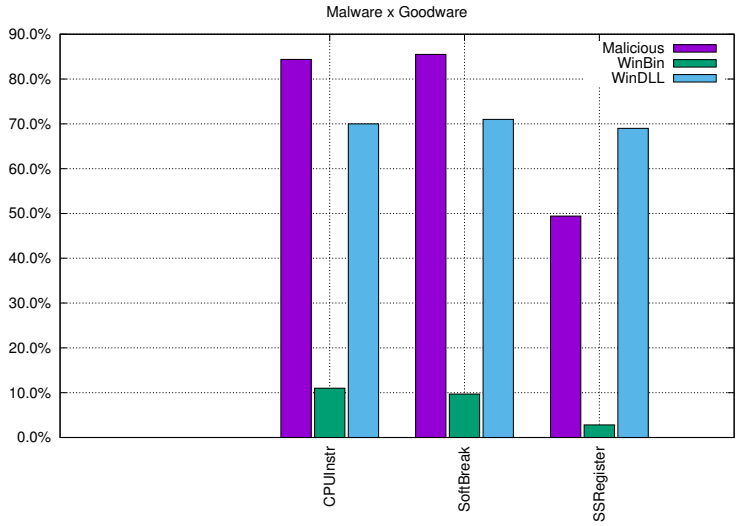


Figure 6. Tricks detection on malware and goodware.

4.4. Distinct Scenarios

The tricks prevalence differs across distinct datasets. In order to provide a view on how these differences affect user in practice, we compared the worldwide crawled dataset[9] to a dataset of 30 thousand Brazilian collected samples[10]. Figure 7 shows the results of comparing the datasets using the PEframe tool. We can observe that the Brazilian dataset presented higher detection rates for the `VmCheck` and the `VirtualBox` tricks and lower for the others. These rates are quite surprisingly, given the previous research results regarding the Brazilian scenario [Botacin et al. 2015]. When performing the same checks using our developed tricks, as shown on Figure 8, we show that the Brazilian scenario presents lower trick rates than the worldwide one. This differences can be explained by the fact that the knowledge behind the tricks detected by the PEframe are more spreaded, since they are simpler. More advanced tricks, such as some of those we have presented in this work,

[9] From `http://malshare.com/` [10] The same as in Botacin et al. 2015

 © 2017 Sociedade Brasileira de Computação

are only present on a broader scenario, i.e., the worldwide dataset.

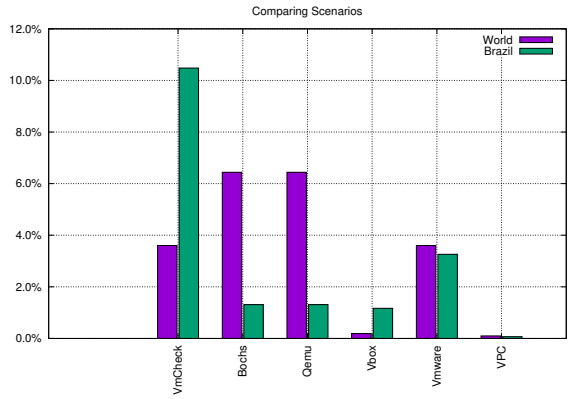


Figure 7. Comparing scenarios: PEframe detection.

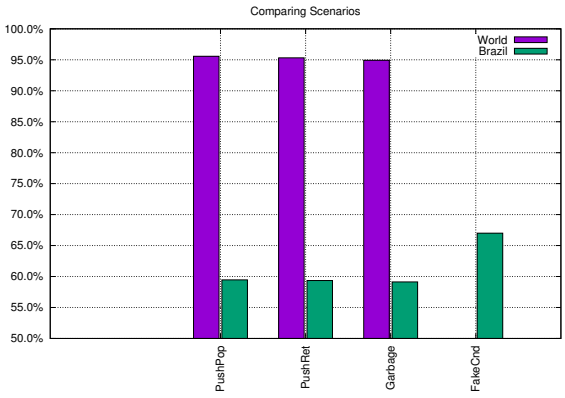


Figure 8. Comparing scenarios: Tricks detection.

4.5. Improving tricks and their detection

In this section, we present ways the tricks can be enhanced and how to detect them.

4.5.1. Trick splitting

A way of evading the trick detection is to split it across distinct blocks. Although we cannot check such usage in practice without dynamic analysis, we can look for signs of splitted-tricks by changing the detection window, as show on Figure 9. The initial value is the `RET` window, on which we traverse the block until this instruction is found. We considered the detection rate of this window as a ground-truth, thus presenting the `100%` detected value. The other values are fixed-size number of instructions which will be traversed, thus increasing the detection rate. We observed a maximum increase of `0.65%`.

4.5.2. Instruction dis-alignment

Another possible way of evading tricks detection is by using unaligned instructions, so the disassembler is not able to present the correct opcode. Although we could only check the effective usage of such approach on a dynamic system, we can look for static signs of such usage. In order to do so, we have implemented some detectors using YARA[11] rules and running them on the binary bytes. The tests results are shown in the Table 4. We have considered 300 random samples, being the `Aligned` considered as ground-truth. We can observe the `Unaligned` results are significantly higher, indicating it is a viable way of hidding code.

4.5.3. Compiler-based tricks

Another way of hidding the trick is to compile the code using instructions unsupported by AVs and other tools or indirect constructions. The *ROP itself malware* [Poulios et al. 2015], for instance,

[11] `https://virustotal.github.io/yara/`

© 2017 Sociedade Brasileira de Computação

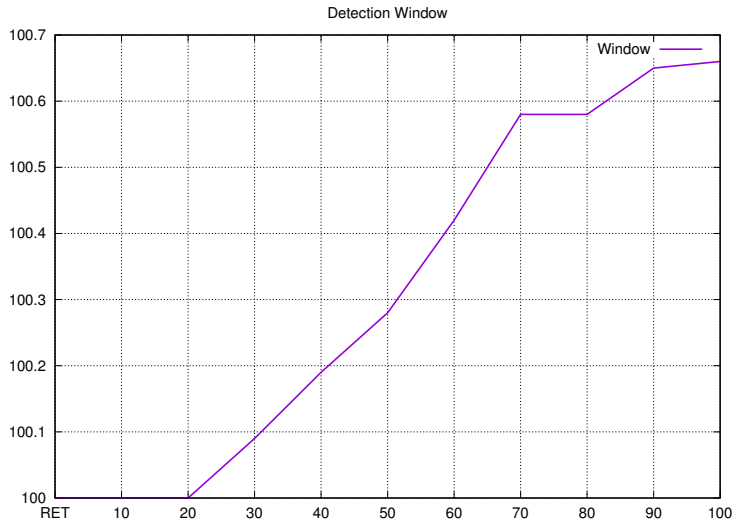


Figure 9. Evaluating block window effect on trick detection.

Table 4. Evaluating the occurrence of misaligned tricks.

Trick	Aligned	Unaligned
CPU	182	287
FakeJMP	63	203

suggested turning a malware sample into a ROP[12] payload, approach which was implemented by the *Ropinjector* tool [Poulios 2015]. The *SSexy tool* [Bremer 2012] compiles the code using `SSE`[13] instructions. The `Movfuscator` [domas 2015] does the same using `XOR` ones. Finally, the work [Barngert 2013] compiles a code to run using only `MMU` instructions[14]. In order to verify such approaches in practice, we submitted some known shellcodes from ExploitDB compiled using the ROPInjector solution, being the results reported in the Table 5. We can notice that the AV were not able to detect the payloads when compiled using the tool.

Table 5. Compilation-based evasion.

ShellCode	1^{15}	2^{16}	3^{17}	4^{18}	5^{19}
Unarmored	4/57	15/58	9/57	7/68	9/53
ROPInjector	0/57	0/57	0/54	0/54	0/53

4.6. General AV detection

The results from the previous section suggests that AV are not able to handle some tricks. Problems on AV emulators were also described on other work [Nasi 2014]. We submitted to VirusTotal some shellcodes armored with our tricks, being the results shown in the Table 6. We can notice that the AVs do not presented the same efficiency to handle armored and unarmored tricks.

[12] Return Oriented Programming [13] Streaming SIMD Extensions [14] Memory Management Unit

 © 2017 Sociedade Brasileira de Computação

Table 6. Evaluating AV Evasion: unarmored and trick-armored samples.

Shellcode	SC1		SC2		SC3	
Technique	W/o Trick	W/ Trick	W/o Trick	W/ Trick	W/o Trick	W/ Trick
Fakejmp	10/58	6/57	20/58	17/58	15/58	10/57
PushRet		7/57		17/58		10/58
NOP		6/57		17/57		10/58

4.7. Discussion

We have presented anti-analysis tricks and ways of detecting them. We also presented some insights on how they can be enhanced as well their detector. The major limitation relies on the fact that the effectiveness of such approach can only be measured using dynamic analysis, which is a straightforward future work. Additionally, we aim to implement some kind of rule-based remediation [Lee et al. 2013]. More details about this solution's limitations are presented in the appendix.

5. Conclusion

In this work, we have studied anti-analysis techniques, their effect on malware analysis, and theoretical limitations. We also developed static detectors able to identify known evasive constructions on binaries. We have tested these detectors against multiple datasets and observed that there are significant differences between the Brazilian scenario compared to the global one.

The list of tricks presented in this paper is not exhaustive, since attackers keep testing, and consequently developing new ways of evading analysis and detecting environments. Hence, our future work consists on implementing new detectors, as they have been discovered, as well as evaluating distinct datasets, in order to identify other trends about malware.

Acknowledgements

This work was supported by the Brazilian National Counsel of Technological and Scientific Development (CNPq, Universal 14/2014, process 444487/2014-0) and the Coordination for the Improvement of Higher Education Personnel (CAPES, Project FORTE, Forensics Sciences Program 24/2014, process 23038.007604/2014-69).

References

Bachaalany, E. (2005). Detect if your program is running inside a virtual machine. `https://www.codeproject.com/articles/9823/detect-if-your-program-is-running-inside-a-virtual`.

Barngert, J. (2013). Trapcc. `https://github.com/jbangert/trapcc`.

Botacin, M., Grégio, A., and de Geus, P. (2015). Uma visão geral do malware ativo no espaço nacional da internet entre 2012 e 2015. *Anais do XV SBSEG*.

Branco, R. R., Barbosa, G. N., and Neto, P. D. (2012). Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti- vm technologies. `https://github.com/rrbranco/blackhat2012/blob/master/blackhat2012-paper.pdf`.

Bremer, J. (2012). Ssexy. `https://github.com/jbremer/ssexy`.

© 2017 Sociedade Brasileira de Computação

Calvet, J., Fernandez, J. M., and Marion, J.-Y. (2012). Aligot: Cryptographic function identification in obfuscated binary programs. In *Proc. of the 2012 ACM Conf. on Computer and Communications Security*, CCS '12, pages 169–182, New York, NY, USA. ACM.

Cert.br (2015). Estatísticas dos incidentes reportados ao cert.br. https://www.cert.br/stats/incidentes/. Access Date: May/2017.

Dinaburg, A., Royal, P., Sharif, M., and Lee, W. (2008). Ether: Malware analysis via hardware virtualization extensions. In *Proc. of the 15th ACM Conf. on Computer and Communications Security*, CCS '08, pages 51–62, New York, NY, USA. ACM.

domas (2015). Movfuscator. https://github.com/xoreaxeaxeax/movfuscator.

Eliam, E. (2005). *Reverse: Secrets of Reverse Engineering*. Willey.

Ferrand, O. (2015). How to detect the cuckoo sandbox and to strengthen it? *Journal of Computer Virology and Hacking Techniques*, 11(1):51–58.

Ferrie, P. (2007). Attacks on virtual machine emulators. https://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf.

Ferrie, P. (2008). Anti-unpacker tricks. http://pferrie.tripod.com/papers/unpackers.pdf.

hexacorn (2014). Protecting vmware from cpuid hypervisor detection. http://www.hexacorn.com/blog/2014/08/25/protecting-vmware-from-cpuid-hypervisor-detection/.

Hexacorn (2014). Rdtscp - a recooked antire trick. http://www.hexacorn.com/blog/2014/06/15/rdtscp-a-recooked-antire-trick/.

Kruegel, C., Kirda, E., and Moser, A. (2007). Limits of Static Analysis for Malware Detection. In *Proc. of the 23rd Annual Computer Security Applications Conference (ACSAC) 2007*.

Laboratory, B. (2012). Detecting vmware. https://brundlelab.wordpress.com/2012/10/21/detecting-vmware/.

Lee, J., Kang, B., and Im, E. G. (2013). Rule-based anti-anti-debugging system. In *Proc. of the 2013 Research in Adaptive and Convergent Systems*, RACS '13, pages 353–354, New York, NY, USA. ACM.

Lindorfer, M., Kolbitsch, C., and Milani Comparetti, P. (2011). Detecting environment-sensitive malware. In *Proc. of the 14th International Conf. on Recent Advances in Intrusion Detection*, RAID'11, pages 338–357, Berlin, Heidelberg. Springer-Verlag.

Liu, Q. (2012). Just another malware analyzer. https://github.com/lqhl/just-another-malware-analyzer.

Lyashko, A. (2011). Stealth import of windows api. http://syprog.blogspot.com.br/2011/10/stealth-import-of-windows-api.html.

Microsoft (2016). Isdebuggerpresent. https://msdn.microsoft.com/pt-br/library/windows/desktop/ms680345%28v=vs.85%29.aspx.

Microsoft (2017a). Download detours express. https://www.microsoft.com/en-us/download/details.aspx?id=52586.

Microsoft (2017b). Getprocessheap function. https://msdn.microsoft.com/pt-br/library/windows/desktop/aa366569%28v=vs.85%29.aspx.

Microsoft (2017c). Peb structure. https://msdn.microsoft.com/pt-br/library/windows/desktop/aa813706%28v=vs.85%29.aspx.

 © 2017 Sociedade Brasileira de Computação

Microsoft (2017d). Peb_ldr_data structure. https://msdn.microsoft.com/pt-br/library/windows/desktop/aa813708%28v=vs.85%29.aspx.

Microsoft (2017e). Setwindowshookex function. https://msdn.microsoft.com/en-us/library/windows/desktop/ms644990%28v=vs.85%29.aspx.

Microsoft (2017f). Virtualquery function. https://msdn.microsoft.com/pt-br/library/windows/desktop/aa366902%28v=vs.85%29.aspx.

MNIN.org (2006). The torpig/anserin/sinowal family of trojans detect virtual machine information and abort infection when present. https://www.mnin.org/write/2006_torpigsigs.pdf.

Nasi, E. (2014). Bypass antivirus dynamic analysis. http://packetstorm.foofus.com/papers/virus/BypassAVDynamics.pdf.

Nguyen, A. M., Schear, N., Jung, H., Godiyal, A., King, S. T., and Nguyen, H. D. (2009). Mavmm: Lightweight and purpose built vmm for malware analysis. In *Computer Security Applications Conf., 2009. ACSAC '09. Annual*, pages 441–450.

Oleg, K. (2016). Anti-debug protection techniques: Implementation and neutralization. https://www.codeproject.com/articles/1090943/anti-debug-protection-techniques-implementation-an.

Pafish (2012). Pafish. https://github.com/a0rtega/pafish.

Paleari, R., Martignoni, L., Roglia, G. F., and Bruschi, D. (2009). A fistful of red-pills: How to automatically generate procedures to detect cpu emulators. In *Proc. of the 3rd USENIX Conf. on Offensive Technologies*, WOOT'09, pages 2–2, Berkeley, CA, USA. USENIX Association.

Peframe (2014). Peframe. https://github.com/guelfoweb/peframe.

Pék, G., Bencsáth, B., and Buttyán, L. (2011). nether: In-guest detection of out-of-the-guest malware analyzers. In *Proc. of the Fourth European Workshop on System Security*, EUROSEC '11, pages 3:1–3:6, New York, NY, USA. ACM.

Poulios, G. (2015). Ropinjector. https://github.com/gpoulios/ROPInjector.

Poulios, G., Ntantogian, C., and Xenakis, C. (2015). Ropinjector: Using return oriented programming for polymorphism and antivirus evasion. https://www.blackhat.com/docs/us-15/materials/us-15-Xenakis-ROPInjector-Using-Return-pdf.

Pyew (2012). Pyew. https://github.com/joxeankoret/pyew.

Radare (2008). Radare. https://github.com/radare/radare/blob/master/src/kradare/dtdumper/dtdumper.c. Access Date: May/2017.

Saleh, M., Ratazzi, E. P., and Xu, S. (2014). Instructions-based detection of sophisticated obfuscation and packing. In *2014 IEEE Military Communications Conf.*, pages 1–6.

Schwarz, B., Debray, S., and Andrews, G. (2002). Disassembly of executable code revisited. In *Proc. of the Ninth Working Conf. on Reverse Engineering (WCRE'02)*, WCRE '02, pages 45–, Washington, DC, USA. IEEE Computer Society.

Securiteam (2004). Red pill... or how to detect vmm using (almost) one cpu instruction. http://www.securiteam.com/securityreviews/6Z00H20BQS.html.

Sikorski, M. and Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition.

Singh, A. (2014). Not so fast my friend - using inverted timing attacks to bypass dynamic analysis. http://labs.lastline.com/not-so-fast-my-friend-using-inverted-timing-attacks-to-bypass-dynamic-analysis.

© 2017 Sociedade Brasileira de Computação

Smith, A. J., Mills, R. F., Bryant, A. R., Peterson, G. L., and Grimaila, M. R. (2014). Redir: Automated static detection of obfuscated anti-debugging techniques. In *2014 International Conf. on Collaboration Technologies and Systems (CTS)*, pages 173–180.

Vasudevan, A. and Yerraballi, R. (2006). Cobra: fine-grained malware analysis using stealth localized-executions. In *2006 IEEE Symposium on Security and Privacy (S P'06)*, pages 15 pp.–279.

VMWare (2010). Mechanisms to determine if software is running in a vmware virtual machine (1009458). `https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1009458`.

Willems, C., Hund, R., Fobian, A., Felsch, D., Holz, T., and Vasudevan, A. (2012). Down to the bare metal: Using processor features for binary analysis. In *Proc. of the 28th Annual Computer Security Applications Conf.*, ACSAC '12, pages 189–198, New York, NY, USA. ACM.

Xiao, X., s. Zhang, X., and d. Li, X. (2010). New approach to path explosion problem of symbolic execution. In *Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conf. on*, pages 301–304.

Yokoyama, A., Ishii, K., Tanabe, R., Papa, Y., Yoshioka, K., Matsumoto, T., Kasama, T., Inoue, D., Brengel, M., Backes, M., and Rossow, C. (2016). *SandPrint: Fingerprinting Malware Sandboxes to Provide Intelligence for Sandbox Evasion*, pages 165–187. Springer International Publishing, Cham.

 © 2017 Sociedade Brasileira de Computação

Detecção de *malware* metamórfico baseada na indexação de grafos de dependência de dados

Luis Rojas Aguilera[1], **Eduardo Souto**[1], **Gilbert Breves Martins**[2]

[1]Instituto de Computação (ICOMP) – Universidade Federal do Amazonas (UFAM)
Av. Rodrigo Otávio, 6200 - Japiim, Manaus, AM, Brasil

[2]Campus Manaus Distrito Industrial (CMDI) – Instituto Federal do Amazonas (IFAM)
Av. Danilo de Matos Areosa, 1672 - Distrito Industrial, Manaus, AM, Brasil

{rojas,esouto}@icomp.ufam.edu.br,gilbert.martins@ifam.edu.br

Abstract. *Metamorphism and code mutation have been used successfully by malware writers to generate obfuscated codes without altering the original features, making them more difficult to detect. This work presents an approach for identifying metamorphic malware through extraction of characteristics on Data Dependency Graphs, to construct a classification index that is able to quickly and accurately recognize if a certain suspicious code belongs to a family of malware. Experimental results on 3045 metamorphic virus samples show higher average accuracy rates than most commercial antiviruses.*

Resumo. *O metamorfismo e a mutação de código têm sido utilizados com sucesso pelos criadores de malware para gerar códigos obfuscados sem alterar as funcionalidades originais, tornando-os mais difíceis de detectar. Este trabalho apresenta uma abordagem para a identificação de malware metamórfico através extração de características a partir de Grafos de Dependência de Dados, para a construção de um índice de classificação que seja capaz de reconhecer de forma rápida e precisa se um determinado código suspeito pertence à uma família de malware. Os resultados experimentais sobre 3045 amostras de vírus metamórficos apresentam taxas médias de acurácia superiores a maioria dos antivírus comerciais.*

Introdução

Apesar dos crescentes investimentos e esforços realizados pelas companhias de segurança na criação de soluções de detecção de software maliciosos (e.g. antivirus, antispyware e adware), a infecção de *malware* continua sendo uma das principais ameaças a segurança dos sistemas computacionais no mundo. A sofisticação dos ataques e o aumento cada vez maior das famílias de *malware* tem tornado a defesa contra cibercriminosos uma tarefa ainda mais difícil. De acordo com a empresa AV-Test [AV-Test 2015], considerando somente os anos de 2013 e 2014, a quantidade de novas amostras de *malware* aumentou de 83 para 142 milhões, representando uma taxa de crescimento superior à 71%. Um relatório produzido pela Symantec descreve um aumento de 274 milhões de amostras de vírus em 2014 para 357 milhões em 2016 [Symantec 2017]. Esses estudos reconhecem a falta de capacidade das ferramentas de detecção de *malware* existentes em lidar com as técnicas usadas pelos atacantes para evadir tais ferramentas.

 © 2017 Sociedade Brasileira de Computação

Essa dificuldade ocorre principalmente porque os criadores de programas maliciosos empregam várias técnicas de evasão como polimorfismo e metamorfismo de código para criar novas variantes de um *malware* a partir de uma mesma instância de código original, gerando diversas estruturas e padrões de código aleatórios [Martins et al. 2016]. Tal nível de variedade pode ser obtido de forma automatizada, a uma taxa exponencial, o que torna a criação de modelos de identificação de *malware* um processo ainda mais difícil.

Para lidar com os desafios colocados pelos *malware* metamórficos, novas abordagens para analisar a semântica e o comportamento de programas suspeitos tem sido propostas. Algumas dessas abordagens incluem: $a)$ modelagem estatística dos padrões de código gerados pelos motores metamórficos [Lin and Stamp 2010] [Singh et al. 2015] [Kuriakose and Vinod 2014]; $b)$ análise das distribuições das ocorrências de sequências de *opcodes* de instruções [Rad et al. 2012]; $c)$ análise estatística composta pela combinação de métodos de ranqueamento de características de grupos de *opcodes* de instruções [Kuriakose and Vinod 2015]; e $d)$ análise de representações intermediárias que expressam as semânticas do código, tais como: grafos de controle de fluxo [Hu et al. 2009] [Alam et al. 2015a], grafos de chamadas a APIs do sistema [Ahmadi et al. 2013] e grafos de dependência de dados [Kim and Moon 2010] [Martins et al. 2016].

Dentre as abordagens citadas, o uso de Grafos de Dependência de Dados (GDDs) se apresenta como uma alternativa promissora para detecção de *malware* metamórficos, visto que as relações semânticas representadas pelos GDDs se mantêm praticamente inalteradas mesmo quando uma nova versão de *malware* é criada. Um GDD é uma representação que usa uma notação baseada em grafos para descrever todas as relações de dependências de dados existentes entre as instruções de um programa de computador. Liu et al. [2006] e Kim e Moon [2010], por exemplo, demonstraram que é possível usar GDDs para identificar efetivamente pelo menos cinco tipos diferentes de técnicas de ofuscação de código: *i)* alteração de formato; *ii)* mudança no nome das variáveis; *iii)* reordenamento de estruturas; *iv)* troca de estruturas de controle; e *v)* inserção de instruções inócuas (ou código lixo).

Entretanto, o uso de grafos para identificação de códigos maliciosos requer um processo de comparação capaz de diferenciar grafos gerados a partir de programas benignos, daqueles gerados a partir de *malware* previamente identificados. Devido ao processo de metamorfismo de código, o casamento (*matching*) de grafos deve ser executado com foco na identificação do nível de similaridade entre subgrafos (máximo isomorfismo de subgrafo), que é reconhecidamente um processo de alta complexidade computacional [Garey and Johnson 1990].

Neste contexto, este trabalho propõe uma abordagem para a detecção de códigos maliciosos metamórficos baseado na indexação de grafos de dependência de dados de um programa. Cada *malware* é representado por um conjunto de grafos de dependência de dados. Os nós dos grafos são rotulados com base na semântica das instruções obtidas do código assembly do *malware*. Modelos de classificação são gerados para simplificar o processo de comparação de grafos, usando somente as características estruturais dos grafos. Tais modelos são utilizados como estruturas na forma de índices que servem para detectar novas instâncias baseado na comparação com os padrões presentes em instâncias conhecidas. Os resultados experimentais sobre 3045 amostras de vírus metamórficos

 © 2017 Sociedade Brasileira de Computação

apresentam taxas médias de acurácia superiores a maioria dos antivírus comerciais.

O restante deste documento está organizado como segue: A Seção 2 descreve alguns trabalhos que empregam abordagens baseadas em grafos para lidar com o problema da identificação de *malware* metamórfico. A Seção 3 introduz os aspectos técnicos envolvidos no reconhecimento de códigos maliciosos ofuscados e conceitos básicos à respeito de grafos de dependência de dados. A Seção 4 descreve alguns resultados experimentais incluindo a comparação da abordagem proposta com soluções comerciais. Por fim, a Seção 5 apresenta as considerações finais e propostas de trabalhos futuro.

Trabalhos Relacionados

Em função de sua natureza não trivial, diversas abordagens foram propostas para tratar o problema de identificação de *malware* metamórficos. Muitas dessas abordagens são baseadas no uso de modelos de detecção baseados em grafos.

Xin Hu et al. [Hu et al. 2009] propõe um sistema de gerenciamento de bases de dados de *malware* denominado de SMIT (*Symantec Malware Indexing Tree*), cuja detecção é baseada na comparação de grafos de chamadas de funções de *malwares* conhecidos. Cada instância de *malware* é representada por um grafo, que é consultado no momento da tentativa de identificação, através de uma busca baseada na regra do vizinho mais próximo (*K-Nearest Neighbor - KNN*) [Paredes and Chávez 2005]. Para diminuir o tempo de execução das consultas, Xin Hu et al. empregam um método que calcula a similaridade dos grafos usando características estruturais à nível de instruções e um mecanhismo de indexação de resolução múltipla baseado na utilização de vetores de características adaptáveis.

Kim e Moon [Kim and Moon 2010] propõem a utilização de grafos de dependências de dados, como base de comparação, pois as relações de dependência costumam ser resistentes ao processo de metamorfismo. Para determinar se uma instância é uma variante metamórfica, o processo de comparação é tratado como a solução do isomorfismo máximo de subgrafos [Raymond and Willett 2002]. Heurísticas baseadas na utilização de algoritmos genéticos são utilizadas para diminuir o tempo de comparação. Apesar de efetiva, esta abordagem deixa de explorar outras características relevantes que poderiam enriquecer o modelo de detecção. [Martins et al. 2016] explorou esta limitação, introduzindo a classificação de nós baseado na semântica e função das instruções correspondentes, dividindo estes nós em três grupos: a)carga, para nós que iniciam uma cadeia de dependência, b)processamento, onde os conteúdos das variáveis são transformados e c)decisão, onde o fluxo do programa muda baseado em uma condição. Os resultados desse trabalho mostram uma diminuição tanto no tamanho dos grafos manipulados, como na variância dos resultados obtidos no processo de comparação.

Eskandari and Hashemi [Eskandari and Hashemi 2012] estudaram os padrões semânticos em códigos executáveis de instâncias metamórficas e propõem a utilização de grafos de controle de fluxo que modelem as sequências de chamadas de APIs do sistema. Com base nestes grafos é construído um vetor de características composto de pares ordenados de arestas e nós contendo as chamadas. Tais vetores são utilizados no treinamento e teste de diferentes classificadores como árvores de decisão, floresta de árvores, *Naive Bayes*, entre outros. Resultados experimentais monstram um melhor desempenho dos modelos criados utilizando florestas de árvores.

© 2017 Sociedade Brasileira de Computação

Alam et.al [Alam et al. 2015b] apresentam uma estrutura chamada de Grafo de Controle de Fluxo Anotado (GCFA), construído a partir das funções do código binário de instâncias de *malware*. Operações comumente utilizadas em código assembly são mapeadas a classes que são utilizadas para anotar os GCFAs. Uma instância é rotulada como *malware* se os padrões coincidem com os presentes numa base de referencia de GCFAs de *malware*s conhecidos.Resultados experimentais apresentam uma taxa de detecção do 98.9% e 4.5% de falsos positivos.

Apesar de todos os trabalhos apresentados nesta seção usarem grafos, cada uma das abordagens propostas usam grafos para modelar características distintas (chamada de funções, relações de dependência entre instruções, entre outras), algumas com maior ou menor carga semântica (i.e representatividade do funcionamento e finalidade do codigo original). Além disso, as estratégias de recuperação da informação e comparação variam desde o uso de heurísticas até esquemas de indexação e modelos de classificação para diminuir o tempo de processamento, dada a complexidade inerente ao processo de comparação de grafos.

Por meio da combinação das características mais interessantes dessas abordagens, este trabalho apresenta uma metodologia de identificação de *malware* metamórfico através extração de características baseadas em Grafos de Dependência de Dados, para a construção de um índice de classificação que seja capaz de reconhecer de forma rápida e precisa se um determinado código suspeito pertence à uma família de *malware*.

Detecção de Malware Metamórfico usando GDD

Antes de apresentar a abordagem proposta, esta seção inicia definindo os principais conceitos utilizados em sua construção.

Um Grafo de Fluxo de Controle (GFC) é uma representação que usa notação de grafo para descrever todos os caminhos que podem ser executados por um programa de computador. Em tal grafo, cada nó representa um bloco básico de instruções, isto é, uma região de código sequencial sem qualquer salto de execução. Dessa forma, o destino de um salto denota o começo de um bloco, ou seja, qualquer salto termina em um bloco. Arestas direcionadas são usadas para representar tais saltos na estrutura de controle. Ainda há dois blocos especiais, o bloco de entrada e o bloco de saída, de onde se começa e termina o fluxo, respectivamente. Formalmente um grafo de controle de fluxo pode ser definido como [Alam et al. 2015b]:

Definição 1 *Um grafo dirigido $G = (V, E)$, onde $V = V(G)$ é o conjunto de vértices do grafo que representa os blocos básicos em um programa e $E = E(G)$ é o conjunto de arestas do grafo que representa as funções de chamadas entre blocos básicos de um programa.*

Um Grafo de Dependência de Dados (GDD) é um grafo dirigido que representa as dependências entre partes de um código fonte (instruções) [Ferrante et al. 1987] baseado na utilização dos espaços de memória e o fluxo dos dados. Neste trabalho, os grafos de dependência são construidos a partir das relações existentes entre as instruções no código assembly de um binário, onde cada instrução é representada por meio de um vértice e as relações são expressadas por meio de arestas direcionadas denominadas de arestas de dependência. Para que o sentido da aresta indique de quais outros vértices, um vértice

 © 2017 Sociedade Brasileira de Computação

em particular depende, este trabalho adota uma definição para arestas de dependência adaptada daquela apresentada em Kim e Moon [Kim and Moon 2010]:

Definição 2 *Sejam $v_i \in V$ e $v_j \in V$, onde $G = (V, E)$ é um GDD. Caso exista ao menos uma variável x tal que x é usada em v_j e o seu valor é estabelecido em v_i, então existe uma aresta de dependência $e_x \in E$ saindo de v_j e chegando em v_i.*

Visão Geral da Abordagem Proposta

A abordagem proposta para detecção de *malware* metamórfico é baseada na comparação de GDDs de programas binários. Para construir os GDDs a partir de um código binário é necessário submetê-los a um processo de engenharia reversa. Neste processo, o código é transformado de linguagem de máquina para a linguagem assembly (*disassembler*), transcrevendo as instruções enviadas ao processador para os seus mnemônicos em assembly (*asm*). Em seguida, o código assembly é usado para identificar todas as funções do programa. Por razões de desempenho, a abordagem cria um GFC para cada função e um GDD para cada GFC criado.

Para simplificar o processo de *matching* de grafos, um vetor baseado nas características estruturais de cada GDD é construido. Tais vetores são usados na construção de modelos que identificam os padrões presentes em *malware*s conhecidos. Na etapa de análise e identificação de instâncias suspeitas, os passos anteriormente descritos até a extração de caracteristicas do GDD são executados e os vetores de características resultantes são submetidos aos modelos treinados. A Figura 1 exibe um diagrama correspondente ao passo a passo da abordagem proposta. Explicações mais detalhadas de cada um dos componentes são apresentadas nas seções seguintes.

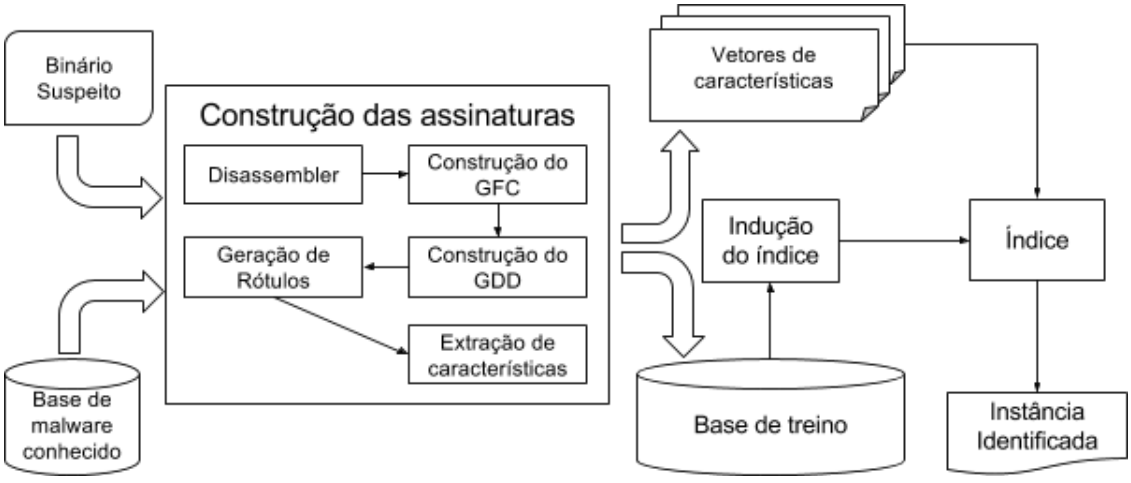


Figura 1. Visão geral da abordagem proposta para detecção de *malware* metamórficos.

Construção dos Grafos de Dependência de Dados

Para obter o GDD a partir de um código executável, a abordagem proposta emprega um processo de engenharia reversa para a reconstrução de um código assembler equivalente. Neste trabalho, este processo é feito através do Radare2 [Radare2 2017], uma ferramenta para análise de código binário que, além de efetuar o disassembler, gera os grafos de

© 2017 Sociedade Brasileira de Computação

chamadas de funções e de controle. Com o objetivo de obter uma melhor diferenciação entre as amostras, a saída deste processo gera um GFC para cada função no código. Esta análise é efetuada apenas no *payload* do código binário, excluindo as seções de dados e código externo como DLLs e funções importadas.

A próxima etapa utiliza o GFC para identificar as cadeias de dependência de dados. Isto é feito pela análise do fluxo de dados de forma a identificar todas as variáveis declaradas e as partes do código onde estas são manipuladas. Nesta fase, o GFC obtido previamente é percorrido ao longo das arestas seguindo um algoritmo iterativo de análise de dados baseado na abordagem conhecida como $worklist$ [Cooper et al. 2004].

O algoritmo $worklist$ proposto, exibido em Algoritmo 1, mapeia as cadeias de dependência com base em duas funções: *In()* e *Out()*. A função *In()* contém o mapeamento de todas as variáveis usadas em todos os blocos que possuem uma aresta partindo deles e chegando no bloco atual (bloco em análise). A função *Out()* mantém o registro das modificações introduzidas no bloco atual para controlar novos mapeamentos. Assim, a origem da cadeia de dependência é sempre atualizada caso exista uma instrução no bloco atual que modifique uma variável, atualizando os resultados que serão usados na análise do próximo bloco. Como consequência disso, blocos isolados não serão considerados na construção do GDD.

No início do processamento de um bloco, a primeira ação a ser tomada é a cópia do estado atual da função *In()* para a função *Out()*. A partir deste momento, para cada instrução no bloco em análise, uma das seguintes ações pode ser tomada:

i. Caso a variável esteja sendo manipulada pela primeira vez, um vértice correspondente a essa instrução é inserido no GDD e uma nova entrada na função *Out()* é inserida para futuras manipulações daquela mesma variável;

ii. Se a variável manipulada já estiver inserida *Out()* e o seu conteúdo não é alterado pela instrução atual, uma das seguintes ações são tomadas: a) caso a instrução de origem já possua um nó inserido no GDD é criado um novo nó correspondente a instrução atual; b) caso contrário, são criados dois nós correspondentes as instruções origem e atual. Em ambos os casos, uma nova aresta é criada ligando o nó origem e nó correspodente a instrução atual.

iii. Quando a instrução estiver alterando o conteúdo da variável, são executadas as mesmas operações descritas no item *ii*, e adicionalmente a instrução atual substituirá a instrução de origem na função *Out()*, visto que esta instrução passará a ser a nova origem da cadeia de dependência desta variável.

As Figuras 2 e 3 exibem um exemplo de um GFC e seu GDD correspondente. No GDD apresentado na Figura 3 existem arestas partindo dos nós referentes as instruções 3 e 1 que chegam no nó referente a instrução 4, pois a instrução 4 utiliza o registrador eax que foi manipulado anteriormente nas instruções 3 e 1.

Construção de Assinaturas

Nos modelos tradicionais de detecção a assinatura de um *malware* é definida como uma cadeia de bits única que representa cada amostra. Neste trabalho, os vetores de características (vc) extraídos a partir dos GDDs são considerados as assinaturas das amostras sementes, isto é, amostras de código de *malware* a partir dos quais as variantes me-

 © 2017 Sociedade Brasileira de Computação

Algoritmo 1 Abordagem Worklist para construção do GDD

```
function BUILDGDD (GFC)
    Out(s) ← ∅ for s in GFC
    V ← ∅
    E ← ∅
    W ← ∅
     W.Push(GFC.entryBB)
    while W ≠ ∅ do
        s ← W.Pop()
        In(s) ← ∩_{s'∈preds(s)} Out(s')
        temp ← In(s)
        for Instruction i in s do
            for Variable v read by i do
                if v exists in temp then
                    if i not in V then V.Add(i)
                    end if
                    if temp[v] not in V then V.Add(temp[v])
                    end if
                    E.Add(Vertex(i, temp[v]))
                end if
            end for
            for Variable v modified by i do
                temp[v] ← i
                if i not in V then V.Add(i)
                end if
            end for
        end for
        if Out(s) ≠ temp then W.Push(succs(s))
        end if
    end while
     GDD = Graph(V, E)
     return GDD
end function
```

tamórficas podem ser criadas.

Uma vez que o GDD é construído, o próximo passo consiste em atribuir um rótulo a cada nó do GDD. Este processo tem por objetivo reduzir os efeitos da técnica de obfuscação: substituição simples de instruções.

A construção dos rotulos é baseada na classificação das partes da instrução (i.e mnemonic e operandos) seguindo o padrão de nomenclatura $r_w = R_m_R_o$ para um rótulo r_w qualquer, onde R_m é a classe do mnemônico e R_o as classes dos operandos separadas pelo símbolo de sublinhado.

A classificação dos mnemônicos segue o padrão adotado pela arquitetura de referência x86 [Lejska 2017]. Existem um total de 60 classes para 709 mnemônicos. A Tabela 1 apresenta algumas das classes de mnemônicos utilizadas.

A classificação dos operandos segue o padrão adotado pela infraestrutura de compilador LLVM (*Low Level Virtual Machine*) [Lattner and Adve 2004], também adotada pelo disassembler Capstone [Nguyen Anh Quynh 2014] [Capstone-Disassembler 2017].

Um exemplo da atribuição dos rótulos a cada nó do GDD pode ser observado

© 2017 Sociedade Brasileira de Computação

Tabela 1. Exemplos de classificações dos mnemônicos propostas na referência x86asm[dot]net.

Classes	Mnemônicos
branch	jnl, jnae, call, jnp, ret, loopne, jno, jb, jnb, js, alter, jpe, ja, jmpe, retn
stack	push, popa, pushad, pushf, pushal, pop
datamov	setne, movd, setge, movlps, movbe, cmovo, sete, fistp
control	fdisi, wait, fstcw, hint, nop,fneni nop, fwait, ud2

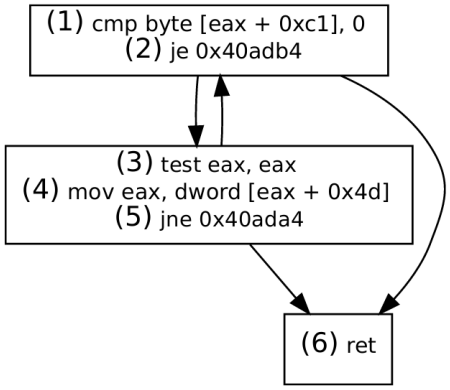


Figura 2. Exemplo de um GFC

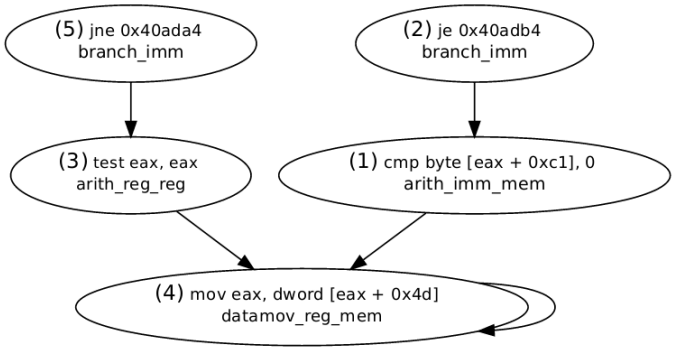


Figura 3. GDD rotulado extraído do GFC na Figura 2

na Figura 3. Na instrução (3), o mnemonic $test$ recebe a classe $arith$ e os operandos eax, eax recebem a classe reg.

Na medida em que o GDD é rotulado o seu vetor de características é atualizado. A abordagem proposta utiliza o padrão de nomenclatura $c_w = F_e_F_r$ para uma característica c_w qualquer extraída do GDD, onde F_r representa um rótulo qualquer e F_e corresponde aos prefixos derivados da estrutura do grafo n, i e o, sendo n a quantidade de ocorrências de F_r no grafo, i a quantidade de arestas entrando em nós contendo F_r e o a quantidade de arestas saindo de nós contendo F_r.

A Tabela 2 mostra um exemplo de um vetor extraído do GDD presente na Figura 3. Como o grafo contém dois nós com o rótulo *branch_imm*, então *n_branch_imm* possui o valor 2. Por outro lado, *i_arith_reg_reg* é atribuído o valor 1, pois existe apenas uma aresta entrando em nós que contém o rótulo *arith_reg_reg*.

Construção dos Índices

O método para detecção de *malware* proposto neste trabalho é baseado na comparação de vetores de características extraídas a partir dos GDDs das funções presentes em códigos binários suspeitos com aqueles de instâncias de *malware* conhecidos.

Devido aos GDDs serem menos suscetíveis à ação do metamorfismo de código, é esperado que instâncias geradas a partir de uma mesma semente sejam representadas

Tabela 2. Exemplo do vetor de características obtido do GDD na Figura 3.

n_branch_imm	i_branch_imm	o_branch_imm	n_arith_reg_reg	i_arith_reg_reg	...
2	0	2	1	1	...

 © 2017 Sociedade Brasileira de Computação

por um mesmo GDD e o seu vetor de características correspondente. No entanto, o processo de construção dos GDDs apresenta deficiências inerentes ao processo de engenharia reversa. Por este motivo, o processo de comparação de grafo é baseado em modelos baseados em padrões estatísticos ao invés de comparações exatas. Os modelos estatístico de identificação são gerados utilizando algoritmos de aprendizagem de máquina baseados em árvore de decisão.

Assim, o problema de identificar uma instância metamórfica gerada a partir de uma semente comum é visto como um problema de classificação, no qual as classes são as instâncias sementes de códigos maliciosos. Desta forma, dado um vetor de características de um GDD de um binário suspeito, o modelo deve ser capaz de reconhecer a classe (família de *malware*) a qual este binário pertence.

A escolha dos algoritmos de árvores de decisão se justifica por estes apresentarem um equílibro entre variância e viés [Munson and Caruana 2009]. Como os GDDs apresentam uma baixa variabilidade, é conveniente adotar um modelo que generalize pouco sem apresentar *overfitting* [1], sem perder as relações de relevância entre as características e as saídas esperadas.

Para a construção dos modelos mais precisos, o vetor de característica é submetido a um processo de redução com o objetivo de eliminar características com baixa variância. Uma vez selecionadas as características, um processo de treinamento supervisionado de modelos de classificação tradicional é executado [Kotsiantis et al. 2007]. A identificação da classe é definida pelo $hash\ md5$ da instância do *malware* a partir do qual foi extraído o vetor de característica.

Para identificar se uma instância suspeita pertence a alguma família de *malware* todos os passos descritos para a construção dos vetores de características são aplicados no código binário da instância em análise. Finalmente, os vetores obtidos são submetidos ao modelo treinado para obter a classe à qual pertence a instância de entrada.

Experimentação e Resultados

Com o objetivo de avaliar a abordagem proposta um conjunto de experimentos foram realizados. Devido ao dataset de vetores de características utilizado apresentar um tamanho superior aos 8GB quando carregado em memoria e para aproveitar a capacidade de procesamento em paralelo fornecida pelo framework utilizado para treino, estes experimentos foram executados em um computador com 128 GB de RAM e 62 núcleos físicos. Os algoritmos utilizados foram implementados na linguagem de programação Python versão 3.5.

Geração de instâncias metamórficas

A base de dados utilizada para a construção dos índices é composta pelos vetores de características extraídos dos GDDs das funções presentes no código *assembly* de instâncias de *malware* obtidos no repositório público de *malware Malshare* [Malshare 2017], os quais foram usados como sementes para gerar instâncias metamórficas.

Para gerar as instâncias metamórficas foram utilizados as ferramentas *Revert4* e *Code Pervertor* disponíveis em *VX Heaven* [VXHeaven 2017]. Estas ferramentas apli-

[1] quando o modelo estatístico se ajusta em demasiado ao conjunto de dados/amostra.

© 2017 Sociedade Brasileira de Computação

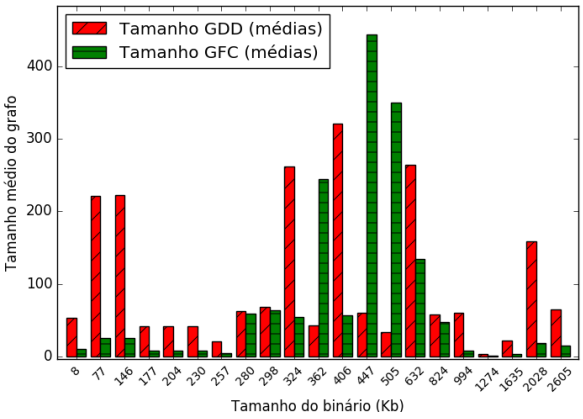


Figura 4. Tamanhos dos grafos por tamanho dos arquivos

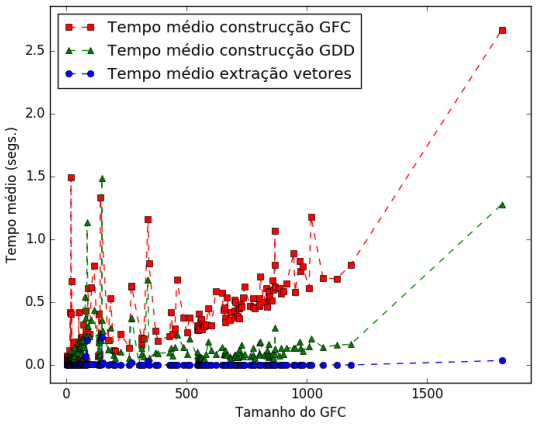


Figura 5. Tempo médio de construção das assinaturas por tamanho dos GFCs

cam técnicas de ofuscação como: a) Substituição de instruções equivalentes simples e em grupo; b) Reordenação de instruções; c) Inserção de código lixo; e d) Renomeação de variáveis. Devido à diversidade e complexidade das técnicas de ofuscação aplicadas, as instâncias geradas podem ser consideradas altamente metamórficas. No total foram utilizadas 301 instâncias de *malware* das quais foram geradas 1021 amostras metamórficas usando o *Code Pervertor* e 2024 amostras usando o *Revert4*.

Resultados

Processo de construção das assinaturas

A Figura 4 mostra o tamanho médio dos GFCs e GDDs por tamanho do arquivo, presentes no conjunto total de arquivos. É possível observar que não existe uma relação linear entre o tamanho dos arquivos e o tamanho do GFC e consequentemente o tamanho dos GDDs. Isto ocorre devido ao processo de engenharia reversa possuir falhas no reconhecimento e reconstrução de trechos do código binário original, e ao uso extensível de funções de bibliotecas externas no código que são descartadas no processo de engenharia reversa.

Como esperado, os GDDs são maiores que os GFCs, pois os nós nos GFCs representam blocos de instruções, enquanto nos GDDs cada instrução da origem a um nó. Entretanto, como pode ser observado na Figura 4, existem casos em que o GFC é maior que o GDD. Isto ocorre devido ao código original estar contaminado com muitas instruções lixo, o que acarreta o aumento na quantidade dos blocos de instruções no GFC. No entanto, estas instruções lixos serão ignoradas no momento da construção dos GDDs.

A Figura 5 apresenta o tempo médio de construção das assinaturas (GFC, GDD e vetores de características) pelo tamanho dos GFCs. Em média o tempo de construção de assinaturas é diretamente proporcional aos tamanhos do GFC. Para GFCs com 1700 nós, o tempo médio de construção de assinatura é de 4,13 segundos. Entretanto, como o tempo de construção do CFG engloba o processo de engenharia reversa, um código grande que consome muito tempo para ser analisado pode dar origem a um GFC pequeno.

 © 2017 Sociedade Brasileira de Computação

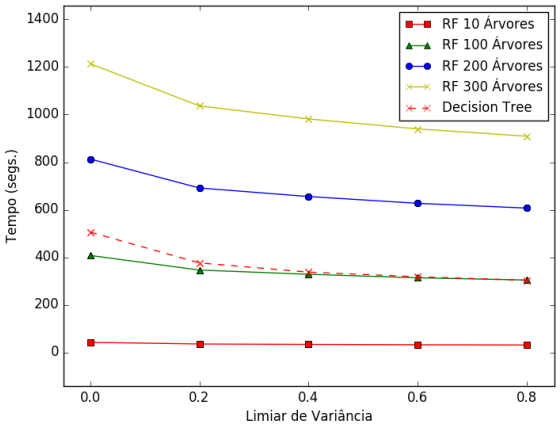


Figura 6. Tempos de treino dos modelos de classificação.

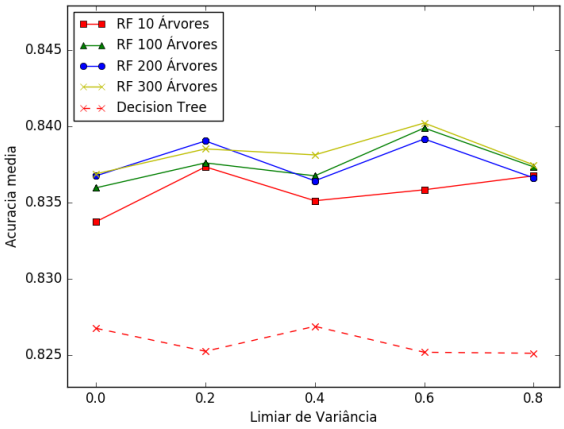


Figura 7. Acurácia dos modelos de classificação.

Avaliação da acurácia dos índices

Para avaliar a acurácia dos índices foram utilizados os seguintes classificadores: uma árvore de decisão C45 e florestas de árvores (*Random Forests*) com 10, 100, 200 e 300 árvores. Os modelos de classificação foram gerados usando 70611 amostras e testados com 30262 amostras. O cálculo da acurácia corresponde ao total de amostras corretamente classificadas sob o total de amostras utilizadas nos experimentos e é obtido pela seguinte fórmula:

$$acuracia(y, y') = \frac{1}{n_{vetores}} \sum_{i=0}^{n_{vetores}-1} 1(y'_i = y_i)$$

A Figura 6 mostra o tempo necessário para gerar os modelos de classificação em função do limiar de redução, obtido a partir da variância das amostras. Nos experimentos realizados foi observado que o tempo de geração dos modelos de classificação diminuia à medida que variância das amostras aumentava. Isto ocorreu pois o nível de variância dos valores das características foi usado como limiar de redução de características que seriam efetivamente usadas na construção do modelo.

A Figura 7 apresenta as acurácias médias dos modelos treinados para os diferentes limiares de variância utilizada para a redução do espaço de características. Os resultados dos experimentos mostram que o processo de redução também não afetou diretamente a acurácia dos modelos de classificação gerados.

Comparação com antivírus comerciais

Esta seção descreve os resultados da comparação da abordagem proposta com ferramentas comerciais disponibilizadas no sistema *VirusTotal* [Total 2017]. Foram selecionadas 300 amostras previamente identificadas pelos 58 antivírus comercias avaliados para a geração de novas instâncias metamórficas com o uso da ferramenta *Revert4*. No total foram geradas 2186 novas instâncias metamórficas as quais foram submetidas ao *VirusTotal*

© 2017 Sociedade Brasileira de Computação

Tabela 3. Acurácia média dos modelos obtidos comparado a antivirus comerciais

Posição Ranking	Detector	Acurácia média
1	McAfee-GW-Edition	88.43
2	CrowdStrike	85.13
3	McAfee	84.26
4	**RF N=300 V=0.6**	84.02
5	**RF N=100 V=0.6**	83.99
6	**RF N=200 V=0.6**	83.92
7	Qihoo-360	83.81
8	Ikarus	81.75
9	Rising	81.33
10	Avast	80.74
...	...	...
56	Paloalto	19.441903
57	ClamAV	15.141812
58	Webroot	13.586459

A Tabela 3 apresenta um ranking da acurácia média obtida pelos 10 melhores resultados incluindo os 3 melhores resultados dos modelos de identificação gerados neste trabalho. Também se incluem os três piores resultados do conjunto total, ao final da tabela. Os valores demonstram que a taxa média de acurácia obtida pelos modelos treinados é superior a maioria das ferramentas comercias disponíveis, provando a validade do modelo de identificação proposto.

Conclusões e Trabalhos Futuros

Este trabalho propôs uma abordagem para a identificação de *malware* resiliente ao metamorfismo de código. Os experimentos mostram (Figura 5) que a abordagem proposta apresenta tempos médios de execução menores aos 5 segundos para a construção das assinaturas utilizadas na detecção. Devido ao tamanho dos grafos não estar relacionado com o tamanho dos arquivos, não existe restrição neste sentido enquanto aos binários que podem ser analisados por esta proposta.

Apesar de estar sendo utilizado para a detecção apenas um GDD por binário analisado, os resultados na acurácia média são competitivos em comparação aos dos antivirus comerciais. Uma abordagem a ser estudada por ser considerada promisora pelos pesquizadores deste trabalho consiste em basear o resultado final da detecção na junção dos resultados da aplicação dos classificadores nos vetores de caracteristicas do conjunto de grafos do binário em análise. Desta forma é esperado o aumento da acuracia e a diminuição dos falsos positivos, aspecto este muito importante na area de detecção de malware.

Próximas etapas deste trabalho incluem o estudo de modelos treinados utilizando outros algoritmos de aprendizagem de maquina assim como a avaliação da qualidade dos classificadores utilizando métricas que considerem o risco de falsos positivos podendo assim também definir estrategias práticas para melhorar a classificação por medio do ajuste dos hiperparametros atendendo á características mais especificas em dependencia do contexto.

 © 2017 Sociedade Brasileira de Computação

Referências

Ahmadi, M., Sami, A., Rahimi, H., and Yadegari, B. (2013). Malware detection by behavioural sequential patterns. *Computer Fraud & Security*, 2013(8):11–19.

Alam, S., Sogukpinar, I., Traore, I., and Nigel Horspool, R. (2015a). Sliding window and control flow weight for metamorphic malware detection.

Alam, S., Traore, I., and Sogukpinar, I. (2015b). Annotated Control Flow Graph for Metamorphic Malware Detection. *The Computer Journal*, 58(10):2608–2621.

AV-Test (2015). AV-Test 2015 Security Report.

Capstone-Disassembler (2017). Capstone disassembler github repository. `https://github.com/aquynh/capstone`.

Cooper, K. D., Harvey, T. J., and Kennedy, K. (2004). Iterative data-flow analysis, revisited. Technical report.

Eskandari, M. and Hashemi, S. (2012). A graph mining approach for detecting unknown malwares. *Journal of Visual Languages & Computing*, 23(3):154–162.

Ferrante, J., Ottenstein, K. J., and Warren, J. D. (1987). The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):319–349.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Hu, X., Chiueh, T.-c., and Shin, K. G. (2009). Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 611–620. ACM.

Kim, K. and Moon, B.-R. (2010). Malware detection based on dependency graph using hybrid genetic algorithm. *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, page 1211.

Kotsiantis, S. B., Zaharakis, I., and Pintelas, P. (2007). Supervised machine learning: A review of classification techniques.

Kuriakose, J. and Vinod, P. (2014). Ranked linear discriminant analysis features for metamorphic malware detection. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 112–117. IEEE.

Kuriakose, J. and Vinod, P. (2015). Unknown metamorphic malware detection: Modelling with fewer relevant features and robust feature selection techniques. *IAENG International Journal of Computer Science*, 42(2):139–151.

Lattner, C. and Adve, V. (2004). LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California.

Lejska, K. (2017). X86 opcode and instruction reference.

Lin, D. and Stamp, M. (2010). Hunting for undetectable metamorphic viruses. *Journal in Computer Virology*, 7(3):201–214.

© 2017 Sociedade Brasileira de Computação

Malshare (2017). Public repository of malware of the malshare project. `http://malshare.com/about.php`.

Martins, G. B., Santos, P., Danrley, V., Souto, E., and Freitas, R. D. (2016). Identificação de Códigos Maliciosos Metamórficos pela Medição do Nível de Similaridade de Grafos de Dependência. In *Anais do XVI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 296–309.

Munson, M. A. and Caruana, R. (2009). On Feature Selection, Bias-Variance, and Bagging.

Nguyen Anh Quynh, C. (2014). Capstone: next generation disassembly framework. `http://www.capstone-engine.org/BHUSA2014-capstone.pdf`.

Paredes, R. and Chávez, E. (2005). Using the k-nearest neighbor graph for proximity searching in metric spaces. In *International Symposium on String Processing and Information Retrieval*, pages 127–138. Springer.

Rad, B. B., Masrom, M., and Ibrahim, S. (2012). Opcodes histogram for classifying metamorphic portable executables malware. In *2012 International Conference on E-Learning and E-Technologies in Education, ICEEE 2012*, pages 209–213. IEEE.

Radare2 (2017). Radare2 github repository. `https://github.com/radare/radare2`.

Raymond, J. W. and Willett, P. (2002). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computer-aided molecular design*, 16(7):521–533.

Singh, T., Di Troia, F., Corrado, V. A., Austin, T. H., and Stamp, M. (2015). Support vector machines and malware detection. *Journal of Computer Virology and Hacking Techniques*.

Symantec (2017). Symantec 2017 internet security threat report.

Total, V. (2017). Virustotal-free online virus, malware and url scanner. *Online: https://www.virustotal.com/en*.

VXHeaven (2017). Computer virus collection. *URL: http://vxheaven. org/vl. php*.

 © 2017 Sociedade Brasileira de Computação

Using Huffman Trees in Features Selection to Enhance Performance in Spam Detection

Cleber K. Olivo[1], Altair O. Santin[1], Luiz E. S. Oliveira[2]

[1] Graduate Program in Computer Science – Pontifical Catholic University of Paraná (PUCPR)
[2] Department of Informatics – Federal University of Paraná (UFPR)
Curitiba – PR - Brazil

{cleber,santin}@ppgia.pucpr.br, lesoliveira@inf.ufpr.

Abstract. *Spam detection is very costly when compared to the simple task of spreading spam. Most approaches aim to reach higher accuracy percentages, leaving the classification performance in background, what may cause many problems, such as bottlenecks in the e-mail system, huge infrastructure investments and waste of resources pooling. To avoid these problems, this paper proposes a hierarchical spam features organization using Huffman Trees, where the most important features stay closer to the root. With the reduction of these trees (leaves pruning) the feature space is significantly reduced, speeding up the e-mail classification process. The experiments showed a performance 60 times faster when compared to Spam Assassin.*

1. Introduction

Spam is the term assigned to messages sent indiscriminately, usually without the consent of the recipient. Although this term was first attributed to a message sent to multiple users in an electronic medium [1], spam is not limited to its virtual form. However, because it can be easily sent, it is in the electronic media that spam causes the biggest problems. In recent decades, even with the popularization of other communication services, e-mail has prevailed as one of the most popular tools for user communication, making it the preferred service for spreading spam.

Historically, spam statistics have always been worrying. In 1997, AOL estimated that 5 to 30 percent of its 10 million e-mails received per day were spam [2]. In 2004, a study predicted that this percentage would reach 95 percent on a global scale by 2015 [3]. Statistics released by Symantec showed a percentage of 89.1 percent in 2010 [4], being the highest percentage until then [5]. The spam percentages have declined in recent years, with 66% in 2013, 60% in 2014 and 53% in 2015. However, it is estimated that the daily global volume of e-mails on the Internet daily is approximately 190 billion, with a forecast increase of 4% (more than 197 million) by the end of 2016 [6].

Spamming goes far beyond the user annoyance. Internet service providers and their users have a high cost, caused by bandwidth waste and the costs of the technologies used to detect it [1]. In some more serious cases, such as phishing, the damage done can go far beyond a simple annoyance, causing the users to compromise their computers or even having financial losses [7].

Most of the proposed approaches in the literature do not consider the performance (resource consumption) of the antispam system as a key factor of the e-mail system, focusing only on the classifier's accuracy. Usually a large number of features is used to

 © 2017 Sociedade Brasileira de Computação

achieve a good percentage of correctness in the classification. The greater the number of features, the greater the complexity of the classification model and the consumption of resources. When a large volume of e-mail is received in a short period, if the classification system is too complex, there may be a queuing of messages that can be perceived by users.

This paper proposes a way to reduce the complexity of the classification model without great impact on the classifier's accuracy. The frequency of the words is used as a feature in the classification model. Words are arranged hierarchically in a Huffman Tree, so that the most frequent words in spam messages are closer to the root of the tree, and the less frequent words closest to the leaves. The size of the tree can be reduced, reducing the complexity of the classification model. The presented results showed that this approach can be promising.

The paper is organized as follows. Section 2 presents an overview of the e-mail system, the spam detection as a textual problem and some of the main approaches proposed to mitigate the problem. Section 3 explains the Huffman tree generation, and how this structure can be used to detect spam. Section 4 details the implementation of the proposal and the obtained results. Section 5 shows related works. Finally, Section 6 presents the conclusions about the proposal.

2. The E-mail Spam

Spam detection techniques based on message content (e.g. text and images) usually involve the area of pattern recognition. The classification of messages using these techniques is usually done in the e-mail servers, before arriving in the user's mailbox.

2.1. The E-mail Service

Simple Mail Transfer Protocol (SMTP) is the standard protocol for e-mail transfer [8]. The rules for exchanging messages between e-mail servers are defined in RFC 2821, which specifies that e-mail consists of an envelope and a content [8]. The content is divided into two parts: header and body. The e-mail body can be composed of images and text using hypertext features such as HTML (Hypertext Markup Language). The body of the e-mail is structured according to the MIME format – Multipurpose Internet Mail Extensions [9].

The SMTP protocol itself has limitations that are exploited by spammers, making it possible, for example, to replace the address in the sender's field with a different one from the e-mail of whom is sending the message. However, it's in the e-mail body that spammers use most of their tricks to circumvent antispam mechanisms.

2.2. Spam as a Textual Classification Problem

Spammers often develop new techniques for spreading spam to circumvent detection techniques. In the message content, they involve technical subterfuges to circumvent the textual classification filters. These techniques range from the purposeful insertion of words, which confuse the e-mail classifier, to the use of technical subterfuges, such as hypertext language (HTML) features incorporated into the e-mail.

In order to confuse antispam tools based on the words frequency, spammers can intentionally insert some words that are also common for non-spam messages. This can occur, for example, with Bayesian classifiers [10], which rely on the most frequent words

 © 2017 Sociedade Brasileira de Computação

for both classes (spam or non-spam) to determine whether a message is spam.

A technique known as textual obfuscation performs the exchange exclusion, or insertion of characters in the words. This creates a new string of characters, but remains providing the same human understanding. The single word 'viagra' can have more than one sextillion of variations [11]. This is possible due to the large number of characters existing in the Unicode pattern, which offers 120,672 codes representing multi-lingual characters, ideograms and symbol collections (v8.0) [12]. The negative effect of textual obfuscation was demonstrated by Liu and Stamm [13], through an experiment where the substitution of characters totally impaired the classification of spam messages.

As it can be seen, there are several factors that make it difficult to classify e-mails, requiring classifiers to use refined techniques or even preprocessing steps before the classification. The refinement of these techniques has as an aggravating factor the consequent increase of computational cost.

2.3. Approaches for Textual Spam Detection

Pattern recognition consists of the automatic discovery of patterns in the data, using computational algorithms that allow to classify the data in different categories [14]. Some of the most used techniques used in the fight against spam are the Bayesian classifiers, Neural Networks, and Support Vector Machines.

Among the proposals that use word frequency for the classification of e-mails, Bayesian filters are among the most common ones. Bayesian decision theory is a fundamental statistical approach to the classification problem [10], with the Naïve Bayes (NB) classifier being one of the most used for textual classification [15]. Its simplicity makes it easy to be implemented, usually achieving good accuracy, when compared to other more sophisticated machine learning algorithms [16]. NB has been one of the most applied algorithms in commercial antispam tools [17]. Chen, C. and colleagues [17] compared three proposals using NB [18, 19, 20] and four other traditional approaches (SVM, C4.5, NB and KNN). They conclude that it is better an improved NB algorithm than traditional methods of classification (including NB itself).

Drucker, H., Wu, S., and Vapnik, V. N [21] presented a piece of work that, showed several of the main concepts used in detection techniques based on terms frequency, such as TF (Term Frequency) – number of times a word appears in a document, TF-IDF (Term Frequency - Inverse Document Frequency) – which defines the relevance of a term within a collection of documents, and Stop List – a list of terms that should not appear in the features vector. In addition, they also present several widely-used techniques to evaluate the performance of the classifier and validate the results, as well as to study the use of the SVM classifier in comparison with other classification algorithms (Ripper, Rocchio and Boosting Decision Trees).

Due to the variety of existing techniques, there are several papers that analyze or test several of these techniques in the e-mail classification [22, 23]. Although highly promising, the most commonly used classifiers become known to spammers, who may exploit certain limitations in important stages of classification. A spammer may send messages that intentionally contain frequent words in e-mails that are not spam, so these words would proportionally generate false positives if those messages are included in the training database. This technique, known as evasion [24], is often used in other security-related applications such as IDS (Intrusion Detection System) [25].

© 2017 Sociedade Brasileira de Computação

3. Hierarchical Features Organization

Most e-mail classification approaches are not concerned with making better use of the features used in the classification model. A reduction in the number of features can bring significant performance gains for the antispam system in terms of reducing the computational cost, avoiding problems such as message queuing or resource wastage. A way to reduce the features number is organizing them according to their occurrence.

Some of the most widely used and accepted ways to measure the occurrence of terms in textual documents are TF (Term Frequency), IDF (Inverse Document Frequency) and TF-IDF (multiplication of TF by IDF). These metrics measure the importance of certain words in a document [26]. Due to its simplicity and good results during preliminary tests, TF was chosen to conduct the experiments.

This section shows how a Huffman Tree can better organize the classifier's features space. Huffman encoding is a lossless data compression technique, i.e., when data decompression occurs, the result is a bit-by-bit sequence identical to the original data sequence [27]. The data organization is done through a binary tree, in which a distribution of the data occurs based on the occurrence of symbols. The most frequent symbols are positioned closer to the root of the tree. In the case of spam, the symbols used to distribute the data in a Huffman tree may be the words found in the messages. The distribution of these words is arranged in the tree, so that the most frequent words of the database are closer to the root and the less frequent words are closer to the leaves.

Figure. 1 represents the hierarchical distribution of the spam words, based on the frequency at which they occur, in a Huffman Tree. At the first step (i) the symbols (words) are given a weight, which represents the frequency at which they occur on the spam database. Words are ordered by their respective weights in an increasing way, and then each pair of words have their weights added together. The sums are always made with the leftmost pairs, grouping from the lowest weight to the largest one. The result is five small trees, with their roots having the sum of the weights of their leaves. After grouping, roots should be rearranged based on weight, if necessary.

In the next step (ii), a new grouping is done, but the root of greater number (weight 68) ends up being left over. Again, the trees are rearranged so that the weighted root 68 is moved to the middle (iii). A new sum of the weights of the roots is made, and this time the remaining one is the root with weight 108. Finally, the two remaining roots are rearranged and have their weights added, forming a single tree (Huffman Tree).

As words are arranged in the tree, it is possible to observe that the most frequent ones (result, please, new, viagra, look and best) are positioned at a level closer to the root than the less frequent words (report, success, price, offer).

This hierarchical distribution facilitates the feature selection process, where less frequent spam words are usually further from the root. These features are usually non-discriminatory and therefore do not have much use in the message classification process. A reduced number of features (a reduced tree) results in a simplified classification model and, therefore, less computationally costly. With multilevel distribution, the tree can be reduced (pruned) level by level, significantly reducing the number of features at each pruning, until the result of classification starts to be impaired. The remaining tree contains the most discriminating and most useful features (words) for the classifier.

 © 2017 Sociedade Brasileira de Computação

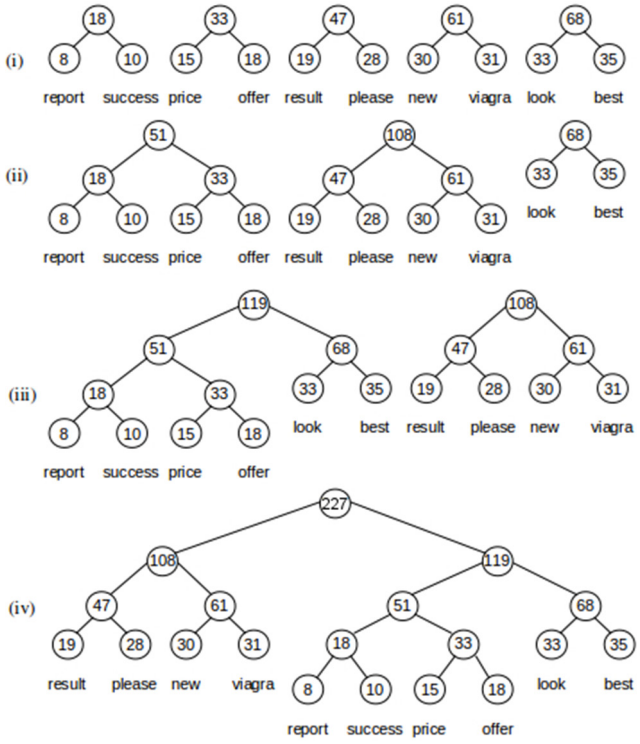


Figure 1. Feature Distribution in a Huffman Tree

4. Implementation of the Proposal

The proposal of this research was implemented in 4 main stages. The first step (section 4.1) involved the preparation of the databases and features extraction. The second step (section 4.2) is the e-mail classification, that also involves a gradual pruning of the Huffman Tree. In third step (section 4.3), performance tests were conducted. Finally, (section 4.4) the results of this approach were compared to Spam Assassin.

4.1. Database Preparation and Features Extraction

This proposal used different e-mail databases that are available online. One of the chosen databases was the Enron-Spam corpus [28, 29]. The Enron-Spam database consists of spam messages received by six users, with spam from various sources [30]. There is a total of 19,088 legitimate messages (not spam) and 32,988 spams. It is a public database very explored in the literature and still has non-spam messages. However, this is a very old database considering the evolution of spam over the years.

Thus, this proposal also used a second and a third database that were composed of e-mails from the database Untroubled Spam [31]. It is also available online and contains messages collected since 1998, but does not have non-spam e-mails. In addition, e-mails from users of mailing lists were collected on the Internet to compose the non-spam database. The second database (Untroubled 2015) consisted of 182,933 e-mails, of which 111,788 were spam (with spam of the whole year of 2015) and 71,145 were not spam e-mails. The third database (Untroubled 2016) consisted of 264,647 e-mails, with 212,216 spam messages (with spam of the whole year of 2016) and 52,431 non-spam messages. These numbers of spam messages were achieved after removing e-mails composed essentially of Asian charsets (e.g. KOI8-R, SHIFT-JIS, EUC-KR etc.), that are more than

© 2017 Sociedade Brasileira de Computação

700 thousand messages in the original 2015 and 2016 Untroubled databases. We consider that classifiers specialized in specific idioms are better than generalized classifiers. This proposal is focused in English language.

Prior to feature extraction, all the terms found in the training databases were validated after a search in an English dictionary. Terms that were not present in the dictionary were not considered as a feature. Additionally, all databases were preprocessed for e-mail header removal, stop-word removal, and suffix reduction.

The Python NLTK (Python Natural Language Toolkit) [32] was used for the removal of non-English words, stop-words and word stemming. Stop-words are terms that are among the most common in a language, for example articles and prepositions. The removal of these words is recommended before processing natural language, since they do not have much use in the classification process, and may even spoil the result. The reduction of suffixes (stemming) was done using the Snowball algorithm. For example, the words "receive" and "receiving", after being processed by the Snowball algorithm, both result in "receiv". Once the stop words removal and the word stemming were done, the features vectors were extracted using TF as a measurement of the words frequencies.

By containing all the "known" features of a spam database, the Huffman Tree was essential for the execution of the work. Therefore, it was of total importance that the Huffman tree was composed only by spam features collected in the training database. The features of non-spam messages were not used in this work. In other words, if a word appears only in e-mails that are not spam or spam messages in the test database, it does not appear in the tree of that classification model.

The Huffman Tree can also be used as a tool to see the more significant changes in spam features (most used terms) among bases or over the years. Table 1 lists the most common terms from the top 50 that are common to all the three bases. That is, after collecting the top 50 words for each base, there are 22 terms that common to all of them.

Considering the Enron Spam database is more than one third smaller than the Untroubled databases, for the training and test steps, similar to k-fold cross-validation, the database were divided into 10 parts (10-fold cross validation). In 10 steps, each part k is separated to test the model, and the remaining parts k-1 are used to train the model. Then a Huffman Tree was generated for each k-1, ensuring that there was a unique set of features and a distinct frequency of words for each step of the cross-validation. The experiments with Untroubled 2015 and 2016 databases were conducted with half of the messages for training the models and the other half for testing. The Untroubled Huffman trees were generated only with the training sets.

Table 1. Most Common terms from top 50 (occurring in all spam databases)

Also	Best	Day	Get	Go	Like	Look	Make
New	One	Order	See	Stop	Take	Time	Us
Use		Visit	Want	Way	Work	Year	

The Enron-Spam tree was composed of 13,159 words that were distributed in 12 different levels. In the trees formed with messages from both Untroubled Spam databases and mailing lists there were 9,067 and 10,161 features distributed in 13 different levels.

 © 2017 Sociedade Brasileira de Computação

4.2. E-mail Classification

The classifier applied for the messages classification was Liblinear [33, 34]. When the number of instances and features is very large, the Liblinear classifier usually consumes less memory, being much faster and with a precision very similar to LibSVM [35], which is usually one of the best classifiers for the detection of spam [17, 21, 23]. Before creating the model, the best value to calibrate each classifier parameter was obtained. The classification method used was the primal-based support vector classification. Table 2 shows the accuracies after each Huffman Tree pruning. In a simple definition, accuracy represents the total of samples correctly classified.

Table 2. Accuracy rate after Huffman tree pruning

No. of Pruning	E-Mail Database		
	Enron	Untroubled 2015	Untroubled 2016
0	98.12	97.90	98.66
1	98.04	97.88	98.64
2	98.14	97.86	98.62
3	97.58	97.70	98.59
4	95.64	97.63	98.50
5	95.26	97.48	98.36
6	94.70	97.32	98.31
7	93.45	97.23	97.55
8	90.46	95.97	97.27
9	84.93	95.53	96.55
10	70.16	92.26	94.67
11	-	87.19	91.68
12	-	75.99	84.76

Note that, for the Enron database there was a decrease of only 0.54% after 3 consecutive pruning steps, which proves that the essential features remain at the top of the Huffman Tree. After 4 consecutive prunings the accuracy drops more than 1%. With 11 prunings the result is inconclusive due to the lack of features for classification.

For the 2015 database, the result after the third pruning decreases only 0.2%. The accuracy kept above 97% even after the seventh pruning. The Untroubled 2016 had a result with only 0.16% lower than the original tree after 4 prunings.

Table 3 shows the number of terms present in the tree after each pruning step. One can be noticed that, for example, in the Enron database, with a tree divided in 9 levels (after three prunings), the accuracy of the classifier drops only 0.54%, but with a reduction of more than 71% in the number of features). Reducing the number of features results in a lighter classifier, providing significant performance gains, as will be shown in Section 4.3. Moreover, an effective feature selection leads to parsimonious classifiers that require

© 2017 Sociedade Brasileira de Computação

less memory and are faster to train and test, and can also reduce the feature extraction costs and lead to better generalization [36].

Table 3. Quantity of Features After Tree Reductions

Tree Size	E-mail Database		
	Enron	Untroubled 2015	Untroubled 2016
13	-	9066	10160
12	13159	8544	9219
11	11566	6867	7420
10	7424	5487	5459
9	3744	4206	4118
8	2402	3111	3147
7	1578	2430	2423
6	1021	1900	1940
5	633	1500	1487
4	338	967	794
3	124	333	349
2	27	80	102
1	1	10	15

If the system administrator considers an accuracy of 97% as acceptable, the reduction of the feature space is even greater for the Untroubled bases, reaching a reduction of 79% (a tree of 6 levels for 2015) and 85% (a tree of 5 levels for 2016).

Table 4. False Positives and False Negatives

Prunings	Enron		Untroubled 2015		Untroubled 2016	
	FP Rate	FN Rate	FP Rate	FN Rate	FP Rate	FN Rate
0	0,013	0,022	0,036	0,012	0,041	0,007
1	0,012	0,024	0,036	0,012	0,042	0,007
2	0,012	0,022	0,036	0,012	0,041	0,007
3	0,010	0,032	0,037	0,014	0,042	0,007
4	0,056	0,036	0,038	0,014	0,044	0,008
5	0,060	0,040	0,039	0,016	0,046	0,009
6	0,068	0,044	0,040	0,018	0,047	0,009
7	0,071	0,062	0,041	0,019	0,063	0,015
8	0,129	0,076	0,053	0,032	0,066	0,018
9	0,175	0,137	0,056	0,038	0,074	0,025
10	0,492	0,186	0,083	0,074	0,096	0,043
11	-	-	0,151	0,113	0,165	0,063
12	-	-	0,329	0,128	0,333	0,087

As a complementary evaluation, the false positives (FP) and false negatives (FN)

 © 2017 Sociedade Brasileira de Computação

are also important to choose the classifier. The values, of these rates are shown in table 4. Note that for the Enron Spam database the FP and FN rates remain practically unchanged even after the occurrence of two prunings. If the administrator considers the false positives a bigger problem than false negatives, the FP rate drops slightly after three prunings. However, it will result in a worst FN rate. Untroubled bases had minimal variations after five or six prunings.

The Enron Spam test database was formed with 3299 spam messages and 1909 non-spam messages. The Untroubled 2015 test database was formed with 55,895 spams and 35,573 non-spam and the Untroubled 2016 test database was formed with 106,111 spam and 26,216 non-spam.

4.3. Performance Tests

The performance tests showed significant performance gains after the reduction of the number of features used by the classifiers, as can be seen in table 5. The values were obtained computing an arithmetic mean, after repeating the experiments 10 times. The coefficient of variation was below 3%. Since the time required for training and classification of e-mails depends heavily on the environment (software and hardware) in which it occurs and the number of classified messages, the time required for the original classifiers (without reduction of features) is considered as 100%. The execution times of the classifiers that suffered reduction in the number of features are presented as a fraction of the time of the original classifier, also in percentages.

The real times took to complete the testing steps (with the original feature space – represented as 100% in table 5) had an average of 65ms, 1s171ms and 1s205ms, respectively. These times refer only to the classification process (after preprocessing and feature extraction). All the performance tests were performed in the same environment and with the same conditions: Intel® CoreTM i7 4510U (4 threads, 2 GHz up to 3.1GHz, 4MB L3 Cache), 8GB RAM and the operating system GNU/Linux Mint v17.3.

At the Enron database, it was observed that with three prunings the percentage of correctness remained almost the same, without significant changes in terms of false positives and false negatives, and the reduction in the number of features was more than 71%. Additionally, it takes 77% of the time of the original tree, as can be seen in table 5. This time reduction of 23% in the test stage (most crucial step in a real environment) could avoid a bottleneck in the e-mail system and also the waste of resources.

The Untroubled 2015 had a reduction of only 2.65% of time in the testing stage after the third pruning and Untroubled 2016 had a reduction of 5.58% (with minimal FP x FN changes). However, supposing the e-mail server is receiving an amount of e-mails bigger than it can handle, the classifier model could be switched to a simpler one that obeys a minimal requirement, i.e., an accuracy above 97%. This requirement allows to apply the feature sets with seven and eight prunings, representing a time reduction of almost 13% and 21%. After the contingency, the antispam system could make use of the more complex classifier again.

It is important to emphasize that, although the increase in performance is one of the main contributions of this research, the accuracy of the classifier is not in the background. Obviously, the main effort was to demonstrate the effect of non-discriminating features on the cost of the classification process (there was no great effort to raise the accuracy of the classifier). For example, accepting an accuracy of 98% does

© 2017 Sociedade Brasileira de Computação

not mean ignoring 2% of e-mails. The lighter classifier could be used only at a critical time, in which the server would have many messages queued, until the situation is normalized. However, opting for a faster classifier with almost the same results would be the most correct and proactive decision to make.

Table 5. Performance results (testing step)

Pruning	Enron	Untroubled 2015	Untroubled 2016
0	100,00	100,00	100,00
1	95,41	99,95	98,33
2	90,31	98,96	97,90
3	77,04	97,35	96,29
4	73,47	95,22	94,42
5	68,88	93,13	92,16
6	64,29	90,69	89,74
7	58,16	87,11	85,25
8	46,94	80,11	79,11
9	30,61	67,60	63,11
10	17,86	40,29	44,55
11	11,22	18,27	24,21
12	-	7,42	9,87

4.4. Tests with Spam Assassin

Tests with Spam Assassin [37] were conducted with two different purposes: i) compare the e-mail classification accuracy of this approach with a well-known antispam tool; ii) find some criteria to check if this approach has an acceptable performance at training and classification stages, even without any code improvement.

Spam Assassin reached an accuracy of 99.90% on the Enron database, 99.86% on Untroubled 2015 and 99.56% on Untroubled 2016. The tool threshold was changed (manually) to obtain the best possible result. These results evidence the feasibility of the proposed approach, once a better accuracy could be reached testing another classifier or using another feature representation (e.g. TF-IDF).

The performance comparison was done by sampling, using a total of 2 thousand e-mails, which were half spam and half non-spam. This step took 10s294ms to perform both classes training. The testing stage, where classification occurs, was executed with the reading of each e-mail of the testing dataset and took 471s434ms to complete.

The training stage of this approach involves the header and stop-words removal, dictionary checking, word stemming, TF calculation, creation of the Huffman Tree, feature extraction and creation of the classification model by LIBLINEAR. All these tasks took 10s728ms. At the testing stage, the e-mails of the test dataset were preprocessed likewise the training stage and the feature vectors were composed. The features vectors were submitted to the classifier, finishing the testing process in 7s743ms. When comparing the real time took for this testing step (with preprocessing and feature extraction) to the real time took by the classification job (section 4.3), it is evident that the heaviest work is previous to the classification itself.

As an overview of the results, the training process of this approach was 4.21% times slower than SpamAssassin, but the testing stage was 60 times faster. It must be

 © 2017 Sociedade Brasileira de Computação

considered that SpamAssassin is a very robust antispam tool and the time calculation does not consider its internal processes during training and classification. However, it should also be considered that this approach was coded in Python language, without any special programming treatment to improve performance. Moreover, the main difference in time occurs at the testing stage, which is the more important (computational costly) step in real environments to prevent bottlenecks in e-mail ingoing. Another advantage of this approach is that these times can be reduced even more with the tree pruning, as shown in table 5.

5. Related Works

The feature subset selection is a procedure which can reduce not only the cost of pattern recognition, but provides a better classification accuracy in some cases [39]. In spam detection as a problem of textual classification, the feature space has a high dimension. A feature space with a large number of terms can be unsuitable for a classifier or even decrease the categorization accuracy, once the ambiguity of meaning of terms can also prohibit the classifier from choosing the categories deterministically [40].

Meng, J. et. al. (2011) proposed a feature selection method named feature contribution degree (FCD). The FCD calculation considers that the features have different distributions in different categories. When a feature has a larger value of FCD, the feature probably has a stronger distinction capacity for categorization. Thus, if the feature has a larger value of FCD, it has a better chance of being selected [40].

Wiratunga et al. evaluated the use of generalization after feature selection in a spam base (LingSpam corpus). The results of generalization combined with boosting outperformed the other algorithms, with the highest accuracy approaching 99% with only 110 features. However, the authors suggest the domain is relatively easy because the LingSpam corpus has a few very discriminatory features for non-spam messages that are sufficient to differentiate spam messages [41].

Trivedi and Dey tested the Genetic feature search and Greedy Stepwise feature search techniques. For the Enron dataset, with only 48 best features out of 1500 initially created features, the Genetic Search reached an accuracy of 87.1% and the Greedy Search reached an accuracy of 94.2% [42].

Most of the related work propose improvements in the feature selection method. There is not much information about the impacts of feature selection in e-mail classification performance, being the main efforts concentrated in classifiers accuracy. Furthermore, when studying spam, a critical issue is the possibility to observe how it changes over the time (due to the spammer tricks or seasonality), what is possible using a tool such as the Huffman Tree.

6. Conclusions

This approach has as main objective the reduction in the time necessary for the spam detection, without significant lost in the accuracy of the classification. This avoids wasting resource pooling, required infrastructure investments, and is also useful in situations such as message queuing caused by the antispam system. Although the accuracy of the classifier is not the main contribution, the results obtained were very close to what was achieved with Spam Assassin. Thus, better results would probably be obtained if there was a special effort to do it.

© 2017 Sociedade Brasileira de Computação

The features organization in a Huffman Tree is a simple and efficient solution and can be easily implemented by hierarchically organizing the features of spam with the most relevant terms quickly accessed (i.e., positioned closer to the root). This form of organization could also, through the observation of trees obtained in different months or years, analyze in a different way the changes in the main words of spam in different epochs. The use of trees of different sizes to train the classifier is an approach that demonstrates a lot of feasibility and flexibility if used in a real environment.

The proposed method of using Huffman Trees to organize and reduce the feature space could also be easily implemented for another idiom such as Portuguese. The Huffman Tree could be reconstructed or updated as necessary, for example, if the administrator perceives an increase of false positives or false negatives.

In general, the main objectives of this work were achieved. It was evidenced that a waste occurs in the number of features used in the classification models. Most of these features are non-discriminatory and, therefore, have no use for the classifier. The performance tests found the wastage in processing time when no type of feature selection is performed. Complementary tests also have shown that the classification phase is 60 times faster than the well-known Spam Assassin tool.

7. References

[1] K. Kleiner. *Happy Spamyversary! Spam Reaches 30* [Online]. Available: https://www.newscientist.com/article/dn13777-happy-spamiversary-spam-reaches-30/

[2] B. Hoanca, "How good are our weapons in the spam wars?", IEEE Technology and Society Magazine, vol. 25, issue 1, pp. 22-30, 2006.

[3] B. Whitworth and E. Whitworth, "Spam and the social technical gap", IEEE Computer, vol. 37, issue 10, pp. 38-45, 2004.

[4] Symantec, “January 2011 Intelligence Report” [Online]. Available: https://www.navixia.com/images/pdf/newsletter/MLI_2011_01_January_Final_en-us.pdf

[5] Symantec, “May 2013 Intelligence Report” [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/security-center/archives/intelligence-report-may-13-en.pdf

[6] Symantec, “Internet Security Threat Report” [Online]. Rep. 21, Apr. 2016. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

[7] Olivo, C. K.; Santin, A. O.; Oliveira, L. S., “Obtaining the Threat Model for E-mail Phishing”, Applied Soft Computing, vol. 13, issue 12, pp. 4841-4848, 2013.

[8] J. Klensin. (2001, April). *RFC 2821 - Simple Mail Transfer Protocol* [Online]. Available: http://www.ietf.org/rfc/rfc2821.txt

[9] N. Freed and I. Borenstein. (1996, November). *RFC 2045 - Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* [Online]. Available: http://tools.ietf.org/rfc/rfc2045.txt

[10] R. Duda, P. Hart and D. Stork, *Pattern Classification*, 2nd edition, Wiley-Interscience, 2000.

 © 2017 Sociedade Brasileira de Computação

[11] *There are 600,426,974,379,824,381,952 ways to spell Viagra* [Online], Available: http://cockeyed.com/lessons/viagra/viagra.html

[12] *The Unicode Standard – Technical Introduction* [Online]. Available: http://www.unicode.org/standard/principles.html

[13] C. Liu and S. Stamm, "Fighting Unicode-Obfuscated Spam", Proceedings of the Anti-Phishing Working Group - 2nd Annual eCrime Researchers Summit, pp. 45-59, ACM, 2007.

[14] C. Bishop, *Pattern Recognition and Machine Learning*, 1st edition, Springer, 2007.

[15] K. Schneider, "A comparison of event models for Naive Bayes anti-spam e-mail filtering", Proceedings of the 10th conference on European chapter of the Association for Computational Linguistics, vol. 1, pp. 307-314, ACM, 2003.

[16] I. Androutsopoulos, G. Paliouras and E. Michelakis, "Learning to Filter Unsolicited Commercial E-Mail" [Online], NCSR "Demokritos" Technical Rep. 2004/2, Mar. 2004. Available: http://nlp.cs.aueb.gr/pubs/TR2004_updated.pdf

[17] C. Chen, Y. Tian and C. Zhang, "Spam Filtering with Several Novel Bayesian Classifiers", IEEE 19th International Conference on Pattern Recognition, pp. 1-4, 2008.

[18] E. Frank, M. Hall and B. Pfahringer, "Locally Weighted Naive Bayes", Proceedings of the 19th conference on Uncertainty in Artificial Intelligence, ACM, pp. 249-256, 2002.

[19] H. Zhang, L. Jiang and J. Su, "Hidden Naive Bayes", Proceedings of the 20th National Conference on Artificial Intelligence, vol. 2, ACM, pp. 919-914, 2005.

[20] G. Webb, J. Boughton, and Z. Wang, "Not so Naive Bayes: Aggregating One-Dependence Estimators", Machine Learning, vol. 58, issue 1, pp. 5-24, 2005.

[21] H. Drucker, S. Wu and V. Vapnik, "Support Vector Machines for Spam Categorization", IEEE Transactions on Neural Networks, vol. 10, issue 5, pp. 1048-1054, 1999.

[22] R. S. S. Kiran and I. Atmosukarto, "Spam or Not Spam – That is the Question" [Online], Technical Report, University of Washington, 2005. Available: http://courses.cs.washington.edu/courses/cse573/04au/Project/mini2/Ravi&Indri/spamfilter_ravi_indri.pdf

[23] T. Guzella and W. Caminhas, "A Review of Machine Learning Approaches to Spam Filtering", Expert Systems with Applications, Elsevier, vol. 36, issue 7, pp. 10206-10222, 2009.

[24] B. Nelson, B. Rubinstein, L. Huang, A. Joseph and J. Tygar, "Classifier Evasion: Models and Open Problems", Privacy and Security Issues in Data Mining and Machine Learning, vol. 6549, Lecture Notes in Computer Science, p. 92-98, Springer, 2011.

[25] M. Barreno, B. Nelson, R. Sears, A. Joseph and J. Tygar, "Can Machine Learning be Secure?", Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, pp. 16-25, 2006.

[26] *"TF-IDF: A Single Page Tutorial – Information Retrieval and Text Mining* [Online]. Available: http://www.tfidf.com

© 2017 Sociedade Brasileira de Computação

[27] M. Sharma, “Compression Using Huffman Coding”, International Journal of Computer Science and Network Security, vol. 10 no.5, pp. 133-141, 2010.

[28] V. Metsis, I. Androutsopoulos and G. Paliouras, "Spam Filtering with Naive Bayes – Which Naive Bayes?", Proceedings of the 3rd Conference on E-mail and Anti-Spam (CEAS 2006), Mountain View, CA, USA, 2006.

[29] *The Enron-Spam Datasets* [Online], Available: http://www.aueb.gr/users/ion/data/enron-spam

[30] G. Cormack, “E-mail Spam Filtering: A Systematic Review”, Foundations and Trends in Information Retrieval, vol. 1, n.4, pp. 335-455, 2006.

[31] *Untroubled Spam Archive* [Online]. Available: http://untroubled.org/spam/

[32] *Natural Language Toolkit – NLTK 3.0 documentation* [Online]. Available: http://www.nltk.org

[33] R. Fan, K. Chang, C. Hsieh, X. Wang and C. Lin, “LIBLINEAR: A Library for Large Linear Classification”, Journal of Machine Learning Research, vol. 9, 2008.

[34] *LIBLINEAR: A Library for Large Linear Classification* [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/liblinear

[35] C. Hsu, C. Chang and C. Lin. *A Practical Guide to Support Vector Classification* [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[36] Z. Xu, G. Huang and K. Weinberger, "Gradient Boosted Feature Selection", Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 522-531, 2014.

[37] *SpamAssassin – The #1 Enterprise Open-Source Spam Filter* [Online], Available: http://spamassassin.apache.org

[38] *SpamAssassin Configuration File* [Online]. Available: http://spamassassin.apache.org/full/3.3.x/doc/Mail_SpamAssassin_Conf.html#language_options

[39] A. Jain and D. Zongker, "Feature Selection: Evaluation, Application, and Small Sample Performance", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, issue 2, pp. 153-158, 1997.

[40] J. Meng, H. Lin and Y. Yu, "A Two-stage Feature Selection Method for Text Categorization", Computers and Mathematics with Applications, Elsevier, pp. 2793-2800, 2011.

[41] N. Wiratunga, I. Koychev and S. Massie, "Feature Selection and Generalization for Retrieval of Textual Cases", Proceedings of the 7th European Conference on Case-Based Reasoning, Springer Verlag, pp. 806-820.

[42] S. Trivedi and S. Dey, "Effect of Feature Selection Methods on Machine Learning Classifiers for Detecting Email Spams", Proceedings of the 2013 Research in Adaptive and Convergent Systems, ACM, pp. 35-40, 2013.

 © 2017 Sociedade Brasileira de Computação

Experiências com um Honeypot DNS: Caracterização e Evolução do Tráfego Malicioso

Tiago Heinrich[1], **Felipe de Souza Longo**[1], **Rafael R. Obelheiro**[1]

[1] Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC – Brasil

tiagoheinrich1995@gmail.com, felipeslongo@gmail.com, rafael.obelheiro@udesc.br

Abstract. *The Domain Name System (DNS) plays a central role in the operation of the Internet, being responsible for translating user-friendly names into machine-friendly IP addresses. However, the DNS has certain structural security vulnerabilities which allow it to be attacked or used as a tool for attacking third parties. Currently, a major concern are Distributed Reflection Denial of Service (DRDoS) attacks, which leverage misconfigured DNS servers to flood victims with traffic. This paper introduces DNSpot, a DNS-specific honeypot that allows attackers to interact with an open recursive DNS server in a controlled manner. We also analyze the DNS traffic observed by this honeypot over two periods, in 2015 (49 days) and 2016–2017 (250 days), with a focus on DRDoS attacks, and highlight some noteworthy aspects of attacker behavior.*

Resumo. *O Domain Name System (DNS) desempenha um papel central no funcionamento da Internet, sendo responsável por traduzir nomes mnemônicos em endereços IP. No entanto, o DNS possui certas vulnerabilidades estruturais de segurança, que permitem que ele seja atacado ou usado como instrumento de ataque a terceiros. Atualmente, uma grande preocupação são os ataques distribuídos de negação de serviço por reflexão (Distributed Reflection Denial of Service, DRDoS), que se aproveitam de servidores DNS mal configurados para saturar vítimas com tráfego. Este artigo introduz o DNSpot, um honeypot específico para DNS que permite que atacantes interajam com um servidor DNS recursivo aberto de forma controlada. O artigo também apresenta uma análise do tráfego DNS observado pelo honeypot durante dois períodos, em 2015 (49 dias) e 2016–2017 (250 dias), com foco em ataques DRDoS, destacando também aspectos notáveis do comportamento de atacantes.*

1. Introdução

O DNS (Domain Name System) [Mockapetris 1987] é um sistema distribuído de resolução de nomes que desempenha um papel vital na Internet. Ele possibilita que usuários refiram-se a nós da rede empregando nomes mnemônicos (como `www.google.com`) em vez de endereços IP (como 216.58.219.110), sendo responsável, entre diversas outras funcionalidades, pela tradução de nomes em endereços.

Em vista de sua ampla utilização, o DNS também é alvo e vetor de ataques. As principais ameaças envolvendo o DNS são resumidas por [Conrad 2012], que as divide em duas classes, ameaças ao próprio DNS e ameaças oportunizadas pelo DNS. A classe de ameaças ao DNS inclui:

 © 2017 Sociedade Brasileira de Computação

- Negação de serviço: bloqueio do acesso de usuários ao DNS, com isso prejudicando ou mesmo impedindo (na prática) o seu acesso à Internet;
- Corrupção de dados: modificação não autorizada dos dados publicados no DNS, o que pode, por exemplo, levar usuários a acessar sites ilegítimos (como sites falsos de bancos ou de comércio eletrônico);
- Exposição de informação: revelação de informações sobre o comportamento dos usuários, como histórico de sites web acessados.

O DNS também pode ser usado como vetor de ataques. A classe de ameaças oportunizadas pelo DNS abrange:

- Ataques de amplificação: servidores DNS mal configurados podem ser usados para realizar ataques distribuídos de negação de serviço por reflexão (*Distributed Reflection Denial of Service*, DRDoS) contra terceiros [CERT.br 2016];
- *Fast flux* DNS: servidores usados para propósitos nefastos, como propagação de software malicioso, podem receber nomes que são mudados com alta frequência, com o propósito de dificultar a localização dos servidores e a identificação dos seus responsáveis;
- Exfiltração de dados: como o tráfego DNS geralmente passa incólume por *firewalls*, ele é usado com frequência para transmissão de dados sensíveis (capturados no curso de uma invasão) de forma sub-reptícia.

Dentre as ameaças acima, os ataques DRDoS despertam particular interesse. Em seu relatório anual sobre segurança na Internet [Arbor 2017], a Arbor Networks reportou que, de acordo com tráfego observado por diversos sensores espalhados pela Internet, em 2016 o DNS foi usado em 47% dos ataques DRDoS, gerando um volume médio de 3.083 Mbps por ataque, ficando à frente de outros protocolos em ambos os quesitos. Ao longo de 2016, os ataques DDoS baseados no DNS aumentaram de 10.500 para 18.500 por semana, em média. Um outro relatório recente da Akamai [Akamai 2017] reporta que, no primeiro trimestre de 2017, 57% dos ataques DDoS observados eram baseados em reflexão, e que o DNS foi um dos protocolos mais usados.

A Figura 1 ilustra o funcionamento de um ataque DRDoS usando o DNS. Um atacante envia a um servidor DNS recursivo[1] um número elevado de consultas com o endereço IP de origem forjado como sendo o da vítima do ataque (passo 1); o servidor recursivo interage com servidores autoritativos para obter a resposta para a consulta (passos 2 e 3); a resposta é enviada para a vítima (passo 4). O ataque é facilitado pelo fato de consultas DNS pequenas poderem gerar respostas grandes; como será discutido na Seção 4, esse fator de amplificação pode chegar a 100 (um tráfego de consulta de 1 Mbps gera um tráfego de resposta de 100 Mbps). Se forem usados diversos clientes e servidores simultaneamente (por exemplo, uma *botnet* de clientes usando servidores DNS distintos), é relativamente fácil gerar uma intensidade de tráfego suficiente para sobrecarregar a vítima, indisponibilizando sua conectividade.

Uma forma de observar o comportamente de atacantes é usando *honeypots*, que são recursos computacionais com o objetivo de serem sondados, atacados ou comprometidos [Steding-Jessen et al. 2008]. Geralmente um *honeypot* é um *host* Internet que possui

[1] Servidores DNS autoritativos são responsáveis pelos dados de domínios específicos e precisam atender a consultas de quaisquer clientes a respeito desses domínios. Servidores DNS recursivos fazem resolução de nomes (recorrendo a servidores autoritativos) em benefício de clientes (*hosts*), e devem ter seu acesso limitado a um conjunto de clientes autorizados [CERT.br 2016].

 © 2017 Sociedade Brasileira de Computação

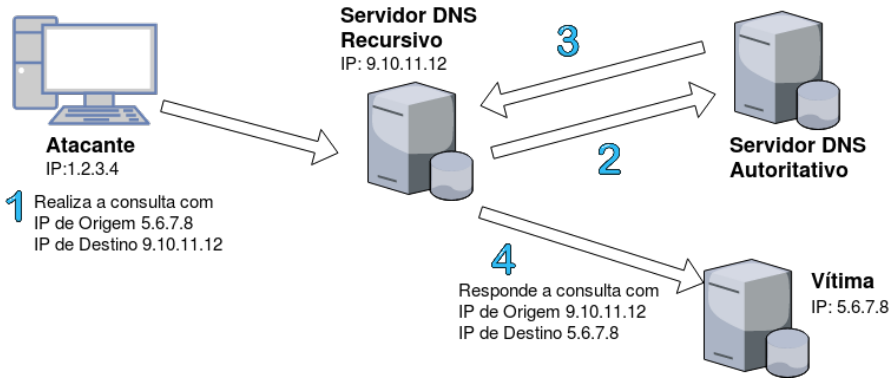


Figura 1. Ataque DRDoS usando o DNS

um endereço IP público (isto é, um endereço IP roteável globalmente), e que hospeda serviços de rede que não são anunciados publicamente. Qualquer interação realizada com um *honeypot* já pode ser considerada suspeita, já que é necessária a realização de varreduras para a descoberta do endereço IP do *honeypot* e de seus serviços.

Este artigo aplica o conceito de *honeypot* à análise de tráfego DNS malicioso. Sua primeira contribuição é a introdução do DNSpot, um *honeypot* DNS que tem por objetivo permitir a interação controlada de atacantes com um servidor DNS recursivo aberto. O diferencial do DNSpot é permitir a observação de tráfego suspeito evitando que esse tráfego cause danos ao próprio servidor ou a terceiros. A segunda contribuição do artigo é uma análise do tráfego coletado pelo DNSpot durante dois períodos, em 2015 (49 dias) e em 2016–2017 (250 dias), mostrando várias características do tráfego de ataque, com foco em ataques DRDoS, e a sua evolução de um período para o outro.

O restante deste artigo está organizado como segue. A Seção 2 discute trabalhos relacionados. A Seção 3 descreve o DNSpot. A Seção 4 apresenta a análise do tráfego coletado pelo *honeypot*, e a Seção 5 conclui o artigo.

2. Trabalhos Relacionados

Estudos envolvendo análise de tráfego DNS vêm sendo conduzidos há vários anos. A primeira linha de investigação foi a caracterização de tráfego DNS sob o prisma de servidores DNS raiz, tanto globais [Danzig et al. 1992, Brownlee et al. 2001, Castro et al. 2010] quanto do Brasil [Barbosa and Pereira 2009]. Outros estudos caracterizam o tráfego DNS sob a ótica de clientes e/ou servidores DNS recursivos [Jung et al. 2002, Gao et al. 2013]. De forma geral, esses trabalhos têm foco em descrever a natureza e a intensidade do tráfego DNS, com pouca ênfase em aspectos de segurança.

Uma outra linha de investigação envolve a detecção de tráfego DNS anômalo e/ou malicioso. [Zdrnja et al. 2007] tem o objetivo de detectar tráfego DNS anômalo, incluindo domínios similares a domínios conhecidos (*typo squatter domains*), que podem ser usados para enganar usuários, nomes *fast-flux*, e nomes associados a campanhas de envio de SPAM. [Perdisci et al. 2009] buscam detectar nomes DNS *fast-flux*. [Zhao et al. 2015] têm o foco na detecção de nomes usados para canais de comunicação e controle (C&C) de *malware*, que são usados para o controle remoto do *malware*, especialmente de *botnets*. O DNSpot introduzido neste trabalho poderia ser adaptado sem

© 2017 Sociedade Brasileira de Computação

grande dificuldade para coletar dados que seriam usados pelos algoritmos propostos nesses trabalhos.

Uma terceira linha de investigação usa varreduras (*scans*) para identificar servidores DNS recursivos abertos na Internet e realiza uma caracterização desses servidores. Em [Takano et al. 2013] são identificados 30 milhões de servidores, dos quais 25 milhões eram recursivos abertos; os servidores são classificados de acordo com sua versão de *software* e domínio Internet. [Kührer et al. 2015] analisam a evolução do número de recursivos abertos encontrados por varreduras (uma queda de 26,8 para 17,8 milhões entre 2014 e 2015) e caracterizam versão de *software*, tipo de *hardware*, e sistema operacional desses servidores. O estudo revela uma incidência significativa de servidores que manipulam respostas DNS, e que essa manipulação pode tanto legítima (*e.g.*, portais captivos) quanto ilegítima (*e.g.*, censura, redirecionamento de anúncios, *phishing* e *malware*).

Alguns trabalhos usam medições para avaliar o fator de amplificação de ataques usando o DNS. [Anagnostopoulos et al. 2013] discute o uso de registros DNSSEC nos ataques DRDoS, e realiza medições com servidores DNS recursivos, onde é observado um fator de amplificação máximo de 44. [van Rijswijk-Deij et al. 2014] realiza um estudo sobre o fator de amplificação obtido com o uso de registros DNSSEC. Foram realizadas consultas em servidores autoritativos responsáveis por 70% dos domínios assinados à época, sendo constatado que os dois fatores de amplificação mais comuns eram 40 e 55, e que o fator máximo era 178,6. [MacFarland et al. 2017] investiga o potencial de ataques DRDoS usando apenas servidores autoritativos como refletores, mostrando que, entre 2013 e 2015, o fator de amplificação médio aumentou de 32,77 para 41,49, e destacando ainda o papel dos registros DNSSEC na amplificação. Esse estudo também mede a adoção de limitação da taxa de resposta (RRL, *response rate limiting*), um mecanismo recomendado para reduzir o impacto de ataques DRDoS [CERT.br 2016]. Enquanto esses trabalhos realizam consultas a servidores DNS recursivos ou autoritativos para mensurar o fator de amplificação (medição ativa), o presente estudo usa uma abordagem de medição passiva, caracterizando o tráfego de ataque associado às consultas DNS recebidas por um *honeypot*, que é efetivamente (e não apenas potencialmente) ofensivo.

O trabalho mais proximamente relacionado é [Fachkha et al. 2015], que apresenta uma análise de 1,44 TB de tráfego DNS enviado a uma *darknet* /13 (mais de 512.000 endereços IP) durante seis meses. Como os endereços IP de uma *darknet* não são utilizados, todo o tráfego destinado a ela é suspeito, podendo ser tráfego de ataque ou devido a erros de configuração. Ao longo do período de monitoração foram observados 134 ataques DDoS, com variados graus de intensidade. Por outro lado, não existem servidores DNS nessa rede, e portanto a análise restringe-se a tráfego enviado às cegas, sem que o atacante saiba se ele será respondido ou não. Como o DNSpot aparenta ser um servidor DNS, os atacantes podem interagir com ele a ponto de acreditar que ele produzirá as respostas necessárias para ataques DDoS. Isso torna o tráfego recebido pelo DNSpot mais representativo que tráfego enviado a endereços não usados em uma *darknet*, cuja eficácia é desconhecida. Por outro lado, o DNSpot oferece um único ponto de vista, o que dificulta a avaliação da real extensão dos ataques DDoS que são observados pelo *honeypot*.

Fora do contexto acadêmico, [Tanasi 2014] implementou um *honeypot* DNS que compartilha similaridades com o DNSpot, mas que registra apenas as consultas recebidas, não as respostas. [Van Impe 2015] implementou um servidor DNS que envia uma

 © 2017 Sociedade Brasileira de Computação

resposta fixa para qualquer consulta, e analisou o tráfego recebido entre o final de dezembro de 2014 e meados de fevereiro de 2015, igualmente desconsiderando o tráfego de resposta. Alguns resultados são consistentes com os observados no DNSpot. Contudo, é inviável afirmar se as divergências devem-se a mudanças no tráfego malicioso entre os períodos monitorados por Van Impe e os descritos neste trabalho ou ao fato do servidor de Van Impe fabricar respostas fixas (e inconsistentes) para qualquer consulta, o que pode afastar atacantes que testam a funcionalidade de servidores recursivos abertos antes de usá-los em ataques.

3. DNSpot

O DNSpot é um *honeypot* projetado especificamente para monitorar e analisar o tráfego DNS. Seu objetivo é permitir a observação das interações de usuários potencialmente maliciosos com servidores DNS recursivos. A arquitetura do DNSpot pode ser observada na Figura 2. Ele possui um *proxy* que escuta na porta 53/UDP, que é a porta padrão do serviço DNS. Ao receber uma consulta, o *proxy* armazena a consulta em um banco de dados e logo em seguida a repassa a um servidor DNS recursivo real. Esse servidor real, que aceita apenas consultas originadas na própria máquina, obtém uma resposta do seu próprio *cache* ou interagindo com servidores autoritativos na Internet. Por último, o *proxy* recebe a resposta do servidor recursivo, armazena no banco de dados e encaminha para o cliente que enviou a consulta.

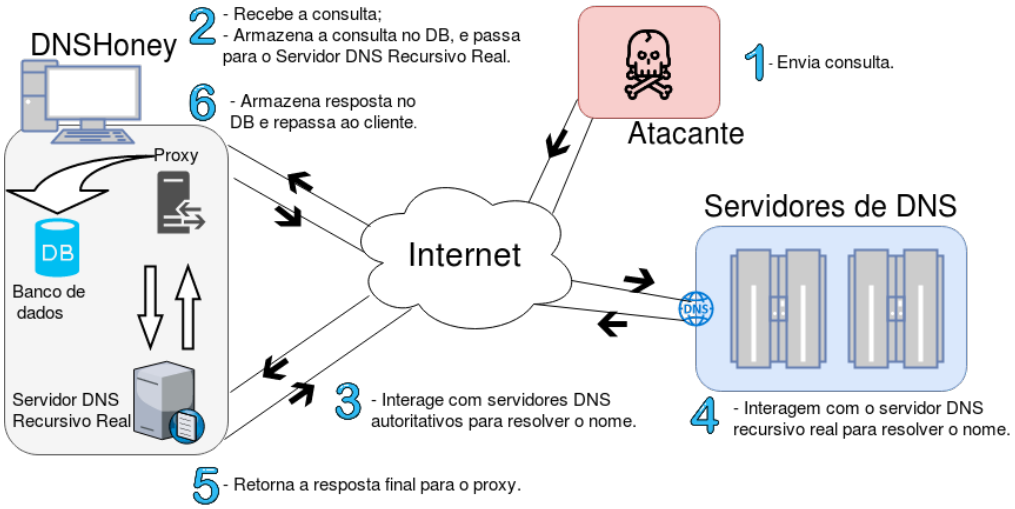


Figura 2. Arquitetura do DNSpot

Ocasionalmente, o DNSpot retorna uma mensagem falsa de erro (*ServFail*, que sinaliza uma falha inespecífica do servidor) para o cliente; a frequência dessas mensagens é configurável. O objetivo é simular, perante um atacante humano (não uma ferramenta automatizada), um servidor DNS inconfiável, tentando não levantar suspeitas caso o DNSpot seja desligado ou não consiga processar algumas consultas.

Para reduzir o impacto caso o DNSpot seja usado como refletor em um ataque DRDoS, é estabelecido um limite diário (também configurável) para o número de consultas atendidas para cada endereço IP de origem. Caso esse limite seja atingido, o *proxy* deixa de enviar respostas para consultas do mesmo endereço, até o dia seguinte. Essa contagem de consultas por endereço IP é zerada diariamente.

© 2017 Sociedade Brasileira de Computação

4. Análise de Tráfego

Esta seção apresenta uma análise do tráfego coletado com o DNSpot, abrangendo implantação (Seção 4.1), estatísticas gerais do tráfego (Seção 4.2), caracterização de ataques DoS (Seção 4.3), resultados notáveis (Seção 4.4), e discussão geral (Seção 4.5).

4.1. Implementação do DNSpot e Coleta de Dados

O DNSpot foi implementado em Python, usando Unbound[2] para o servidor DNS real e SQLite[3] para o banco de dados, em plataforma OpenBSD.[4] O *honeypot* foi instalado na rede da Universidade, coletando dados de forma quase ininterrupta durante os períodos mostrados na Tabela 1. Em ambas as coletas houve alguns períodos de indisponibilidade, causados por falta de energia, interrupção do acesso à Internet ou travamentos do sistema; a indisponibilidade total é estimada em 0,5% a 1% dos 300,1 dias de coleta.

O DNSpot foi configurado para responder a 20% das requisições com uma mensagem de falha do servidor, de modo a aparentar confiabilidade limitada. O número de consultas diárias por endereço IP foi fixado em 30; as consultas excedentes são processadas e armazenadas no banco de dados, mas as respostas não são enviadas. Esses parâmetros foram estabelecidos empiricamente, visando a limitar o tráfego gerado em ataques DRDoS e ainda permitir que atacantes humanos interajam com o *honeypot*; o segundo objetivo não seria alcançado caso o DNSpot não enviasse nenhuma resposta.

Existem diversos projetos que realizam varreduras para procurar servidores DNS recursivos abertos na Internet.[5] Essas varreduras podem ser identificadas pelos nomes ou sufixos consultados, e o DNSpot foi configurado para ignorar tais consultas.

Dataset	Início	Fim	Total (dias)
DS1	09/09/2015 07:57	28/10/2015 22:29	49,6
DS2	17/09/2016 08:00	25/05/2017 20:47	250,5

Tabela 1. *Datasets* coletados com o DNSpot

4.2. Estatísticas de Tráfego

A Tabela 2 resume o tráfego processado pelo DNSpot nos dois períodos de coleta. Foram mais de 4 milhões de requisições no *dataset* DS1 e mais de 32 milhões em DS2, totalizando 36,4 milhões de requisições processadas. Destaca-se o número pequeno de consultas respondidas (5,7% do total) e o grande número de consultas ignoradas (98,5% das não respondidas, ou 92,8% do total), o que reflete o funcionamento do mecanismo de limitação diária de consultas por endereço IP. A fração de consultas respondidas foi menor em DS2 do que em DS1, o que se explica tanto pelo aumento de erros (requisições malformadas) quanto pela maior intensidade e duração dos ataques DoS (discutidas na Seção 4.3). A taxa média de processamento de requisições passou de 81.353,02 transações por dia (0,96 tps) em DS1 para 129.955,61 em DS2 (1,50 tps), o que representa um incremento de 59,7% na intensidade de tráfego processado pelo DNSpot.

[2] `https://www.unbound.net/`.
[3] `http://www.sqlite.org/`.
[4] `https://www.openbsd.org/`.
[5] Exemplos: `http://dnsresearch.cymru.com` e `http://openresolverproject.org`.

 © 2017 Sociedade Brasileira de Computação

Transações	DS1		DS2		Total	
	Quantidade	%	Quantidade	%	Quantidade	%
Respondidas	488.289	12,1	1.600.386	4,9	2.088.675	5,7
– Válidas	391.050	*80,1*	1.280.425	*80,0*	1.671.475	*80,0*
– *ServFail* falso	97.249	*19,9*	319.961	*20,0*	417.210	*20,0*
Não respondidas	3.547.306	87,9	30.758.542	95,1	34.305.848	94,3
– Ignoradas	3.544.876	*99,9*	30.243.241	*98,3*	33.788.117	*98,5*
– Erros	2.370	*0,1*	515.301	*1,7*	517.671	*1,5*
Total	4.035.605	100,0	32.358.928	100,0	36.394.533	100,0

Tabela 2. Tráfego DNS processado pelo DNSpot. Porcentagens em itálico representam a subdivisão dentro de uma categoria.

A Tabela 3 apresenta o volume de tráfego processado pelo DNSpot. Ao todo foram processados 6.801,1 MB de tráfego pelo DNSpot, sendo 1.361,5 MB (20,0%) de consultas e 5.440,1 MB (80,0%) de respostas. Caso todas as consultas tivessem sido processadas, o tráfego esperado seria de quase 50 GB; restringir o número diário de consultas por IP levou a uma redução de 88,7% no tráfego total, mais uma vez comprovando a eficácia do mecanismo para limitar o tráfego gerado pelo DNSpot. A proporção de consultas aumentou de 10,6% em DS1 para 22,8% em DS2, levando a uma menor redução do tráfego de resposta no segundo *dataset*.

Tipo	DS1		DS2		Total	
	MB	%	MB	%	MB	%
Tráfego processado	1.560,1	100,0	5.241,5	100,0	6.801,6	100,0
– Consultas	165,1	*10,6*	1.196,4	*22,8*	1.361,5	*20,0*
– Respostas	1.395,0	*89,4*	4.045,1	*77,2*	5.440,1	*80,0*
Tráfego esperado	14.775,6	–	34.775,9	–	49.551,5	–
Respostas	14.610,4	–	33.579,5	–	48.189,9	–
Redução de tráfego de resposta	13.215,4	90,5	29.534,4	84,9	42.749,8	88,7

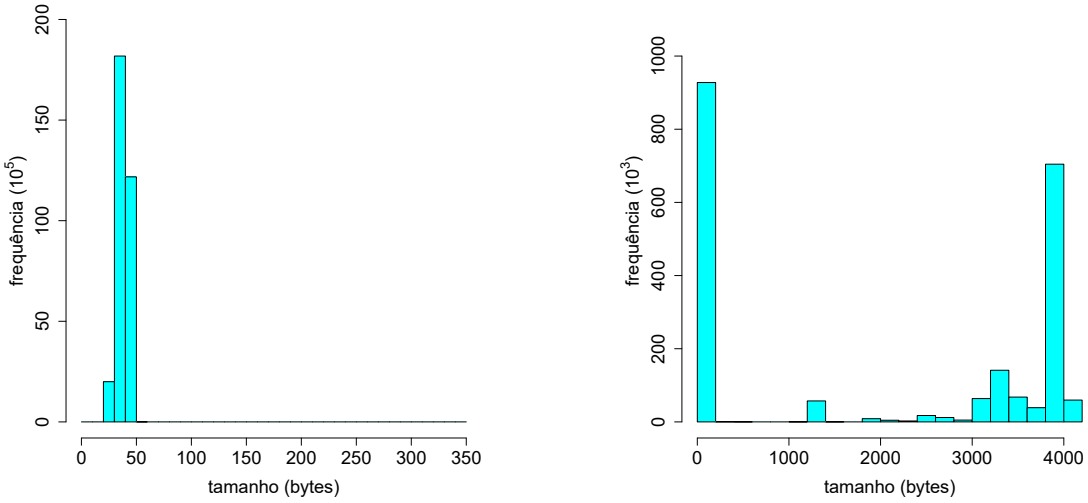
Tabela 3. Volume de tráfego processado e esperado.

Cada consulta especifica um registro de recurso (RR), que é a unidade básica de informação do DNS [Mockapetris 1987]. Cada RR é caracterizado pelo seu nome, tipo e classe, e possui um conteúdo que depende do tipo e da classe. A classe padrão é IN (Internet), e será implicitamente assumida no restante do texto. É possível ter vários RRs com o mesmo nome e tipo (*e.g.*, múltiplos registros A, que armazenam endereços IP), assim como vários RRs para o mesmo nome mas com tipos distintos. A ampla maioria das consultas observadas pelo DNSpot é por RRs com tipo ANY: 94,6% no total, sendo 99,2% em DS1 e 94% em DS2. O tipo ANY pode ser usado apenas em consultas, e recupera todos os RRs (de qualquer tipo) associados a um dado nome. Isso pode ser feito *(i)* para depurar domínios DNS, *(ii)* para obter múltiplas informações com uma única consulta, *(iii)* para descobrir potenciais alvos de ataques, ou *(iv)* para produzir respostas grandes a partir de consultas pequenas, amplificando o tráfego (este último caso é o mais pertinente a este estudo). Em função dos riscos associados, as especificações do DNS estão sendo alteradas para recomendar a limitação de respostas a consultas ANY [Abley et al. 2017]. Em contraste com o DNSpot, [Van Impe 2015] observou 51,3% de consultas ANY e 47,9% de consultas A; uma possível explicação é que, como seu *honeypot* enviava apenas respostas

© 2017 Sociedade Brasileira de Computação

falsas, ele pode ter sido menos usado para ataques DRDoS.

A distribuição dos tamanhos de consultas e respostas no *dataset* DS1 é fortemente distorcida pela dominância de um único registro de recurso RR, `hehehey.ru ANY`, presente em 97% das consultas (com tamanho de 39 bytes) e 90,4% das respostas (3.850 bytes). Em DS2 a variabilidade é maior: o tamanho das consultas possui uma distribuição assimétrica positiva (concentração de valores pequenos), enquanto o tamanho das respostas possui uma distribuição bimodal, conforme pode ser visto na Figura 3. 99,999% das consultas têm até 50 bytes, enquanto que, dentre as respostas, 43,9% têm até 100 bytes, 54,9% têm mais de 3.000 bytes, e 36,4% têm mais de 3.800 bytes. As respostas grandes podem ser associadas a tráfego DRDoS, enquanto que as pequenas devem-se a uma conjugação de *(i)* ataques DRDoS malsucedidos (consultas por nomes inexistentes ou que geram respostas pequenas), *(ii)* tráfego de sondagem (*probes*), e *(iii)* uso do DNSpot como servidor recursivo por usuários finais (que será discutido na Seção 4.4).



(a) Consultas (b) Respostas

Figura 3. Distribuição dos tamanhos de consultas e respostas no *dataset* DS2.

A Tabela 4 resume a distribuição geográfica dos endereços IP que originaram consultas para o DNSpot.[6] Essa análise não distingue entre clientes que realizam consultas e endereços de vítimas de ataques DRDoS, os quais usam IPs de origem forjados. Comparando os *datasets* DS1 e DS2, percebe-se que no segundo há uma maior diversidade, tanto em número de países quanto em concentração de requisições: o número de países aumentou de 73 para 161, e a proporção de endereços IP dos cinco países mais frequentes caiu de 84,0% para 36,5%. China e EUA aparecem nas duas primeiras posições em ambos *datasets*, mas a proporção desses dois países caiu de 57,2% para 24,1%. Os resultados de geolocalização para o *dataset* DS1 são semelhantes aos de [Van Impe 2015], o que faz sentido devido aos períodos de monitoração serem mais próximos.

4.3. Ataques DoS

É esperado que uma parte do tráfego direcionado a um servidor DNS recursivo aberto seja devido a ataques DRDoS em que o servidor esteja sendo usado como refletor. Nesta

[6]Os dados de geolocalização foram obtidos de `http://freegeoip.net/`.

 © 2017 Sociedade Brasileira de Computação

	DS1			DS2		
Posição	País	IPs distintos	%	País	IPs distintos	%
1	China	1.287	30,0	China	28.705	15,6
2	Estados Unidos	1.164	27,2	Estados Unidos	15.613	8,5
3	Rússia	759	17,7	Brasil	9.838	5,3
4	Alemanha	297	6,9	Coreia do Sul	6.815	3,7
5	Canadá	94	2,2	Japão	6.398	3,5
	outros	686	16,0	outros	117.195	63,5
Total	*73 países*	4.287	100,0	*161 países*	184.564	100,0

Tabela 4. Top 5 países de origem das consultas ao DNSpot.

seção são caracterizados os ataques DRDoS em que o DNSpot foi usado como refletor. Para essa caracterização, foi adotada a seguinte definição de ataque DoS:

> *Um ataque DoS é formado por um conjunto com no mínimo 5 consultas com o mesmo IP de origem e com espaçamento máximo de 60 segundos entre consultas consecutivas, e pelas respostas a essas consultas.*

Essa definição foi estabelecida com base em uma análise preliminar do tráfego do DNSpot. Embora cinco consultas representem no máximo 20 KB de tráfego, esse número mínimo de consultas foi estabelecido considerando que o DNSpot esteja sendo usado como um dos vários refletores envolvidos em um ataque DRDoS, e não como um servidor recursivo regular. Cabe destacar ainda que, como ataques DRDoS usam endereços IP de origem forjados, do ponto de vista do DNSpot não é possível distinguir entre ataques DRDoS em que o *honeypot* é usado como refletor e ataques DoS diretos contra o *honeypot*; estes são bem menos prováveis, já que o sistema não hospeda nenhum serviço anunciado publicamente.

A Tabela 5 mostra o número de ataques DoS segundo a definição proposta, e as quantidades de endereços IP de origem, RRs e requisições envolvidos em tais ataques. Observa-se uma redução tanto na frequência de ataques quanto nas proporções de endereços IP, RRs e requisições envolvidos em ataques DoS. Foram identificados dois fatores que contribuem para essa redução. De um lado, no *dataset* DS1 houve uma concentração de requisições – os 97% de consultas por `hehehey.ru ANY` –, o que indica que o DNSpot foi intensamente usado como refletor nesse período, provavelmente por uma única ferramenta automatizada. De outro lado, em DS2 verificou-se a partir de maio de 2017 um aumento significativo da proporção de requisições que não se encaixam na definição de ataque, o que ajuda a explicar as quedas mais expressivas, de endereços IP e RRs envolvidos em ataques. Essa mudança de característica é discutida na Seção 4.4; como ela é recente, é preciso acompanhar sua evolução para ver se é algo efêmero ou mais duradouro.

Em ataques DRDoS, o fator de amplificação mede quanto tráfego de resposta um refletor gera em função de um dado tráfego de entrada. Nos ataques usando o DNS, esse fator é dado pela razão entre o tamanho de respostas e o tamanho das consultas correspondentes. A Tabela 6 mostra os fatores de amplificação observados nos dois *datasets*. O fator máximo foi comparável nas duas coletas, mas o fator médio foi menor em DS2. Essa redução deve-se a um número expressivo de consultas em DS2 (mais de 2 milhões, ou 6,5% do total) para RRs com fator de amplificação inferior a 10, que pode ser explicada por nomes equivocados (Seção 4.4) e por medidas de contenção do tamanho de

© 2017 Sociedade Brasileira de Computação

	DS1		DS2	
N° de ataques	7.940		23.788	
Ataques/dia (média)	160,1		95,0	
Métricas	Total	Envolvidos em DoS	Total	Envolvidos em DoS
IPs	4.287	3.499 (81,6%)	184.564	23.745 (12,9%)
RRs	136	87 (64,0%)	4.982	840 (16,8%)
N° de requisições	4.035.605	4.032.778 (99,9%)	32.358.928	30.661.228 (94,7%)

Tabela 5. Métricas envolvidas em ataques DoS contra o DNSpot

respostas, como mudanças no processamento de consultas com tipo ANY em servidores DNS autoritativos [Abley et al. 2017]. No geral, os maiores fatores de amplificação foram observados para consultas ANY cujas respostas contêm diversos registros DNSSEC, corroborando o observado em [Anagnostopoulos et al. 2013, van Rijswijk-Deij et al. 2014, MacFarland et al. 2017].

Fator de amplificação	DS1	DS2
Médio	96,3	74,1
Máximo	110,7	103,6

Tabela 6. Fator de amplificação dos ataques DoS

A Figura 4 mostra as distribuições empíricas da duração dos ataques nos dois *datasets*. Em DS1, 95% dos ataques duraram até 538 s (menos de 9 min). Em DS2, 50% dos ataques duraram até 480 s (8 min), e 25% duraram 1.140 s (19 min) ou mais. Isso significa que, embora a frequência de ataques DoS tenha caído, a duração dos ataques foi maior. O número de requisições por ataque também aumentou bastante, conforme mostrado na Tabela 7; a mediana foi 17,1 vezes maior, e as demais medidas aumentaram cerca de 12 vezes. Em ambos os *datasets*, a ampla maioria dos endereços IP esteve envolvida (i.e., foi vítima) em apenas um ataque, sendo que os máximos observados foram de 73 ataques para um único IP em DS1 e 41 ataques em DS2.

Dataset	média	mediana	3° quartil	95° percentil	máximo
DS1	507,9	132,0	402,0	2.100,2	25.363
DS2	6.029,7	2.264,0	5.024,0	25.594,4	203.474

Tabela 7. Estatísticas de requisições por ataque DoS

4.4. Resultados Notáveis

Durante a operação do DNSpot, foram observados alguns resultados notáveis do ponto de vista qualitativo, que serão descritos a seguir.

Varreduras UDP e SIP. Foram recebidas requisições malformadas, que não respeitavam o formato das mensagens DNS. Algumas dessas requisições tinham conteúdo vazio, e foram classificadas como varreduras de portas UDP, como as realizadas pelo Nmap[7]. Também foram identificadas varreduras do protocolo SIP (*Session Initiation Protocol*),

[7] `https://nmap.org/`.

 © 2017 Sociedade Brasileira de Computação

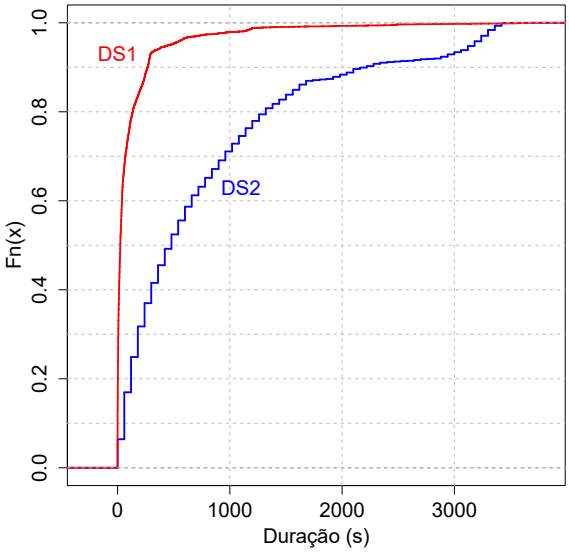


Figura 4. Distribuições empíricas da duração de ataques

que buscavam terminais e servidores de telefonia IP usando a porta 53/UDP, sendo que a porta padrão para sinalização SIP é 5060/UDP.

Domínios projetados para amplificação. Foram encontrados diversos domínios contendo RRs que não possuem nenhum significado ou utilidade para uma consulta normal DNS, cujo único propósito é gerar respostas grandes (próximas a 4 KB), úteis para ataques DRDoS, algo que já havia sido descrito na literatura (*e.g.*, [Anagnostopoulos et al. 2013]). No DNSpot foram observados nomes com mais de 250 registros A pertencentes a uma mesma sub-rede, e nomes com mais de 30 registros TXT cujo conteúdo é a letra "x" repetida 99 vezes combinada com um sufixo numérico. Alguns desses nomes pertencem a domínios distintos mas possuem estrutura idêntica, com os mesmos registros SOA, NS, MX e A associados ao nome, o que indica que foram criados pela mesma pessoa ou grupo.

Desaparecimento e redução no tamanho de domínios. Durante os períodos de observação, foram constatadas mudanças no conteúdo dos domínios consultados que levaram à redução no fator de amplificação. No *dataset* DS1, foi verificado o desaparecimento do quarto domínio com mais consultas, que teve 13.455 requisições. Inicialmente, esse domínio gerou 2.970 respostas com 3.875 bytes, e posteriormente houve 2.316 respostas com 96 bytes, que acusavam domínio inexistente; aparentemente o domínio continuou sendo usado por ferramentas automatizadas para ataques DRDoS mesmo depois de seu desaparecimento (que ocorreu pela expiração do registro do domínio). Em DS2 desapareceram 17 RRs. Diferente do que ocorreu com DS1, em DS2 foram registradas poucas consultas após o desaparecimento dos RRs, o que sugere que as ferramentas de ataque passaram a atualizar com maior frequência suas listas de RRs usados em ataques DRDoS.

© 2017 Sociedade Brasileira de Computação

Consultas por nomes equivocados. Em ambos os *datasets* foram observadas quantidades expressivas de consultas por domínios de primeiro nível (DPNs) obviamente inexistentes, como `.3858` (13,5 mil consultas em DS1) e `.pkt` (1,74 milhões de consultas em DS2). Foram também registradas consultas com tipos aparentemente trocados; por exemplo, em DS1 houve 6,5 mil requisições para um registro TXT inexistente, sendo que uma consulta pelo mesmo nome com tipo ANY gerava uma resposta de 4.071 bytes. Acredita-se que essas consultas por nomes equivocados sejam fruto de erro na configuração de ferramentas usadas em ataques DRDoS. Incidências significativas de consultas por DPNs inexistentes também foram observadas em servidores DNS raiz [Castro et al. 2010], tendo sido atribuídas a roteadores domésticos mal configurados.

Consultas de usuários finais. No *dataset* DS2 foi observada uma quantidade expressiva de consultas por nomes tipicamente associados a usuários finais, incluindo *sites* populares (*e.g.*, `google.com`, `facebook.com`, `amazon.com`) e nomes usados por ferramentas anti-*malware* (*e.g.*, `avqs.mcafee.com`). Essas consultas, cujo fator de amplificação é baixo (< 10), sugerem que usuários estão usando o DNSpot como resolvedor DNS padrão. Elas tiveram um aumento significativo em maio de 2017, e estão associadas a um incremento no número de endereços IP distintos que interagiram com o *honeypot*. Também foram observadas consultas por nomes usados para descoberta de serviços de rede (DNS-SD, *DNS-Based Service Discovery*) [Cheshire and Krochmal 2013], no caso com endereços de redes privadas (192.168.*.*). Essas consultas podem ser reflexo do uso do DNSpot como resolvedor regular, embora também possam tratar-se de varreduras usando DNS-SD em busca de alvos para ataques [Atlasis 2017].

4.5. Discussão dos Resultados

Uma análise dos dados coletados pelo DNSpot, considerando de forma conjunta as diferentes métricas observadas, permite chegar a algumas conclusões importantes:

- Ataques DRDoS constituem o principal abuso envolvendo servidores DNS recursivos abertos. A escolha adequada dos nomes consultados permite obter um fator de amplificação superior a 100, consideravelmente maior que os fatores típicos reportados para o DNS, entre 28 e 54 [CERT.br 2016]. Nesse sentido, o mecanismo de limitação diária de consultas, embora pouco sofisticado, foi eficaz em restringir o tráfego de ataque obtido com o abuso do DNSpot a 11,3% do volume pretendido.
- O volume de requisições maliciosas, especialmente considerando o fato de ser um servidor não anunciado publicamente, é significativo: considerando as duas coletas, o DNSpot recebeu em média 1,4 requisições por segundo, com um tráfego médio diário potencial de 165,1 MB. O DNSpot começou a receber as primeiras requisições segundos após o início do seu funcionamento, e menos de 28 h depois já estava sendo usado em ataques DRDoS.
- Uma análise da evolução dos ataques DoS entre as duas coletas revela uma diminuição na frequência dos ataques, contrabalançada por um aumento significativo na sua duração e intensidade. Também foi possível constatar uma redução no fator de amplificação médio dos ataques, embora não seja possível afirmar se tal redução foi provocada por uma mudança na configuração de servidores DNS auto-

 © 2017 Sociedade Brasileira de Computação

ritativos (especialmente o tratamento de consultas do tipo ANY) ou por equívocos cometidos pelos atacantes.

5. Conclusão

Este artigo introduz o DNSpot, um *honeypot* específico para DNS que tem por objetivo propiciar a monitoração de requisições DNS sob o prisma de servidores recursivos abertos. O artigo também apresenta uma análise de dados coletados do DNSpot durante dois períodos, em 2015 (49 dias) e 2016–2017 (250 dias), no qual foram processadas 36,4 milhões de requisições. O estudo demonstrou a incidência de ataques DRDoS usando um servidor DNS recursivo aberto, e o alto fator de amplificação oferecido pelo DNS. Além disso, foram descobertos aspectos pouco conhecidos, como a existência de domínios projetados para amplificação. A comparação entre os dois períodos de coleta revelou algumas mudanças relevantes no tráfego, que incluem:

- aumento na intensidade do tráfego direcionado ao DNSpot;
- aumento na duração e no volume de tráfego em ataques DDoS que usam o DNSpot como refletor;
- evidências de uso do DNSpot como servidor DNS recursivo regular por parte de usuários finais ou em nome destes.

Para a continuidade deste trabalho, algumas perspectivas podem ser apontadas, como manter a coleta de dados do DNSpot por períodos ainda maiores, conjugando análises de longo e curto prazo, e investigar como múltiplas instâncias do DNSpot, em diferentes pontos da Internet, podem ser usadas em um sistema distribuído de monitoramento e detecção de ataques ao DNS.

Referências

[Abley et al. 2017] Abley, J., Gudmundsson, O., and Majkowski, M. (2017). Providing minimal-sized responses to DNS queries that have QTYPE=ANY. IETF Draft draft-ietf-dnsop-refuse-any-04 (Proposed standard). Disponível em `https://tools.ietf.org/html/draft-ietf-dnsop-refuse-any-04`.

[Akamai 2017] Akamai (2017). Q1 2017 state of the Internet/security report. Technical report. Disponível em `http://www.akamai.com/`.

[Anagnostopoulos et al. 2013] Anagnostopoulos, M., Kambourakis, G., Kopanos, P., Louloudakis, G., and Gritzalis, S. (2013). DNS amplification attack revisited. *Computers & Security*, 39:475–485.

[Arbor 2017] Arbor (2017). Worldwide infrastructure security report, vol. XII. Technical report, Arbor Networks. Disponível em `http://www.arbornetworks.com/`.

[Atlasis 2017] Atlasis, A. (2017). An attack-in-depth analysis of multicast DNS and DNS service discovery. In *Hack in the Box*, Amsterdam. Disponível em `http://tinyurl.com/ycybxp59`.

[Barbosa and Pereira 2009] Barbosa, K. R. and Pereira, E. S. J. (2009). Análise passiva do tráfego DNS da Internet brasileira. In *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*, pages 203–216.

[Brownlee et al. 2001] Brownlee, N., Claffy, K. C., and Nemeth, E. (2001). DNS measurements at a root server. In *IEEE Global Telecommunications Conference (GLOBECOM)*, San Antonio, TX.

[Castro et al. 2010] Castro, S., Zhang, M., John, W., Wessels, D., and Claffy, K. C. (2010). Understanding and preparing for DNS evolution. In *Traffic Monitoring and Analysis Workshop (TMA)*, pages 1–6, Zurich, Switzerland.

© 2017 Sociedade Brasileira de Computação

[CERT.br 2016] CERT.br (2016). Recomendações para melhorar o cenário de ataques distribuídos de negação de serviço (DDoS). Disponível em `http://www.cert.br/docs/whitepapers/ddos/`.

[Cheshire and Krochmal 2013] Cheshire, S. and Krochmal, M. (2013). DNS-based service discovery. RFC 6763.

[Conrad 2012] Conrad, D. (2012). Towards improving DNS security, stability, and resiliency. Technical report, Internet Society.

[Danzig et al. 1992] Danzig, P. B., Obraczka, K., and Kumar, A. (1992). An analysis of wide-area name server traffic: a study of the Internet Domain Name System. *ACM SIGCOMM Computer Communication Review*, 22(4):281–292.

[Fachkha et al. 2015] Fachkha, C., Bou-Harb, E., and Debbabi, M. (2015). Inferring distributed reflection denial of service attacks from darknet. *Computer Communications*, 62:59–71.

[Gao et al. 2013] Gao, H., Yegneswaran, V., Chen, Y., Porras, P., Ghosh, S., Jiang, J., and Duan, H. (2013). An empirical reexamination of global DNS behavior. *ACM SIGCOMM Computer Communication Review*, 43(4):267–278.

[Jung et al. 2002] Jung, J., Sit, E., Balakrishnan, H., and Morris, R. (2002). DNS performance and the effectiveness of caching. *IEEE/ACM Transactions on Networking*, 10(5):589–603.

[Kührer et al. 2015] Kührer, M., Hupperich, T., Bushart, J., Rossow, C., and Holz, T. (2015). Going wild: Large-scale classification of open DNS resolvers. In *Internet Measurement Conference (IMC)*, pages 355–368, Tokyo, Japan.

[MacFarland et al. 2017] MacFarland, D. C., Shue, C. A., and Kalafut, A. J. (2017). The best bang for the byte: Characterizing the potential of DNS amplification attacks. *Computer Networks*, 116:12–21.

[Mockapetris 1987] Mockapetris, P. (1987). Domain names – concepts and facilities. RFC 1034.

[Perdisci et al. 2009] Perdisci, R., Corona, I., Dagon, D., and Lee, W. (2009). Detecting malicious flux service networks through passive analysis of recursive DNS traces. In *Annual Computer Security Applications Conference (ACSAC)*, pages 311–320, Honolulu, HI.

[Steding-Jessen et al. 2008] Steding-Jessen, K., Vijaykumar, N. L., and Montes Filho, A. (2008). Using low-interaction honeypots to study the abuse of open proxies to send Spam. *InfoComp*, 7(1):44–52.

[Takano et al. 2013] Takano, Y., Ando, R., Takahashi, T., Uda, S., and Inoue, T. (2013). A measurement study of open resolvers and DNS server version. In *Internet Conference (IEICE)*.

[Tanasi 2014] Tanasi, A. (2014). Homemade custom interaction DNS honeypot. Disponível em `http://tinyurl.com/y955ohns`.

[Van Impe 2015] Van Impe, K. (2015). Analyzing queries on a honeypot name server for better DNS log quality. *Security Intelligence*. Disponível em `http://tinyurl.com/ybsze9zd`.

[van Rijswijk-Deij et al. 2014] van Rijswijk-Deij, R., Sperotto, A., and Pras, A. (2014). DNSSEC and its potential for DDoS attacks: A comprehensive measurement study. In *Internet Measurement Conference (IMC)*, pages 449–460, Vancouver, BC, Canada.

[Zdrnja et al. 2007] Zdrnja, B., Brownlee, N., and Wessels, D. (2007). Passive monitoring of DNS anomalies. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pages 129–139, Lucerne, Switzerland.

[Zhao et al. 2015] Zhao, G., Xu, K., Xu, L., and Wu, B. (2015). Detecting APT malware infections based on malicious DNS and traffic analysis. *IEEE Access*, 3:1132–1142.

 © 2017 Sociedade Brasileira de Computação

Combatendo vulnerabilidades em programas P4 com verificação baseada em asserções

Lucas Freire, Miguel Neves, Alberto Schaeffer-Filho, Marinho Barcellos[1]

[1] Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{lmfreire, mcneves, alberto, marinho}@inf.ufrgs.br

Abstract. *Current trends in SDN extend the network programmability to the data plane through the usage of programming languages like P4. In this context, the chances of introducing errors, and consequently software vulnerabilities in the network significantly increases, since packets can be processed in a general fashion. Existing data plane verification mechanisms are incapable of modeling P4 programs or present severe restrictions in the set of modeled properties. To circumvent these limitations and make programmable data planes more secure, we present a P4 program verification approach based on the usage of assertions and symbolic execution. First, P4 programs are annotated with assertions expressing general correctness and security properties. Then, the annotated programs are transformed in C code and all their possible paths are symbolically executed to verify the inserted assertions validity. Results show that it is possible to prove an expressive set of properties in the order of seconds using the proposed technique.*

Resumo. *Tendências recentes em SDN estendem a programabilidade da rede ao plano de dados através do uso de linguagens de programação como P4. Nesse contexto, a chance de se introduzir erros e, por consequência, vulnerabilidades de software na rede aumenta significativamente já que pacotes passam a ser processados de maneira genérica. Mecanismos de verificação de planos de dados existentes são incapazes de modelar programas P4 ou apresentam grandes restrições no conjunto de propriedades modeladas. Para contornar essas limitações e tornar planos de dados programáveis mais seguros, apresentamos um método de verificação de programas P4 baseado no uso de asserções e execução simbólica. Primeiro, programas P4 são anotados com asserções que expressam propriedades genéricas de corretude e segurança. Em seguida, os programas anotados são transformados em código C e todos seus possíveis caminhos são executados simbolicamente a fim de verificar a validade das asserções inseridas. Resultados mostram que é possível provar um conjunto expressivo de propriedades na ordem de segundos utilizando a técnica proposta.*

1. Introdução

A programabilidade do plano de dados permite que operadores implantem novos protocolos de comunicação e desenvolvam serviços de rede com agilidade. Através do uso de linguagens de programação como P4, é possível especificar em poucas instruções quais e

 © 2017 Sociedade Brasileira de Computação

como cabeçalhos de pacotes serão manipulados pelos diferentes dispositivos de encaminhamento da infraestrutura. Apesar da flexibilidade proporcionada por esse paradigma, uma série de desafios precisam ser tratados antes da sua ampla adoção.

Um dos principais desafios enfrentados diz respeito à segurança. Se por um lado P4 introduz um novo eixo de programação (o plano de dados da rede), por outro também eleva as chances de ocorrerem bugs causados por implementações incorretas de protocolos. Tais bugs podem facilmente se transformar em vulnerabilidades caso eles possam ser explorados no sentido de violar políticas de segurança da rede. Uma forma de combater esse problema é testando se a rede satisfaz propriedades de interesse utilizando métodos de verificação. Diversos mecanismos foram propostos nesse sentido [Son et al. 2013, Dobrescu and Argyraki 2014, Lopes et al. 2015, Lopes et al. 2016]. No entanto, nenhum deles é capaz de verificar propriedades de segurança em programas P4.

Com o objetivo de habilitar a verificação de propriedades genéricas de corretude e segurança em programas P4 (p.ex. isolamento, integridade de cabeçalhos e ausência de laços), nós propomos um mecanismo baseado em asserções e execução simbólica. Primeiro, o programa P4 é anotado com asserções que expressam propriedades de interesse. O programa anotado é então traduzido para um modelo expresso em linguagem C e uma máquina de execução simbólica percorre todos os possíveis caminhos de execução do modelo gerado. Nosso mecanismo é capaz de provar que uma propriedade é satisfeita verificando se nenhum caminho de execução viola as asserções que a expressam.

Nós prototipamos o mecanismo proposto utilizando a ferramenta Klee [Cadar et al. 2008] e o compilador de referência da linguagem P4, disponibilizado pelo *P4 Language Consortium*[1]. Resultados mostram que o mecanismo proposto é capaz de verificar propriedades de segurança (p.ex. integridade de cabeçalhos e ausência de amplificação de tráfego) na ordem de segundos para programas com até 15 tabelas. Além disso, um número razoável de asserções (cerca de 10) pode ser analizado sem impactar significativamente o tempo de verificação. Em resumo, esse artigo apresenta as seguintes contribuições: i) uma linguagem para especificação de propriedades de corretude e segurança (na forma de asserções) envolvendo programas P4; ii) um mecanismo de verificação de propriedades baseado em asserções e execução simbólica; e iii) a avaliação experimental do mecanismo proposto. Esse é o primeiro trabalho a abordar segurança em planos de dados programáveis.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta a linguagem de programação P4 e os principais conceitos relacionados a planos de dados programáveis. Na sequência, a Seção 3 traz uma descrição detalhada da solução de verificação proposta. A Seção 4 descreve a avaliação experimental realizada, enquanto uma visão geral do estado-da-arte é apresentada na Seção 5. Por fim, a Seção 6 expõe as principais conclusões e trabalhos futuros.

2. Linguagem P4 e Planos de Dados Programáveis

A linguagem de programação P4[2], proposta por [Bosshart et al. 2014], tem o objetivo de possibilitar a programação do plano de dados de dispositivos de rede (isto é, switches e

[1] p4.org

[2] Consideramos neste trabalho a versão $P4_{16}$ por ser a mais recente

 © 2017 Sociedade Brasileira de Computação

roteadores) de maneira simples e independente de arquitetura. Dessa forma, novos protocolos de comunicação podem ser rapidamente implantados na rede com apenas algumas linhas de código.

Um programa P4 inclui basicamente definições de cabeçalhos, metadados, *parsers*, ações, tabelas, blocos de controle e objetos externos. *Parsers* descrevem o mapeamento dos bits de um pacote de entrada para os respectivos *cabeçalhos* declarados no programa. Uma vez mapeados, tais cabeçalhos podem ser manipulados por *tabelas* e *ações*. A sequência exata de tabelas e ações aplicadas durante o processamento do pacote é definida de forma imperativa por *blocos de controle*. *Metadados* permitem armazenar temporariamente informações de estado do pacote. Por fim, *objetos externos* atuam como interfaces para estruturas de dados e funções específicas de cada dispositivo de encaminhamento. Por exemplo, contadores disponíveis em switches programáveis são manipulados por programas P4 através de objetos externos. Cabe ao fabricante dos dispositivos implementar essas interfaces.

O código de um programa P4 é normalmente organizado em bibliotecas. Uma biblioteca padrão contendo definições de tipos básicos da linguagem é fornecida juntamente com seu *framework* de programação. Exemplos de tipos básicos envolvem *packet_in* e *packet_out*, utilizados para representar o pacote (isto é, o *stream* de bits) de entrada e saída, respectivamente. Outras bibliotecas podem ser livremente definidas por fabricantes de dispositivos e programadores de rede.

Para que possa ser executado por um dispositivo de rede, um programa P4 é traduzido por um compilador para instruções específicas do dispositivo alvo. Isso permite que um programa P4 seja executado por dispositivos baseados em CPU, GPU, FPGA, processadores de rede ou ASICs programáveis, por exemplo. No processo, o compilador P4 converte o código fonte do programa em uma representação intermediária, modelada por um grafo acíclico direcionado (em inglês: directed acyclic graph, ou DAG). Cada nodo do grafo, representando um elemento do programa P4, é então transformado nas respectivas instruções de baixo nível de acordo com o dispositivo alvo. Várias otimizações de código costumam ser aplicadas durante esse processo, permitindo por exemplo que dispositivos de rede processem pacotes a taxas mais elevadas.

3. Verificação de programas P4 Baseada em Asserções

Para entender melhor o tipo de problema que estamos tratando, a Figura 1 mostra um exemplo de vulnerabilidade em um programa P4. Esse programa especifica um pipeline de processamento de pacotes contendo duas tabelas (*tcp_table* e *acl_table*), invocadas dentro do bloco de controle *ingress*. De maneira equivocada, ambas as tabelas serão aplicadas somente sobre tráfego TCP, fazendo com que todo pacote UDP seja encaminhado sem que qualquer tipo de filtragem possa ser aplicado.

Assumindo que a política de segurança da rede proíbe esse tipo de prática, o programa em questão poderia ser utilizado como ponto de partida para a execução de diversos ataques. Apesar da correção para esse exemplo ser simples (basta mover a tabela que implementa a lista de controle de acesso para fora da estrutura condicional), a capacidade de identificar automaticamente essa e outras vulnerabilidades em programas P4 representa um desafio de pesquisa relevante, para o qual apresentaremos uma proposta de solução no restante desse artigo.

© 2017 Sociedade Brasileira de Computação

```
 1 control ingress() {
 2   apply {
 3     ...
 4     if (headers.ip.nextHeader == TCP) {
 5       tcp_table.apply();
 6       // ACL será aplicada apenas sobre tráfego TCP
 7       acl_table.apply();
 8     }
 9     ...
10   }
11}
```

Figura 1. Exemplo de vulnerabilidade em um programa P4

3.1. Visão geral

A ideia-chave da proposta consiste em verificar modelos dos programas originais anotados com asserções simples, de forma a reduzir a complexidade do processo de testes como um todo. Nesse sentido, a Figura 2 mostra o fluxo de controle do processo de verificação. Primeiro, o programa P4 a ser verificado é anotado com asserções que expressam propriedades genéricas de interesse. Essas propriedades podem refletir uma política de segurança da rede ou simplesmente representar o comportamento esperado do programa. Uma vez anotado, o programa dá origem a um modelo expresso em linguagem C, criado por meio de um processo de tradução. Opcionalmente, regras de encaminhamento também podem ser utilizadas como entrada pelo tradutor. O modelo gerado é verificado por uma máquina de execução simbólica, a qual testa todos os caminhos de execução do programa para encontrar potenciais falhas ou vulnerabilidades. Se nenhuma das asserções inseridas é violada durante o processo de execução simbólica, o programa P4 é considerado seguro com relação às propriedades analisadas. Caso contrário, a respectiva violação é reportada para que as correções necessárias possam ser aplicadas. Na sequência, descrevemos em detalhe cada uma das etapas do processo de verificação proposto.

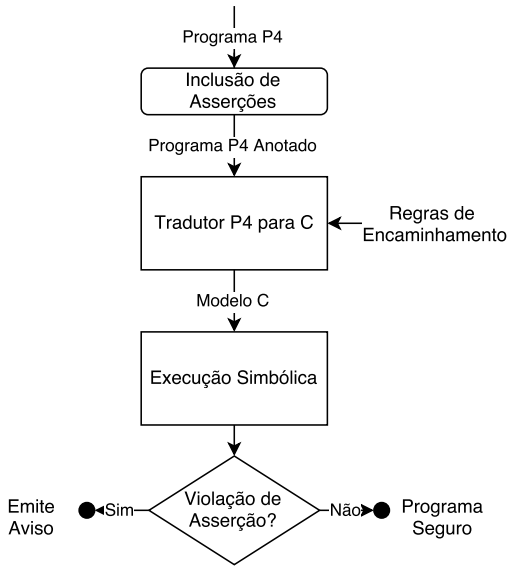


Figura 2. Fluxo de controle do processo de verificação.

 © 2017 Sociedade Brasileira de Computação

3.2. Especificação de Asserções

Para incluir asserções em um programa P4, nós definimos uma linguagem de especificação que permite expressar propriedades de interesse a respeito de qualquer programa para plano de dados. Nossa linguagem faz uso do mecanismo de anotações de código disponível em P4 através da anotação *assert*, de tal modo que tanto o desenvolvedor do código quanto uma terceira pessoa podem expressar e/ou interpretar propriedades a respeito do programa de maneira intuitiva. A Figura 3 apresenta a sua gramática. De acordo com a figura, cada asserção a é composta de uma expressão e ou método $m \in \{forward, traverse_path, constant, if, extract_header, emit_header\}$. Expressões e métodos, por sua vez, operam sobre um ou mais valores v, campos de cabeçalho f ou cabeçalhos h. O conjunto de métodos descrito foi escolhido de forma a facilitar a expressão das propriedades de segurança de maior interesse (p.ex., isolamento de rede, integridade de cabeçalhos, fluxo de informação, etc). Nosso objetivo não é permitir a especificação de qualquer propriedade. De qualquer forma, é possível facilmente estender a linguagem de asserções proposta no sentido de torná-la mais expressiva adicionando novos métodos.

Semanticamente, cada asserção representa um booleano que deve assumir valor lógico verdadeiro ou falso. Expressões têm a mesma semântica das suas equivalentes na linguagem P4, enquanto métodos possuem semântica própria. *forward* retorna verdadeiro quando o pacote sendo processado pelo programa é encaminhado para uma porta de saída válida. *traverse_path* indica se um dado trecho do programa foi executado, retornando verdadeiro sempre que essa asserção é de fato verificada. *constant(f)* é verdadeiro se o valor do campo *f* não é alterado durante a execução do programa. *if*(a_1, a_2, *[*a_3*])* possui semântica semelhante a declarações condicionais tradicionais: se a condição representada pela asserção a_1 for verdadeira, então a asserção a_2 será avaliada. Caso contrário, a asserção opcional a_3 terá sua validade verificada. *extract_header(h)* é avaliado como verdadeiro se bits do pacote de entrada são extraídos para o cabeçalho *h* durante o processo de *parsing* do pacote. Por fim, *emit_header(h)* retorna verdadeiro se o cabeçalho *h* for parte do pacote de saída ao final da execução do programa.

A Figura 4 mostra um exemplo de programa P4 anotado com asserções (em negrito). Por razões de espaço, somente as partes mais relevantes do programa são mostradas. Esse programa descreve um *pipeline* de processamento de pacotes com uma única tabela (*dmac*), instanciada dentro do bloco de controle *TopPipe*. Cada entrada dessa tabela pode invocar uma de duas ações (*Drop_action* ou *Set_dmac*). As asserções anotadas nesse programa buscam verificar: i) se pacotes marcados para descarte jamais são encaminhados (linha 7); e ii) se nenhum pacote com TTL igual a zero é encaminhado (linha 22).

3.3. Construção de Modelos C

Uma vez que o programa P4 é anotado, nosso mecanismo gera um modelo equivalente em linguagem C por meio de um processo de tradução. Essa seção descreve esse processo em detalhe, tomando como base as principais estruturas de um programa P4 (isto é, cabeçalhos, tabelas, ações, *parsers*, blocos de controle e objetos externos). O processo como um todo envolve duas etapas: i) transformação do programa P4 em um DAG; e ii) transformação do DAG gerado em código C a partir dos nodos de interesse. A Figura 5 exemplifica o processo de tradução das principais estruturas P4.

© 2017 Sociedade Brasileira de Computação

```
a ::= e | m
e ::= f
    | v
    | ! e
    | e && e
    | e || e
    | e == e
    | e != e
    | e >= e
    | e <= e
    | e + e
    | e - e
    | e * e
    | e / e

m ::= forward
    | traverse_path
    | constant(f)
    | if(a, a, [a])
    | extract_header(x)
    | emit_header(x)
```

Figura 3. Gramática da linguagem de asserções.

```
1  ...
2 control TopPipe(inout Parsed_packet headers,
3                 out OutControl outCtrl) {
4  ...
5  action Drop_action() {
6    outCtrl.outputPort = DROP_PORT;
7    @assert("if(traverse_path, !forward)");
8 }
9  action Set_dmac(EthernetAddress dmac) {
10   headers.ethernet.dstAddr = dmac;
11 }
12 table dmac {
13   key = { nextHop : exact; }
14   actions = { Drop_action; Set_dmac; }
15   size = 1024;
16   default_action = Drop_action;
17 }
18 apply {
19   ...
20   dmac.apply();
21   ...
22   @assert("if(forward, headers.ip.ttl != 0)");
23 }
24}
```

Figura 4. Exemplo de programa P4 anotado com asserções.

Cabeçalhos. Cabeçalhos P4 são adequadamente modelados por *structs* em C, dado que possuem representações semelhantes. Cada campo do cabeçalho é mapeado para um membro da *struct*. Utiliza-se *bit fields* em C para manter a correspondência entre o tamanho do campo de cabeçalho e o tamanho do respectivo membro na *struct* gerada. Tal correspondência permite modelar comportamentos de *overflow* e *underflow* das variáveis P4. Cada tipo básico em P4 é mapeado para um tipo correspondente em C, considerando seu tamanho declarado. Note que apesar de P4 não impor um limite de tamanho para campos de cabeçalho, nenhum campo com mais de 64 bits pode ser modelado por essa abordagem devido a restrições nos tipos de dados suportados pela linguagem C. Planejamos investigar maneiras de representar campos com mais de 64 bits utilizando múltiplas variáveis ou *arrays* em C como trabalhos futuros.

Tabelas. Cada tabela de um programa P4 é modelada como uma função em C. Funções criadas a partir de tabelas são construídas de maneira diferente dependendo se as regras de encaminhamento são fornecidas para o tradutor ou não. Caso as regras sejam fornecidas, os campos de *match* da tabela P4 são testados contra os respectivos valores das regras utilizando a abordagem de casamento especificada (e.g., exata). Caso contrário, a decisão de qual ação executar é feita com base em um valor simbólico especialmente declarado para forçar a criação de múltiplos caminhos de execução para o programa pela máquina de execução simbólica (um para cada ação que uma regra na respectiva tabela poderia invocar).

Ações. Assim como tabelas, ações são modeladas como funções em C. Os parâmetros das ações devem ser traduzidos levando em consideração a estratégia de modelagem de tabelas. Quando as regras são desconhecidas, os valores dos parâmetros da ação também são desconhecidos. Nesse caso eles são tratados como variáveis simbóli-

 © 2017 Sociedade Brasileira de Computação

	Programa P4	Modelo C

Cabeçalho	header ethernet_t {
	  bit<48> dstAddr;
	  bit<48> srcAddr;
	  bit<16> etherType;
	}
		typedef struct {
		 uint8_t isValid : 1;
		 uint64_t dstAddr : 48;
		 uint64_t srcAddr : 48;
		 uint32_t etherType : 16;
		} ethernet_t;

Tabela	table forward_table() {
	  actions = {
	  forward1;
	  NoAction;
	  }
	  key = {
	  hdr.ethernet.dstAddr: exact;
	  }
	  size = 32;
	  default_action = NoAction();
	}
		void forward_table() {
		  int symbol;
		  make_symbolic(symbol);
		  switch(symbol) {
		  case 0: forward(); break;
		  default: NoAction(); break;
		  }
		}

Ação	action forward(bit<9> port) {
	  standard_metadata.egress_spec = port;
	}
		void forward() {
		 uint32_t port;
		 make_symbolic(port);
		 standard_metadata.egress_spec = port;
		}

Bloco de controle	control ingress(inout headers hdr,
	  inout metadata meta) {
	  apply {
	  forward_table.apply();
	  }
	}
		// global variables
		headers hdr;
		metadata meta;
		
		void ingress() {
		  forward_table();
		}

Parser	parser TopParser(packet_in b,
	  out Parsed_packet hdr) {
	
	  state start {
	  transition parse_ethernet;
	  }
	
	 state parse_ethernet {
	  b.extract(hdr.ethernet);
	  transition select(hdr.ethernet.etherType) {
	  0x0800: parse_ipv4;
	  default: accept;
	  }
	 }
	
	}
		Parsed_packet hdr;
		
		void TopParser() {
		  make_symbolic(hdr);
		  start();
		}
		
		void start(){
		  parse_ethernet();
		}
		
		void parse_ethernet() {
		  hdr.ethernet.isValid = 1;
		  switch(hdr.ethernet.etherType){
		  case 0x0800: parse_ipv4(); break;
		  default: accept(); break;
		  }
		}

Asserção	action a1() {
	 @assert("traverse_path");
	}
		int path1 = 0;
		void a1(){
		  path1 = 1;
		}
		int main(){
		  ...
		  if(!path1) {
		  printf("Fail: traverse_path");
		  }
		  return 0;
		}

Figura 5. Exemplos de modelos C das principais estruturas P4.

cas. Se as regras são especificadas, então os parâmetros assumem os valores concretos especificados na regra equivalente.

Blocos de Controle. Como a declaração de um bloco de controle também inclui a declaração de suas ações e tabelas, os blocos acabam sendo traduzidos para múltiplas

© 2017 Sociedade Brasileira de Computação

funções C. Variáveis de escopo local do bloco de controle são declaradas como variáveis globais no modelo C para permitir que elas possam ser referenciadas por qualquer tabela e ação pertencentes ao bloco. O corpo do bloco tipicamente contém chamadas para tabelas e ações, que são modeladas como invocações de suas respectivas funções C.

Parser. Parsers são traduzidos para múltiplas funções em C: uma para o parser propriamente dito e outra para cada estado pertencente a ele. Como parâmetros e variáveis locais do parser podem ser acessados por qualquer estado em seu escopo, ambos são modelados como variáveis globais em C. Além disso, parâmetros de saída do parser (tipo *out*) ainda são declarados como variáveis simbólicas. Tais parâmetros representam os cabeçalhos do pacote, e portanto são equivalentes às entradas do modelo C. Sempre que o método externo *extract* (associado ao tipo básico *packet_in*) é encontrado em um estado do parser, o campo de validade do respectivo cabeçalho é configurado de acordo, deixando de ser uma variável simbólica. Em P4, ao final de cada estado do parser o fluxo de controle é transferido para o estado seguinte (comandos *transition* e *select)*. Nós modelamos esse comportamento em C através do encadeamento de chamadas de funcões, adotando elementos condicionais (isto é, *if* e *switch*) para as transições sempre que necessário.

Asserções. Cada tipo de asserção é modelada em C de maneira distinta. Operações booleanas e numéricas, bem como o método *if*, são diretamente traduzidas para construções equivalentes na linguagem C. Para modelar os métodos *extract_header*, *emit_header*, *traverse_path* e *forward*, é criada uma variável booleana global para cada ocorrência deles no programa P4. Tais variáveis assumem valor inicial falso e tem atribuído o valor verdadeiro em pontos diferentes do modelo de acordo com o respectivo método. Para o método *extract_header(h)* a atribuição ocorre logo após a invocação do método *extract* que recebe como parâmetro o cabeçalho *h* no programa P4. De forma similar, o método *emit_header(h)* faz com que a atribuição seja imediatamente após a invocação do método *emit* (associado ao tipo básico *packet_out*) cujo parâmetro é o cabeçalho *h*. Para *traverse_path*, a atribuição se dá logo antes da asserção que o contém. No caso do método *forward*, a atribuição ocorre ao final do *pipeline* de processamento de pacotes apenas se a porta de saída para qual o mesmo será encaminhado for válida. *constant(f)* é traduzido armazenando-se o valor do cabeçalho *f* em uma variável C no ponto da asserção, e testando se o valor dessa variável continua o mesmo ao final do programa.

Objetos externos. Esse tipo de estrutura é específico de cada dispositivo de encaminhamento, sendo somente sua interface utilizada por programas P4. Por essa razão, a modelagem do comportamento de cada objeto externo deve ser previamente conhecida. Na prática, isso significa integrar o modelo referente aos objetos externos de um dispositivo ao tradutor, por exemplo, por meio de bibliotecas.

3.4. Execução simbólica de modelos de programas P4

Após gerado pelo processo descrito na seção anterior, o modelo C de um programa P4 é verificado por uma máquina de execução simbólica. A execução simbólica de um programa requer que todos os seus possíveis fluxos de controle (isto é, seus caminhos de execução) sejam avaliados a partir de variáveis de entrada simbólicas. Nesse sentido, programas de planos de dados, e consequentemente seus respectivos modelos C, apresentam algumas particularidades específicas desse nicho de aplicação.

Essencialmente, programas P4 descrevem como um pacote de dados deve ser pro-

 © 2017 Sociedade Brasileira de Computação

cessado ao ingressar em um dispositivo de encaminhamento, gerando ao final um pacote de saída ou então simplesmente descartando o pacote original. Nesse caso, os cabeçalhos do pacote que ingressa no dispositivo são considerados entradas do modelo, e portanto sempre assumem valores simbólicos. O número de caminhos de execução de um programa P4, por sua vez, é dado basicamente pela estrutura do seu *pipeline* de processamento de pacotes. Sempre que uma tabela pode ser acessada somente sob alguma condição (p.ex., dependendo do protocolo sendo utilizado), um novo caminho de execução é criado. O mesmo acontece sempre que mais de uma opção de ação pode ser invocada por uma mesma tabela.

4. Avaliação Experimental

Essa seção descreve a avaliação do mecanismo de verificação proposto. Nosso principal objetivo é responder às seguintes questões: (i) é possível encontrar vulnerabilidades em programas P4 utilizando o mecanismo proposto? (ii) quão expressivo é o mecanismo em termos do conjunto de propriedades de segurança que pode ser verificado? e (iii) quão eficiente é o mecanismo (isto é, em quanto tempo é possível verificar propriedades de interesse de acordo com as características de um programa)?

O restante da seção está organizado como segue: a Seção 4.1 apresenta o protótipo implementado bem como o ambiente de testes utilizado, enquanto as Seções 4.2 e 4.3 descrevem os estudos de caso realizados para avaliar tanto o potencial do mecanismo proposto como ferramenta de segurança quanto a sua capacidade de verificar um conjunto variado de propriedades. Por fim, a Seção 4.4 faz uma análise do seu desempenho.

4.1. Implementação e ambiente de testes

Nós prototipamos o mecanismo de verificação proposto utilizando a ferramenta Klee [Cadar et al. 2008] (versão 1.3.0) como máquina de execução simbólica. Para a construção de modelos C, primeiro convertemos um programa P4 para sua representação JSON utilizando o compilador de referência da linguagem, disponibilizado pelo *P4 Language Consortium*. Em seguida, traduzimos o modelo JSON (representado através de um DAG) para código C utilizando um tradutor implementado exclusivamente para esse propósito. O tradutor contém aproximadamente 750 linhas de código Python. Scripts Shell são utilizados para coordenar de forma automática a invocação de cada ferramenta no processo de verificação. O código fonte do tradutor, bem como os scripts e as cargas de trabalho utilizadas estão disponíveis online [3]. Todos os experimentos foram executados utilizando-se uma máquina virtual Linux (kernel versão 4.8.0) com um núcleo de 3 GHz e 16 GB de RAM.

4.2. Estudo de caso: DC.p4

Para mostrar o potencial do mecanismo de verificação proposto como ferramenta de defesa (isto é, prevenção) contra ataques a redes oriundos de vulnerabilidades em programas P4, selecionamos o programa DC.p4 como estudo de caso. DC.p4 foi proposto por [Sivaraman et al. 2015], e captura o comportamento de um switch de data center. Entre suas principais funcionalidades estão encaminhamento L2 e L3, VLAN, MAC learning, espelhamento de pacotes, tunelamento, listas de controle de acesso (ACLs) e balanceamento de carga (ECMP). Tal programa contém mais de 2500 linhas de código distribuídas

[3] https://github.com/ufrgs-networks-group/assert-p4

© 2017 Sociedade Brasileira de Computação

entre 37 tabelas. Essas tabelas, por sua vez, estão organizadas assumindo uma arquitetura com dois pipelines sequenciais de processamento de pacotes, um de entrada/ingresso e outro de saída/egresso, intercalados por um sistema de filas. Escolhemos esse programa pela sua complexidade e representatividade de cenários reais, e executamos verificações sobre o mesmo no intuito de encontrar falhas que pudessem ser exploradas em potenciais ataques.

Falhas em funcionalidades de segurança. O primeiro conjunto de experimentos realizado diz respeito à verificação de falhas nas funcionalidades de segurança implementadas pelo programa (p.ex. VLAN e ACL para prover isolamento de rede). Mais especificamente, verificamos se é possível desviar de alguma dessas funcionalidades de maneira a evitar que a respectiva política de segurança seja de fato aplicada sobre um pacote. Utilizamos asserções no formato *if(bypass == 0, traverse_path)* para expressar esse tipo de propriedade, onde *bypass == 0* indica que a funcionalidade de segurança está ativa (isto é, foi configurada pelo administrador), e *traverse_path* verifica se a mesma está sendo executada para todas as entradas possíveis.

Dada a complexidade do programa, o processo de verificação como um todo leva na ordem de dias para terminar (veja Figura 6(a)). Entretanto, nosso verificador foi capaz de encontrar violações das propriedades testadas em menos de 10s. Constatamos que apesar do programa oferecer múltiplas opções de listas de controle de acesso (p.ex. listas de níveis 2 e 3), pacotes IPv4 não passam por uma lista de nível 2, o que pode se transformar numa vulnerabilidade dependendo da configuração adotada pelo administrador da rede (p.ex. caso o administrador resolva configurar somente uma ACL de nível 2).

Falhas em replicações de tráfego. Algumas funcionalidades do programa DC.p4 (p.ex., espelhamento de pacotes) envolvem a replicação de tráfego entre portas do dispositivo ou para a sua CPU. Ambos os casos tipicamente fazem parte de alguma tarefa de monitoramento da rede. Se executados erroneamente, tais procedimentos podem servir de origem para ataques baseados em amplificação de tráfego [Krupp et al. 2016] ou replicação de pacotes [Lee et al. 2017] (p.ex., caso a réplica seja encaminhada ao mesmo destino do pacote original). Nesse sentido, verificamos se os procedimentos de replicação do programa DC.p4 são executados corretamente (isto é, se as réplicas são endereçadas às portas corretas ou se são devidamente modificadas antes do seu encaminhamento). Utilizamos asserções no formato ! *(outport == original_port && constant(outport))* dentro de ações que replicam o tráfego para expressar essas propriedades. O primeiro e o segundo membro que compõem essas asserções testam se a porta de saída do pacote replicado é a mesma do pacote original e se tal porta é alterada durante o processamento do pacote replicado, respectivamente.

Assim como no caso anterior, nosso mecanismo foi capaz de encontrar violações de propriedades rapidamente (menos de 13 minutos). De maneira geral, o programa é genérico o suficiente para permitir que o plano de controle configure a porta de saída tanto da réplica quanto do pacote original, ao mesmo tempo que nenhuma alteração de cabeçalho é feita sobre a réplica. Dessa forma, cabe ao plano de controle através da configuração das tabelas do dispositivo garantir que um mesmo pacote não seja encaminhado mais de uma vez para o mesmo destino, o que pode não ser uma tarefa simples.

 © 2017 Sociedade Brasileira de Computação

4.3. Outros estudos de caso

Para avaliar a expressividade do mecanismo proposto em termos do conjunto de propriedades que podem ser verificadas, nós especificamos e testamos asserções para diferentes programas P4 utilizando a linguagem descrita na Seção 3.2. Nesse sentido, a Tabela 1 lista os diferentes programas testados, seus tamanhos (em linhas de código), as respectivas propriedades verificadas, o tempo de verificação em segundos e o consumo de memória observado (em MBs)[4]. Note que o tempo de verificação e o consumo de memória foram consideravelmente baixos para todos os casos.

Tabela 1. Estudos de caso

Programa	Tamanho	Propriedades	Tempo	Memória
VSS	226	Integridade do cabeçalho IPv4, pacotes com TTL igual a zero são descartados, pacotes marcados para descarte não são encaminhados	1s	< 2MB
MRI	278	Switch IDs adicionados em pacotes são autênticos, IDs adicionados não são removidos	2s	< 2MB

4.4. Análise de desempenho

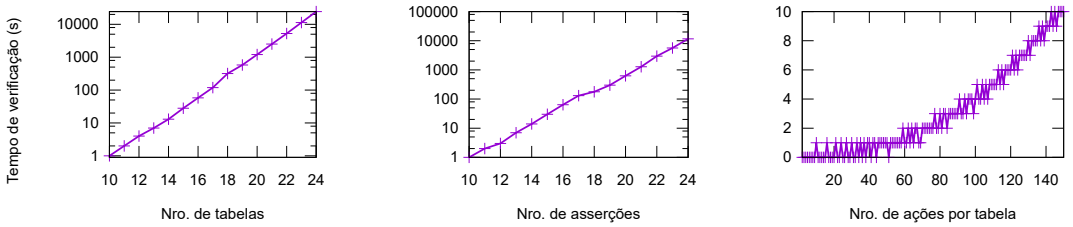
Nessa seção, nós procuramos avaliar como o mecanismo de verificação proposto escala com relação a variações nas caracterísitcas de um programa P4. Para tanto, utilizamos o *benchmark* de planos de dados programáveis Whippersnapper [Dang et al. 2017] para gerar sinteticamente programas P4 com diferentes tamanhos de *pipeline* (isto é, quantidades de tabelas). Adicionalmente, testamos o impacto da variação: i) da quantidades de ações associadas com cada tabela; e ii) do número de asserções inseridas no código, para uma instância padrão de programa P4 gerada pelo *benchmark*, uma vez que esse não permite variar automaticamente o número de ações ou de anotações no código. Consideramos para esses casos um programa com somente duas tabelas (no intuito de minimizar o impacto desse fator no tempo de verificação), inserindo novas ações e asserções através de *scripts*.

Conforme esperado, o tempo de verificação cresce exponencialmente tanto para variações do número de tabelas quanto de asserções. Entretanto, é interessante notar que tal tempo é da ordem de algumas dezenas de segundos para programas com *pipelines* de até 15 tabelas (Figura 6(a)), o que inclui diversos programas P4 existentes atualmente [Dang et al. 2015, Sivaraman et al. 2017]. Ao considerarmos o número de asserções, podemos ainda perceber que verificações com mais de 20 propriedades podem levar algumas horas para serem concluídas (Figura 6(b)). Nesse caso, pode ser interessante verificar conjuntos diferentes de propriedades para um mesmo programa em execuções distintas, de forma que o tempo de verificação seja relativamente baixo para todas as execuções (verificações com até 10 propriedades não impactam significativamente o tempo de execução).

[4]VSS: https://github.com/p4lang/p4c/blob/master/testdata/p4_16_samples/vss-example.p4
MRI: https://github.com/p4lang/tutorials/blob/master/P4D2_2017/exercises/mri/solution/mri.p4

© 2017 Sociedade Brasileira de Computação

Por fim, podemos observar que o tempo de execução cresce de forma polinomial com a variação do número de ações (Figura 6(c)). Nesse caso, o grau do polinômio depende da quantidade de tabelas do programa P4 (p.ex., duas para o programa testado). Na prática, para um programa composto de p tabelas sequenciais, onde cada tabela pode invocar uma dentre q opções de ações diferentes, um total de q^p caminhos distintos devem ser executados simbolicamente. Pragmaticamente, programas P4 não costumam ter mais do que uma dezena de opções de ações para cada tabela [Dang et al. 2015, Sivaraman et al. 2017], o que torna o mecanismo proposto viável para a maioria das instâncias existentes.



(a) (b) (c)

Figura 6. Análise de desempenho do mecanismo proposto.

5. Trabalhos Relacionados

Verificação de redes. É possível encontrar na literatura diferentes abordagens de verificação que reforçam a segurança de planos de dados. Flover [Son et al. 2013] permite provar que o conjunto de regras instanciadas em switches OpenFlow satisfazem políticas de segurança da rede. Essa solução é inadequada para verificação de planos de dados programáveis devido a sua incapacidade de modelar comportamentos arbitrários de encaminamento. Para habilitar a verificação de redes com dispositivos programados em P4, recentemente foi proposto um mecanismo que cria automaticamente modelos em Datalog a partir de programas P4, de regras de encaminhamento e da topologia da rede [Lopes et al. 2016]. Os modelos criados são verificados com a ferramenta NOD (um resolvedor SAT) [Lopes et al. 2015] para provar propriedades relacionadas a alcançabilidade e formação correta de pacotes. Diferentemente desse trabalho, nossa abordagem permite verificar propriedades de segurança como integridade de cabeçalhos e ausência de amplificação de tráfego. Além disso, utilizamos uma linguagem de asserções que torna simples e intuitiva a especificação dessas propriedades.

Execução simbólica. A técnica de execução simbólica já foi usada anteriormente para verificar planos de dados. [Dobrescu and Argyraki 2014] prova que *pipelines* compostos por elementos Click satisfazem propriedades de *crash-freedom*, *bounded execution* e filtragem de pacotes. Os autores tentam lidar com o problema de explosão de estados executando simbolicamente os elementos Click de forma isolada. Symnet [Stoenescu et al. 2016], por sua vez, é um verificador de modelos de planos de dados construídos utilizando a linguagem SEFL, também proposta pelos autores. Tal linguagem contém instruções que simplificam a sua execução simbólica, permitindo a verificação eficiente de programas complexos. Apesar da sua escalabilidade, Symnet não pode verificar programas P4 em seu estado atual, uma vez que a linguagem SEFL não consegue

 © 2017 Sociedade Brasileira de Computação

modelar todas as estruturas e manipulações de dados necessárias (p.ex., operações sobre bits).

Linguagem de asserções. Em [Beckett et al. 2014], os autores apresentam uma linguagem de asserções para verificar aplicações SDN. Com ela, é possível expressar propriedades que o plano de dados deve satisfazer em diferentes pontos de um programa de controle. As asserções são verificadas utilizando-se a ferramenta VeriFlow [Khurshid et al. 2012], que assim como Flover atua sobre regras de encaminhamento instanciadas em dispositivos OpenFlow. Enquanto a linguagem proposta por [Beckett et al. 2014] é utilizada em aplicações SDN, nossa abordagem é anotar diretamente um programa de plano de dados para provar propriedades de interesse.

6. Conclusão e Trabalhos Futuros

Apresentamos nesse trabalho uma linguagem de asserções que pode ser utilizada por programadores P4 para expressar propriedades de corretude e segurança de uma determinada implementação. Nossa solução é mais expressiva que outras abordagens de verificação de planos de dados, sendo o primeiro trabalho a permitir provar propriedades específicas de códigos fonte P4 e opcionalmente as regras de encaminhamento utilizadas por suas tabelas. Nosso mecanismo verifica as asserções utilizando execução simbólica sobre modelos C gerados automaticamente a partir do programa e asserções.

Avaliamos nossa abordagem provando propriedades de replicação de tráfego e contorno de funcionalidades de segurança do programa DC.p4, bem como diversas propriedades de programas encontrados na especificação e tutoriais de P4. A análise de desempenho do mecanismo proposto revelou que apesar de sua eficiência em verificar programas pequenos, o tempo de execução cresce exponencialmente com número de tabelas, ações e asserções. Por essa razão, em paralelo à abordagem aqui descrita, estamos trabalhando com os autores de SEFL na extensão da linguagem para habilitar a modelagem completa de programas P4. Este é um esforço de médio prazo, mas vislumbramos que através do Symnet será possível a verificação escalável de asserções em programas P4.

Referências

Beckett, R., Zou, X. K., Zhang, S., Malik, S., Rexford, J., and Walker, D. (2014). An assertion language for debugging sdn applications. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 91–96, New York, NY, USA. ACM.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Cadar, C., Dunbar, D., and Engler, D. (2008). Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 209–224, Berkeley, CA, USA. USENIX Association.

Dang, H. T., Sciascia, D., Canini, M., Pedone, F., and Soulé, R. (2015). Netpaxos: Consensus at network speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on*

© 2017 Sociedade Brasileira de Computação

Software Defined Networking Research, SOSR '15, pages 5:1–5:7, New York, NY, USA. ACM.

Dang, H. T., Wang, H., Jepsen, T., Brebner, G., Kim, C., Rexford, J., Soulé, R., and Weatherspoon, H. (2017). Whippersnapper: A p4 language benchmark suite. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 95–101, New York, NY, USA. ACM.

Dobrescu, M. and Argyraki, K. (2014). Software dataplane verification. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 101–114, Seattle, WA. USENIX Association.

Khurshid, A., Zhou, W., Caesar, M., and Godfrey, P. B. (2012). Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 49–54, New York, NY, USA. ACM.

Krupp, J., Backes, M., and Rossow, C. (2016). Identifying the scan and attack infrastructures behind amplification ddos attacks. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1426–1437, New York, NY, USA. ACM.

Lee, T., Pappas, C., Perrig, A., Gligor, V., and Hu, Y.-C. (2017). The case for in-network replay suppression. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 862–873, New York, NY, USA. ACM.

Lopes, N., Bjørner, N., McKeown, N., Rybalchenko, A., Talayco, D., and Varghese, G. (2016). Automatically verifying reachability and well-formedness in p4 networks. Technical report.

Lopes, N. P., Bjørner, N., Godefroid, P., Jayaraman, K., and Varghese, G. (2015). Checking beliefs in dynamic networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 499–512, Oakland, CA. USENIX Association.

Sivaraman, A., Kim, C., Krishnamoorthy, R., Dixit, A., and Budiu, M. (2015). Dc.p4: Programming the forwarding plane of a data-center switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 2:1–2:8, New York, NY, USA. ACM.

Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., and Rexford, J. (2017). Smoking out the heavy-hitter flows with hashpipe. *SOSR '17*.

Son, S., Shin, S., Yegneswaran, V., Porras, P., and Gu, G. (2013). Model checking invariant security properties in openflow. In *2013 IEEE International Conference on Communications (ICC)*, pages 1974–1979. IEEE.

Stoenescu, R., Popovici, M., Negreanu, L., and Raiciu, C. (2016). Symnet: Scalable symbolic execution for modern networks. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 314–327, New York, NY, USA. ACM.

 © 2017 Sociedade Brasileira de Computação

Cache nFace: uma Contramedida Simples para o Ataque em Conluio Produtor-Consumidor em Redes Centradas em Conteúdo

André Nasserala [1,2], **Igor Monteiro Moraes**[1]

[1]Laboratório MídiaCom, PGC-TCC
Instituto de Computação - Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brasil

[2]Centro de Ciências Exatas e Tecnológicas, CCET
Universidade Federal do Acre
Rio Branco, Acre, Brasil

{anasserala,igor}@ic.uff.br

Abstract. *This paper proposes and evaluates a countermeasure to mitigate the producer-consumer collusion attack in the Content Centric Networking architecture called Cache nFace. The proposed countermeasure mitigates this attack by dividing the cache of a node in sub-caches. Each sub-cache only stores contents requested through a specific network interface. The goal is to increase the robustness of the cache under attack. We perform simulations to evaluate our proposal for different network topologies and attack configurations. Results show Cache nFace reduces the attack efficiency by up to 50% and outperforms other proposal found in the literature in all analyzed scenarios.*

Resumo. *Esse artigo propõe e avalia uma contramedida para comedir o ataque de negação de serviço por conluio produtor-consumidor na arquitetura CCN chamada de Cache nFace. A contramedida proposta comede este ataque dividindo o cache de um nó em sub-caches. Cada sub-cache armazena somente os conteúdos solicitados através de uma interface de rede específica. Isso aumenta a robustez do cache sob ataque. São realizadas simulações para avaliar a proposta para diferentes topologias de rede e configurações do ataque. Os resultados mostram que o Cache nFace reduz em até 50% a eficiência do ataque e supera outra proposta encontrada na literatura em todos os cenários analisados.*

1. Introdução

As Redes Orientadas a Conteúdo são um novo paradigma de comunicação para a Internet [Brito et al. 2012]. O principal objetivo dessas redes é a entrega de conteúdo para os usuários independentemente da localização desse conteúdo, ao contrário da arquitetura TCP/IP, cujo objetivo é a comunicação entre sistemas finais. Diversas arquiteturas foram propostas para esse novo paradigma de comunicação e uma das arquiteturas com maior destaque na literatura é a *Content Centric Networking* (CCN) [Jacobson et al. 2009, Smetters and Jacobson 2009]. Entre as principais características da CCN estão o roteamento através de nomes de conteúdo e o armazenamento de conteúdo em nós intermediários da rede [Jacobson et al. 2009].

 © 2017 Sociedade Brasileira de Computação

Outra característica da CCN é que a segurança é aplicada diretamente aos conteúdos, diferentemente da arquitetura TCP/IP, na qual a segurança é geralmente aplicada ao canal de comunicação entre os sistemas finais [Smetters and Jacobson 2009]. Um pacote de dados CCN é auto-certificado, isto é, ele contém a assinatura digital do conteúdo e do seu nome e a chave pública do produtor, ou informações para obtê-la [Jacobson et al. 2009]. Portanto, qualquer nó de posse de um pacote pode verificar a integridade do pacote e se esse pacote foi gerado pelo produtor que possui tal chave pública. Além disso, a CCN é mais robusta a ataques de negação de serviço (*Denial of Service* - DoS) comuns na Internet atual, como o de esgotamento de banda e o de reflexão, porque na CCN não há garantias de quem um pacote sempre será encaminhado até um dado nó vítima. Isso em virtude do uso de *cache* e da agregação de solicitações de conteúdo feita pelos nós intermediários [Gasti et al. 2013]. No entanto, ataques de negação de serviço específicos para a CCN foram identificados [Smetters and Jacobson 2009].

Um ataque de negação de serviço particular, chamado de conluio produtor-consumidor, entretanto, pode ser efetivo porque os mecanismos nativos empregados pela CCN não são suficientes para inibi-lo. Nesse ataque, consumidores maliciosos solicitam conteúdos que são disponibilizados apenas por produtores maliciosos a uma alta taxa. Isso aumenta o tempo de recuperação de conteúdos legítimos, em virtude do aumento da taxa de erro do *cache* (*cache miss*) para esses conteúdos e, consequentemente, da necessidade de nós legítimos terem que recuperar o conteúdo diretamente do seu produtor. Os mecanismos de segurança da CCN padrão são ineficazes na detecção do ataque em conluio, pois, do ponto de vista da rede, as solicitações e os conteúdos são legítimos. São enviados pacotes de interesse para conteúdos que existem e que são disponibilizados por produtores. O conteúdo é malicioso porque torna popular um conteúdo que não é de interesse de usuários legítimos. Como o produtor malicioso assina os conteúdo de acordo com a política definida pela CCN, os consumidores maliciosos podem solicitá-los sem risco de que esses conteúdos sejam descartados por mecanismos de verificação de assinaturas e chaves.

Em trabalhos anteriores [Nasserala and Moraes 2015, Nasserala and Moraes 2016b, Nasserala and Moraes 2016a], o ataque em conluio produtor-consumidor foi avaliado sem nenhuma proposta de contramedida e em uma topologia de rede específica. Os resultados mostraram que o ataque aumenta o tempo de recuperação de conteúdos em virtude do aumento da ocupação de conteúdos maliciosos e a taxa de erro dos *caches* dos nós da rede. Em um dos cenários avaliados, se 20% dos consumidores são maliciosos e enviam 500 solicitações de conteúdo por segundo cada um, o tempo de recuperação de conteúdos por consumidores legítimos aumenta em 20 vezes. Isso comprova a potencialidade do ataque.

Este artigo, por sua vez, propõe uma contramedida ao ataque em conluio-produtor consumidor chamada Cache nFace. O mecanismo proposto divide o *cache* de um nó em sub-*caches*. Cada um desses sub-*caches* é associado a uma interface de rede de um nó e o espaço de armazenamento de cada sub-*cache* é definido de acordo com taxa de transmissão da interface associada. Dessa forma, um sub-*cache* s_i só pode armazenar conteúdos, cujas solicitações de conteúdo tenham sido recebidas pela interface i. Isso aumenta a robustez do *cache* sob ataque devido à probabilidade de todas as interfaces de um nó serem atacadas ao mesmo tempo ser pequena. Caso exista ao menos uma interface

 © 2017 Sociedade Brasileira de Computação

que não esteja recebendo pacotes maliciosos, essa interface pode responder a solicitações legítimas e aumentar a taxa de acerto do *cache*. Resultados de simulação mostram que o Cache nFace pode reduzir em até 50% o tempo de recuperação de conteúdos legítimos em um dos cenários de ataque avaliados.

O trabalho está organizado da seguinte forma. Na Seção 2 são apresentadas as características principais da CCN e a definição do ataque em conluio produtor-consumidor. Na Seção 3 são apresentados os trabalhos relacionados. Em seguida, na Seção 4, o mecanismo Cache nFace é descrito. Na Seção 5 são definidos os cenários de avaliação e os resultados dos experimentos são analisados e discutidos. Por fim, na Seção 6 as conclusões são apresentadas.

2. A Arquitetura CCN e o Ataque em Conluio Produtor-Consumidor

Esse seção descreve brevemente o funcionamento da arquitetura CCN e define o ataque em conluio produtor-consumidor.

A CCN emprega dois tipos de pacotes: interesse e dados [Jacobson et al. 2009]. Consumidores enviam pacotes de interesse para solicitar um conteúdo. Os produtores ou roteadores de conteúdo respondem aos interesses com pacotes de dados, que carregam o conteúdo em si ou pedaços de conteúdo, chamados de *chunks* [Brito et al. 2013]. Os nós encaminham tanto pacotes de interesse quanto pacotes de dados com base no próprio nome do conteúdo, ao invés do endereço de destino do nó que possui o conteúdo. Um nó recebe um pacote de interesse e responde com o pacote de dados correspondente, ou encaminha o interesse se não tiver o dado em seu *cache*. Para realizar o encaminhamento de pacotes, cada nó CCN tem duas estruturas de dados: a *Pending Interest Table* (PIT) e a *Forwarding Information Base* (FIB). A PIT guarda o estado de cada pacote de interesse encaminhado por um nó que ainda não recebeu uma resposta, ou seja, os interesses que esperam por um pacote de dados. Cada entrada da PIT também armazena as interfaces de recepção de um pacote de interesse e possuem um temporizador associado. É importante ressaltar que o tamanho da PIT é limitado e, dessa forma, novos interesses que chegam enquanto a tabela está cheia não são encaminhados. A FIB atua como uma tabela de encaminhamento para pacotes de interesse. Essa tabela contém uma lista de entradas, cada uma contendo o prefixo de um nome e uma lista de interfaces de saída para as quais os pacotes de interesse com nomes de mesmo prefixo devem ser encaminhados. Os dados são armazenados no *Content Store* (CS), que é efetivamente o *cache* do nó.

Quando um nó CCN recebe um pacote de interesse, ele verifica seu CS para encontrar uma cópia do conteúdo solicitado, cujo nome está no cabeçalho do pacote de interesse. Se o conteúdo está armazenado em *cache*, o nó envia um pacote de dados para o consumidor. Caso contrário, o nó verifica a sua PIT. Se houver uma entrada na PIT para o mesmo conteúdo, o nó atualiza a lista de interfaces de entrada e descarta o pacote de interesse. Caso não haja entrada, consulta a FIB para determinar a interface de saída para encaminhar o pacote de interesse, então, o nó cria uma nova entrada na PIT. Se não houver nenhuma entrada na FIB relacionada com o nome do conteúdo, o pacote de interesse é descartado. Os nós repetem este processo de encaminhamento para cada pacote de interesse recebido. Os pacotes de conteúdo seguem o caminho reverso dos pacotes de interesse, ou seja, são encaminhados por todas as interfaces listadas na entrada da PIT mantida por um nó e que corresponde ao nome do conteúdo [Jacobson et al. 2009].

© 2017 Sociedade Brasileira de Computação

Ataques de negação de serviço (DoS) são uma ameaça na Internet atual. A arquitetura CCN, entretanto, é mais robusta a esse tipo de ataque do que a pilha TCP/IP em virtude do armazenamento de conteúdo pelos nós intermediários e da agregação de pacotes de interesse [Gasti et al. 2013]. Ataques de esgotamento de banda e de reflexão, por exemplo, são pouco eficientes na CCN. Apesar de ser mais robusta, a arquitetura CCN possui outros ataques e vulnerabilidades identificados [Gasti et al. 2013].

Os ataques de negação de serviço em CCN que esgotam os recursos da rede, afetando a infraestrutura, são classificados, principalmente, em dois tipos: ataques por inundação de interesses ou envenenamento de *cache* [Smetters and Jacobson 2009]. O objetivo dos ataques de inundação de interesses é sobrecarregar a PIT com solicitações de conteúdo enviadas por um nó malicioso a uma alta taxa. Por outro lado, o objetivo do ataque de envenenamento de *cache* é ocupar o *cache* dos nós com conteúdo poluído. Esse conteúdo é enviado por consumidores maliciosos para fazer com que nós armazenem um conteúdo que possua uma assinatura válida, porém corrompido ou aumentem a popularidade de conteúdos menos populares. O ataque em conluio é uma variação do envenenamento do *cache*, cujo objetivo é prejudicar o consumo indireto de conteúdos, isto é, obrigar um consumidor legítimo a recuperar o conteúdo desejado diretamente do produtor. Esse objetivo é alcançado através da manipulação da popularidade dos conteúdos armazenados em *cache*. Consumidores maliciosos enviam pacotes de interesse para um grupo de conteúdos que existem e que são respondidos pelo produtor malicioso. Assim, se solicitado com frequência, um conteúdo se torna popular, apesar de não ter sido solicitado por usuários legítimos. Por isso, o conteúdo é dito malicioso.

O ataque em conluio é possível, porque a CCN emprega políticas de substituição do *cache* baseadas, em sua maioria, na popularidade dos conteúdos. Assim, se um determinado conteúdo não é solicitado com frequência ou não foi solicitado recentemente pelos consumidores, ele é considerado menos popular. Dessa forma, terá prioridade de descarte quando houver necessidade de armazenar novos conteúdos. Ao solicitar um conjunto específico de conteúdos e em taxas altas, os nós maliciosos manipulam a política de substituição do *cache*. Com mais conteúdos maliciosos em *cache*, maior a taxa de erro para os conteúdos legítimos e, consequentemente, maior a necessidade de nós legítimos terem que recuperar o conteúdo diretamente do seu produtor. Mesmo que os consumidores legítimos não tenham que consumir diretamente dos produtores, eles terão seus interesses encaminhados por mais saltos até conseguir o conteúdo desejado.

Uma das principais razões para que o ataque em conluio consumidor-produtor seja bem sucedido é o fato de que os pacotes de interesse e de dados usados no ataque são legítimos para a rede e, portanto, não são detectados por mecanismos de verificação de assinaturas. O pacote com o conteúdo malicioso possui uma assinatura válida, carrega a chave do publicador, ou informações para obtê-la, e assim, passa no teste de verificação de integridade e autenticidade. Logo, não é identificado como malicioso e nem descartado.

3. Trabalhos Relacionados

Todos os trabalhos descritos a seguir avaliam e/ou propõem contramedidas para os ataques de negação de serviço por inundação de interesses e por envenenamento de *cache*. Porém, nenhum, especificamente, propõe uma contramedida para comedir ataques em que consumidores e produtores maliciosos agem em conluio para gerar, disponibilizar

 © 2017 Sociedade Brasileira de Computação

e manipular a popularidade de conteúdos. O mecanismo proposto nesse trabalho visa comedir esse tipo de ataque, como detalhado na Seção 4.

Gasti *et al.* propõem um mecanismo de *push-back* como contramedida ao ataque de inundação de interesses [Gasti et al. 2013]. Esse mecanismo monitora a ocupação da PIT e identifica quando uma determinada interface está próxima de atingir seu número máximo de entradas. Desta forma, o mecanismo avalia e controla o fluxo de pacotes de interesse que contém os mesmos nome de prefixos para determinar um limiar. Além disso, a contramedida envia uma notificação na interface supostamente atacada que será recebida por um nó vizinho. Esse nó, por sua vez, deve propagar tal informação no sentido das interfaces atacadas e, ao mesmo tempo, limitar a taxa de interesses encaminhados que contenham o prefixo sob ataque. Portanto, o objetivo da contramedida é empurrar o ataque para o caminho de volta até o atacante, ou pelo menos para um nó no qual seja detectado [Gasti et al. 2013]. A principal característica dessa contramedida é não modificar a arquitetura padrão proposta para a CCN. O ponto fraco do trabalho de Gasti *et al.* é que nem o impacto do ataque e nem a contramedida proposta são avaliados por simulação ou experimentos práticos.

Afanasyev *et al.* [Afanasyev et al. 2013] também avaliam o ataque de inundação de interesse através de simulações, porém consideram diferentes cenários e uma rede de maior escala do que a usada no trabalho de Choi *et al.* Os autores também avaliam a contramedida baseada em um mecanismo de *push-back* proposta por Gasti *et al.*. Os resultados mostram que essa contramedida é eficiente, pois isola por completo os atacantes de modo que eles causem pouco ou nenhum impacto no desempenho percebido por usuários legítimos. Os autores não variam a posição do atacante. Para validar o mecanismo definiram o pior cenário, onde os interesses não serão satisfeitos com *cache* de um nó intermediário. Deste modo os dados são sempre encaminhados diretamente para o produtor. Os autores justificam essa medida afirmando que esse seria o pior caso para o ataque avaliado.

Os trabalhos a seguir tratam dos ataques de envenenamento de *cache*. Kim *et al.* [Kim et al. 2013b] investigam o impacto de fluxos de conteúdo de longa duração na CCN. A presença de fluxos de longa duração pode ter efeito similar ao ataque de envenenamento de *cache*. Se fluxos de longa duração ocuparem temporariamente um *cache* de um nó por determinado conteúdo, eles podem expulsar pedaços de conteúdos populares do *cache*. Consequentemente, reduz-se a taxa de acertos do *cache* (*cache hit*). Os resultados das simulações mostram que há degradação da taxa de acertos do *cache* quanto maior é o número de fluxos de longa duração.

Xie *et al.* [Xie et al. 2012] propõem um mecanismo chamado CacheShield, que usa dados estatísticos para verificar se um conteúdo é poluído ou não. O objetivo do CacheShield é aumentar a robustez do *cache* contra ataques de poluição, em particular perturbações de localidade (*locality disruption*). O mecanismo tenta impedir que conteúdos não populares sejam armazenados no *cache*. Quando um roteador habilitado para o CacheShield recebe um pedaço de conteúdo, o nó executa uma função de proteção que determina se o pedaço de conteúdo deve ser armazenado em *cache* ou não. Se a função de proteção retornar verdadeiro, o roteador encaminha o interesse e armazena em *cache* o pedaço de conteúdo. Se a função de proteção retorna falso, apenas o nome do pedaço de conteúdo (ou o *hash* de seu nome) e um contador são armazenados no *cache* num espaço

© 2017 Sociedade Brasileira de Computação

reservado, fora da área do CS. Se o mesmo pedaço de conteúdo, que ainda não está armazenado em *cache*, é solicitado novamente, o contador correspondente é incrementado, e a função de proteção é novamente executada. Assim a cada vez que o pedaço de conteúdo é avaliado pela função aumenta-se a chance desse ser efetivamente armazenado no *cache*. A função de proteção executa uma escolha aleatória com base em uma função logística: $\psi(t) = \frac{1}{1+e^{(p-t)/q}}$, onde t representa a *t-ésima* requisição para um determinado pedaço de conteúdo no CS, e p e q são os parâmetros da função. O mecanismo CacheShield será usado, para fins de comparação, com o mecanismo proposto por esse trabalho.

4. O Mecanismo Cache nFace

A contramedida proposta, chamada de Cache nFace, não tenta identificar o tráfego malicioso. A ideia principal da proposta é criar sub-*caches* em um nó a partir da divisão do espaço total de armazenamento em *cache* desse nó. Cada sub-*cache* está associado a uma das interfaces de rede do nó, e o seu tamanho é definido de acordo com a taxa de transmissão nominal da interface, como mostra o Algoritmo 1. Esse algoritmo recebe como entradas o tamanho total do *cache* do nó, S, o número de interfaces do nó, N, I o conjunto composto por todas as interfaces, e a taxa de transmissão q_i de cada interface i do nó e retorna o tamanho do sub-*cache* s_i de cada interface i. Q é soma das taxas de transmissão das N interfaces. Dessa forma, o tamanho de cada sub-*cache* é proporcional à capacidade de transmissão de cada interface.

Algoritmo 1: MECANISMO DE DIVISÃO DO CACHE NFACE

Entrada: S, q_i, N, I
Saída: Sub-Cache s_i
início
  $Q \leftarrow \sum_{n=1}^{N} q_n$
  para *cada* $i \in I$ **faça**
    $s_i \leftarrow \frac{S * q_i}{Q}$
  fim
fim
retorna s_i

Com o Cache nFace, somente o CS de um nó é modificado, ou seja, ao invés de um único *cache* agora existe um sub-CS para cada interface. Tanto a FIB quanto a PIT não são alteradas. Além disso, o encaminhamento de pacotes de interesse e de pacotes de dados não se alteram. Durante o encaminhamento de um pacote de interesse, todos os sub-*caches* são verificados para saber se o nó possui uma cópia do pedaço de conteúdo solicitado e não somente o sub-*cache* da interface de recepção do interesse. Os conteúdos continuam sendo encaminhados pelo caminho reverso ao percorrido pelo interesse. A única modificação é que, quando um interesse é atendido por um pacote de dados, o pedaço de conteúdo contido nesse pacote é armazenado no sub-*cache* correspondente à interface que recebeu o interesse. Se um determinado interesse está pendente por mais de uma interface de rede, o pacote de dados ficará armazenado no sub-*cache* correspondente a interface que primeiro recebeu o pacote de interesse.

Vale ressaltar também que um mesmo conteúdo não é armazenado em diferentes sub-*caches*. Para responder um interesse, é verificado se os outros sub-*caches* já possuem

 © 2017 Sociedade Brasileira de Computação

o conteúdo. Portanto, não é encaminhado um novo interesse para um conteúdo que já está em um dos outros sub-*caches*. Assim, impede-se a duplicação de conteúdos no *cache* de um nó. Esse mesmo motivo obriga todos os sub-*caches* a usarem a mesma política de descarte definida para o CS. A premissa para o bom funcionamento do Cache nFace é que a probabilidade de pacotes de interesse usados em um ataque de conluio produtor-consumidor serem recebidos a altas taxas por todas as interfaces de um nó é pequena. Os roteadores de borda geralmente são mais afetados por estarem ligados a uma interface que pertence à rede do atacante. Assim, eles recebem altas taxas de interesses pela interface ligada à rede do atacante. Porém, a probabilidade de que as demais interfaces de rede do roteador de borda pertençam a redes de outros atacantes e/ou recebam tráfego malicioso encaminhado por outros roteadores em altas taxas é pequena. Deste modo, pelo menos uma interface que não pertence à rede do atacante pode responder a interesses legítimos que estão armazenados no seu sub-*cache*. Isso pode acontecer em cenários em que produtores e consumidores maliciosos tenham poucos saltos entre eles e a topologia não possua muitos caminhos alternativos entre eles. Isso indica que posição do atacante dentro da topologia e a quantidade de caminhos entre eles é fundamental para a eficiência do ataque.

5. Resultados

Nessa seção serão discutidas as avaliações dos mecanismos Cache nFace e CacheShield em diferentes cenários. Uma análise dos resultados é realizada para discutir a eficiência de ambos mecanismos sob ataque em conluio. Ainda são definidas as configurações, métricas e topologias utilizadas nas simulações.

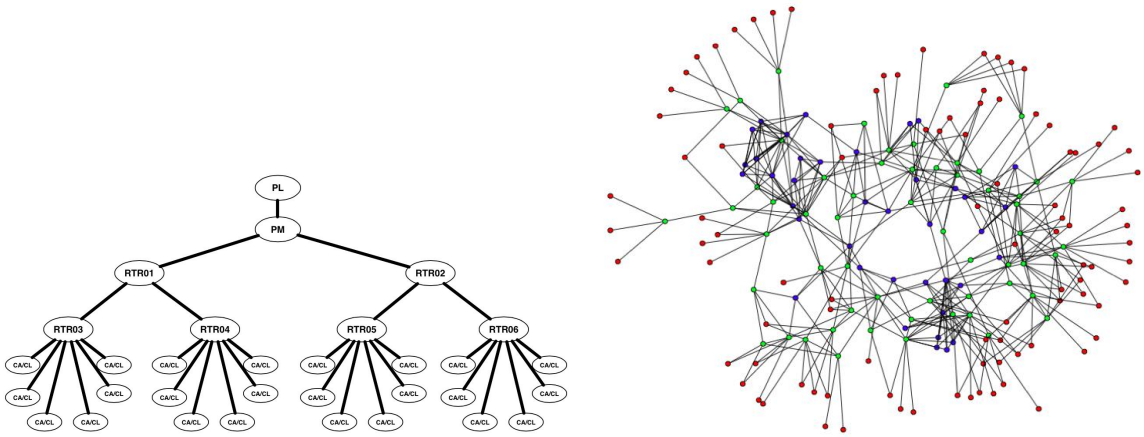
5.1. Cenários de Avaliação

As topologias de rede usadas na avaliação do ataque em conluio e das contramedidas, são as seguintes. Uma topologia composta por 32 nós dispostos em forma de árvore, como mostra a Figura 1(a). Outra topologia em malha, criada com o mapeador de topologias Rocketfuel [Spring et al. 2004], com 192 nós, como mostra a Figura 1(b).

Na topologia em árvore, os 24 nós-folha são consumidores. O número de consumidores legítimos (CL) é fixo em todas as configurações e igual a 16. O número de consumidores maliciosos (CA) varia de 0 a 8. A posição dos CLs e dos CAs é definida aleatoriamente em cada rodada de simulação. O produtor legítimo (PL) é sempre o nó raiz. O produtor malicioso (PA) é o nó filho do nó raiz. Os demais 6 nós que compõem a topologia são os roteadores da rede (RTR). Os enlaces que interconectam os nós possuem taxa de transmissão de 100 Mb/s e atraso de 1 ms.

Na topologia em malha, tem-se 92 nós-folha como consumidores. O número de consumidores legítimos é fixo em todas as configurações e igual a 68. O número de consumidores maliciosos varia de 0 a 24, valores escolhidos para manter a mesma proporção da topologia em árvore. A posição dos consumidores legítimos e consumidores maliciosos é definida, aleatoriamente, em cada rodada de simulação. Existem 2 produtores legítimos e 1 produtor malicioso com posições aleatórias na rede. Os demais 97 nós que compõem a topologia são os roteadores (58 de borda e 39 de núcleo). Os enlaces que interconectam os nós possuem taxa de transmissão entre 124 kb/s e 92 Mb/s e atraso variante entre 1 e 70 ms.

© 2017 Sociedade Brasileira de Computação



(a) 32 nós. (b) 192 nós.

Figura 1. Topologias utilizadas nas simulações.

Os conteúdos são solicitados da seguinte forma. Os consumidores maliciosos enviam interesses para 12 conteúdos que são disponibilizados pelo produtor malicioso a taxas de 10, 100 e 500 interesses por segundo. Como cada pacote, seja de dados ou de interesse, tem tamanho máximo de 65.563 bytes [Kim et al. 2013a, Gallo et al. 2015], cada consumidor malicioso, no pior caso, vai enviar interesses a uma taxa de 65 kbps, 6,5 Mbps e 32,5 Mps respectivamente. Cada conteúdo malicioso possui 100 pedaços (*chunks*) e prefixos de nome diferentes. Os pedaços de conteúdos maliciosos são solicitados de acordo com o consumo sequencial, no qual um consumidor envia pacotes de interesse ordenados e de forma cíclica. Os consumidores legítimos sempre enviam 10 interesses/s para outros 12 conteúdos disponibilizados pelo(os) produtor(es) legítimo(s). Cada conteúdo legítimo possui 100 pedaços (*chunks*) e prefixos de nome diferentes. Os pedaços legítimos são solicitados seguindo uma distribuição Zipf com parâmetro $\alpha = 0,7$ [Breslau et al. 1999].

O *cache* dos consumidores legítimos e dos roteadores tem capacidade para armazenar até 1000 pedaços de conteúdo, e cada pedaço possui 1024 bytes. Os consumidores maliciosos não possuem *cache* para potencializar o ataque, isto é, sempre enviam interesses, independentemente se já receberam o conteúdo anteriormente ou não. A PIT tem tamanho ilimitado para que seja possível avaliar apenas o efeito do aumento da ocupação maliciosa no *cache* dos nós. A política de substituição de *cache* é a *Least Recently Used* (LRU) para todos os sub-*caches* do Cache nFace e para o CacheShield. Os parâmetros da função logística usada pelo CacheShield são $p = 20$ e $q = 1$. De acordo com os autores, esses valores são ideais para as topologias avaliadas [Xie et al. 2012].

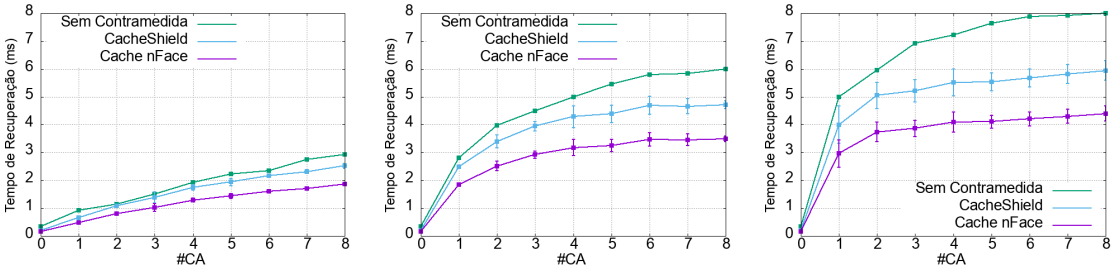
O módulo ndnSIM [Afanasyev et al. 2012] do simulador NS-3 é usado na avaliação. Para cada configuração, são realizadas 50 rodadas de simulação, cada uma com duração de 180 s. Para os pontos dos gráficos obtidos, são calculados intervalos de confiança representados por barras verticais para um nível de confiabilidade de 95%.

5.2. Tempo Médio de Recuperação de Conteúdos Legítimos

As Figuras 2 e 3 mostram o comportamento do tempo médio de recuperação de conteúdos legítimos em função do número de consumidores maliciosos, sendo que as Figuras 2(a), 2(b) e 2(c) referem-se à topologia em árvore, enquanto as Figuras 3(a), 3(b) e

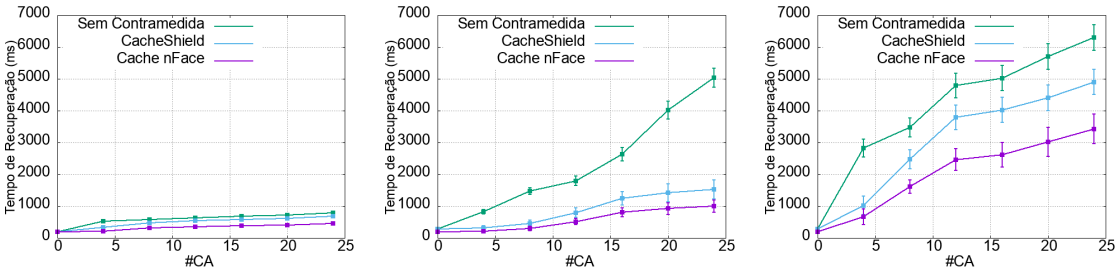
 © 2017 Sociedade Brasileira de Computação

3(c) referem-se à topologia em malha. O comportamento observado é o mesmo: quanto mais consumidores maliciosos, maior o tempo médio de recuperação de conteúdos. Da mesma forma, quanto maior a taxa de interesses maliciosos, maior o tempo médio de recuperação de conteúdos legítimos. Para topologia em árvore observa-se que o mecanismo CacheShield (Figura 2(c)), no pior caso em que atacantes enviam interesses a 500 interesses/s, é capaz de reduzir o tempo de recuperação de conteúdos em 2 ms quando comparado com a rede sob-ataque e sem uso de nenhuma contramedida. Isso representa uma redução de 25% na eficiência do ataque. Observa-se que o mecanismo Cache nFace, por sua vez, reduz em quase 50% a eficiência do ataque, se comparado com a rede sem contamedida, e em cerca de 35%, quando atacantes operam a 500 interesses/s, em relação ao CacheShield. É importante ressaltar que como os consumidores legítimos possuem *cache* e como eles sempre consomem de acordo com a popularidade, pode-se observar que sem ataque o consumo é, muitas vezes, feito do próprio *cache* do nó, o que resulta em um tempo de recuperação inferior a 1 ms.



(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

Figura 2. O tempo de recuperação de conteúdos legítimos na topologia em árvore.



(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

Figura 3. O tempo de recuperação de conteúdos legítimos na topologia em malha.

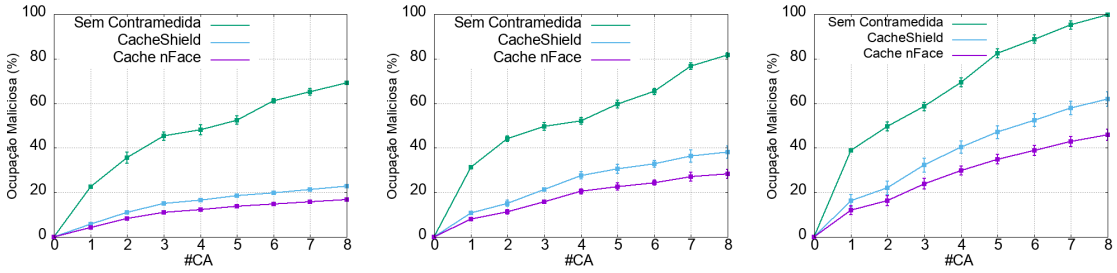
A efetividade do ataque pode ser observada também na topologia em malha. Quando não há contramedida na rede, Figuras 3(a), 3(b), 3(c), tem-se um tempo médio de recuperação de conteúdos 2 vezes maior quando tem-se 24 consumidores maliciosos enviando 10 interesses/s, 12 vezes maior quando enviam 100 interesses/s e 16 vezes maior quando operam a 500 interesses/s, se comparado com o Cache nFace em mesmas condições. Outro resultado interessante para topologia em malha, quando atacantes operam a 500 interesses/s, é que o mecanismo Cache nFace (Figura 3(c)) é cerca de 35% mais eficaz que o CacheShield, e ainda reduz, em cerca de 50%, a eficiência do ataque quando comparado com a rede sem contramedida.

© 2017 Sociedade Brasileira de Computação

5.3. Ocupação Maliciosa dos Caches e Taxa de Erro do Cache

O motivo da contramedida Cache nFace ser mais eficiente é que ela reduz a ocupação maliciosa dos *caches* dos roteadores e, consequentemente, reduz a taxa de erro de *cache* dos conteúdos legítimos se comparada ao CacheShield. Esses resultados são mostrados pelas Figuras 4, 5, 6 e 7. Na topologia em árvore, enquanto tem-se na curva sem ataque da Figura 2(b) o tempo de recuperação de 6 ms para taxa de 100 interesses/s com 8 consumidores maliciosos, tem-se ocupação maliciosa e taxa de erro do *cache* na ordem de 80% (Figura 6(b)). Essa característica implica que, sob ataque, os consumidores devem dar mais saltos para recuperar o conteúdo solicitado. Como, na topologia em árvore, o atraso de cada enlace é de 1 ms, conclui-se que os conteúdos legítimos são recuperados mais frequentemente de nós que estão a mais saltos do consumidor do que os nós de borda. Isso indica que os roteadores de borda estão com uma alta ocupação de conteúdos maliciosos em seu *cache*.

Os mecanismos avaliados reduzem para, aproximadamente 5 ms com o CacheShield e 3 ms com o Cache nFace, com 8 consumidores maliciosos enviando 100 interesses/s. Isso indica que com CacheShield, na topologia em árvore, faz-se em média 3 saltos para recuperar o conteúdo, enquanto com Cache nFace são necessários em média 2 saltos. Quando a rede está sob ataque e não utiliza nenhum dos mecanismos estudados, operando na mesma taxa e quantidade de consumidores do exemplo anterior, tem-se em média 4 saltos. Deste modo, pode-se dizer que o Cache nFace reduz pela metade o tempo de recuperação de conteúdos legítimos nesse cenário, reduzindo a eficiência do ataque.



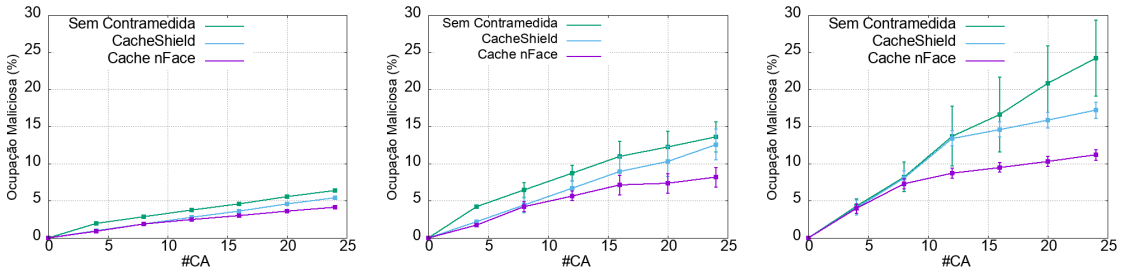
(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

Figura 4. O percentual de ocupação do *cache* dos roteadores por conteúdos maliciosos para as contramedidas avaliadas na topologia em árvore.

Na topologia em malha, Figuras 5(a), 5(b) e 5(c), observa-se uma ocupação maliciosa de cerca de 20% quando existem na rede 20 consumidores maliciosos enviando 500 interesses/s. Portanto, na topologia em malha, se cerca de 10% da rede é composta de consumidores maliciosos, a ocupação maliciosa dos *caches* do roteadores de conteúdo já compromete o consumo de conteúdo legítimo. Por outro lado, o mecanismo CacheShield pode reduzir em cerca de 5 pontos percentuais e o mecanismo Cache nFace em 10 pontos percentuais essa ocupação maliciosa, como se observa na Figura 5(b).

Na topologia de em malha, observa-se uma ocupação maliciosa e taxa de erro do *cache* menor que na topologia em árvore. Isso deve-se ao fato que a topologia em árvore tem a característica de ter o produtor malicioso no caminho do produtor legítimo, obrigando, assim, um consumidor legítimo a utilizar o *cache* de, pelo menos, um nó comprometido para recuperar um conteúdo legítimo. Ainda é possível observar uma maior

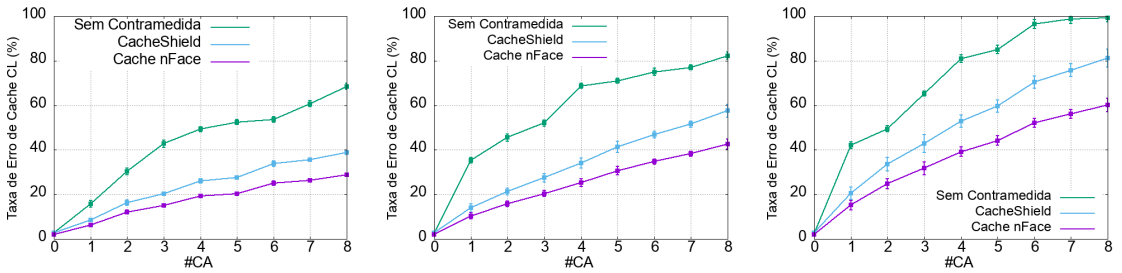
 © 2017 Sociedade Brasileira de Computação



(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

Figura 5. O percentual de ocupação do *cache* dos roteadores por conteúdos maliciosos para as contramedidas avaliadas na topologia em malha.

eficiência do ataque em comprometer o consumo de conteúdos legítimos quando se tem 24 consumidores maliciosos enviando 500 interesses/s. Nessa curva, observa-se uma elevada ocupação maliciosa e taxa de erro do *cache*, como mostrado nas Figuras 5(c) e 7(c) em ordens de 25% e 42% respectivamente. Pode-se concluir que com 24 consumidores maliciosos, que representam 12,65% do total de nós, pode-se aumentar a taxa de erro do *cache*, comprometendo o consumo de conteúdos legítimos. Os mecanismos estudados, porém, reduzem a eficiência do ataque. O CacheShield pode, por exemplo, reduzir a taxa de erro do cache (Figura 7(b)) para 100 interesses/s com 15 consumidores maliciosos em cerca de 40%, enquanto o Cache nFace, na pior configuração (500 interesses/s para 24 consumidores maliciosos) pode reduzir a ocupação maliciosa (Figura 5(c)) também em cerca de 40%. Ou seja, em taxas de 500 interesses/s com 24 consumidores maliciosos, o mecanismo Cache nFace tem a mesma eficiência que o CacheShield em taxa inferior de 100 interesses/s com 24 consumidores maliciosos, se mostrando mais eficaz para reduzir os efeitos do ataque nos cenários avaliados.



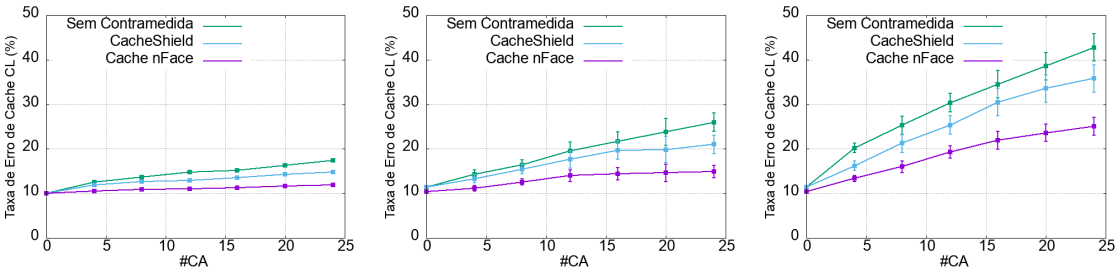
(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

Figura 6. A taxa de erros de *cache* para os conteúdos legítimos para as contramedidas avaliadas na topologia em árvore.

5.4. Conteúdos Recuperados do Produtor Legítimo

As Figuras 8 e 9 mostram o percentual de conteúdos legítimos recuperados do produtor legítimo em função do número de consumidores maliciosos e da taxa de envio de interesses por esses nós. Esses resultados corroboram que o ataque em conluio reduz e eficiência do emprego do *cache* pela CCN. A Figuras 8(a), 8(b) e 8(c) mostram que, se não há ataque, cerca de 0,5% dos conteúdos solicitados são recuperados diretamente do produtor original. Nesse caso, cada conteúdo legítimo é recuperado do produtor original

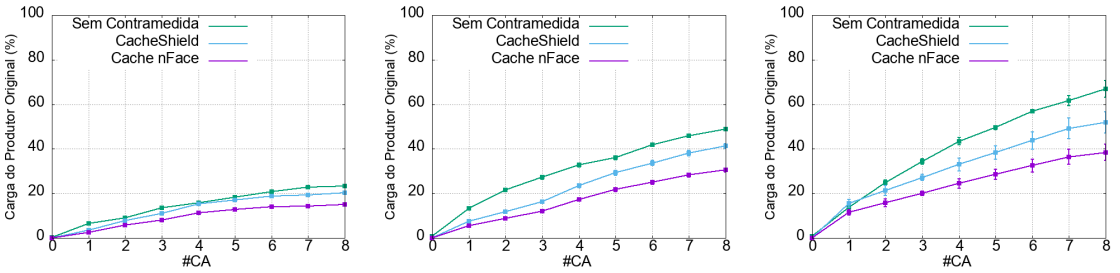
© 2017 Sociedade Brasileira de Computação



(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

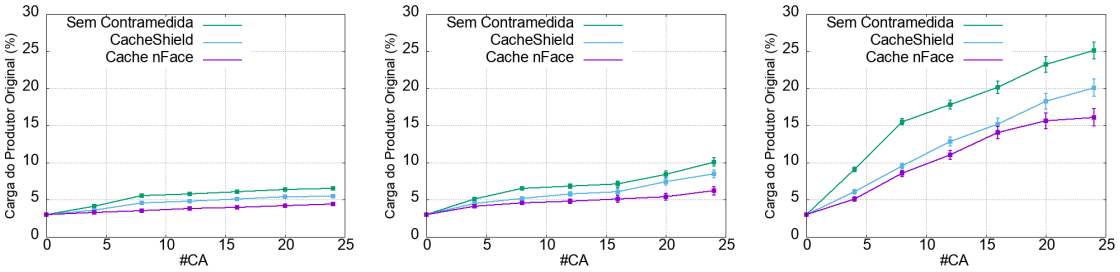
Figura 7. A taxa de erros de *cache* para os conteúdos legítimos para as contramedidas avaliadas na topologia em malha.

no máximo duas vezes, até que seja armazenado pelos nós RTR1 e RTR2. Porém, basta se ter 4 consumidores maliciosos enviando 10 interesses/s (Figura 8(a)) para que esse valor aumente para cerca de 12%. No pior caso, os consumidores legítimos estão recuperando cerca de 67% dos conteúdos legítimos diretamente do produtor legítimo na topologia em árvore. Com os mecanismos Cacheshield e Cache nFace tem-se redução de no carga no produtor legítimo. Isso deve-se ao fato de que roteadores de conteúdo estarem com menos ocupação maliciosa, fazendo aumentar a taxa de acerto do *cache* e recuperando conteúdos legítimos em menos saltos.



(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

Figura 8. Percentual de carga do produtor para as contramedidas avaliadas na topologia em árvore.



(a) 10 interesses/s. (b) 100 interesses/s. (c) 500 interesses/s.

Figura 9. Percentual de carga do produtor para as contramedidas avaliadas na topologia em malha.

É interessante observar que, sob ataque, apenas 4 atacantes, enviando 100 interesses/s, elevam a carga do produtor a cerca de 38% (Figura 8(b)), com CacheShield nas

 © 2017 Sociedade Brasileira de Computação

mesmas configurações tem-se apenas 22%. Para o mecanismo Cache nFace, nas mesmas configurações, observa-se que os mesmos 4 atacantes deixam a carga no produtor legítimo em 19%. O mecanismo Cache nFace mostra-se mais eficiente em reduzir o tempo de recuperação, a ocupação maliciosa, a taxa de erro do *cache* e a carga no produtor que o CacheShield. As características observadas na topologia em árvore podem ser observadas também na topologia me malha para o consumo direto do produtor legítimo, Figuras 9(a), 9(b) e 9(c). Na topologia em malha, ainda tem-se no pior caso, no qual 24 consumidores maliciosos enviam 500 interesses/s, uma carga do produtor legítimo de cerca de 25% sem o uso das contramedidas. Assim, como os pacotes de dados são recuperados diretamente do produtor, a eficiência da rede em entregar conteúdos legítimos mais próximos do consumidor é comprometida. Observa-se, ainda, que o mecanismo CacheShield reduz a carga produtor em cerca de 20%, enquanto o mecanismo Cache nFace reduz a carga do produtor em cerca de 45% nas configurações avaliadas.

Uma característica relevante para a avaliação é que o Cache nFace divide o CS do nó em sub-*caches*. Essa característica é importante porque se um determinado nó, que conta com quatro interfaces de rede, por exemplo, usar o Cache nFace, terá o CS divido em quatro sub-*caches*, que tomam decisões sobre armazenar um conteúdo de forma independente. Isso dificulta a ação do atacante, pois seria necessário afetar todos os quatro sub-*caches* para o nó negar serviço à rede. O funcionamento do Cache nFace impede que um sub-*cache* seja poluído por outro do mesmo nó. Nas topologias estudadas, o mecanismo Cache nFace se mostrou mais eficaz que o CacheShield em todos os cenários avaliados.

6. Conclusão

Nesse trabalho foi proposta uma contramedida para o ataque em conluio produtor-consumidor chamada Cache nFace. O funcionamento do mecanismo baseia-se em criar sub-*caches* em um nó, a partir da divisão do espaço total de armazenamento em *cache* desse nó. Cada sub-*cache* está associado a uma das interfaces de rede do nó.

A contramedida foi comparada com outra encontrada na literatura através de simulação para diferentes topologias e configurações de ataque. Resultados mostraram que a contramedida Cache nFace reduz em cerca de 50% a efetividade do ataque e se mostra mais eficaz nos cenários avaliados que o CacheShield. A contramedida Cache nFace é capaz de comedir o ataque nas configurações avaliadas, reduzindo sua eficiência na topologia malha e na topologia em árvore. Como trabalhos futuros pretende-se avaliar o efeito de aumentar o número de produtores maliciosos para aumentar a probabilidade de ter mais do que uma interface de rede sendo atacada. Ou, ainda, avaliar quantos sub-*caches* de um nó possuem conteúdos maliciosos.

Agradecimentos

Este trabalho é apoiado por Dinter UFF/UFAC, CNPq, CAPES, FAPERJ, Proppi/UFF, TBE/ANEEL e CELESC/ANEEL.

Referências

Afanasyev, A., Mahadevan, P., Moiseenko, I., Uzun, E., and Zhang, L. (2013). Interest flooding attack and countermeasures in named data networking. In *IFIP Networking*, pages 1–9.

© 2017 Sociedade Brasileira de Computação

Afanasyev, A., Moiseenko, I., and Zhang, L. (2012). ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, NDN.

Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. (1999). Web caching and zipf-like distributions: evidence and implications. In *IEEE Conference on Computer Communications - INFOCOM*, pages 126–134.

Brito, G. M., Velloso, P. B., and Moraes, I. M. (2012). Redes orientadas a conteúdo: Um novo paradigma para a Internet. In *Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC*, pages 211–264.

Brito, G. M., Velloso, P. B., and Moraes, I. M. (2013). *Information-Centric Networks, A New Paradigm for the Internet*. FOCUS - Networks and Telecommunications Series. Wiley-ISTE, 1 edition.

Gallo, M., Perino, D., and Muscariello, L. (2015). Content-centric networking packet header format. Technical Report BCP-78, Internet Engineering Task Force.

Gasti, P., Tsudik, G., Uzun, E., and Zhang, L. (2013). DoS and DDoS in named-data networking. In *International Conference on Computer Communications and Networks - ICCCN*, pages 1–7.

Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and Braynard, R. (2009). Networking named content. In *International Conference on emerging Networking EXperiments and Technologies - CoNEXT*.

Kim, J., Shin, D., and Ko, Y.-B. (2013a). Top-ccn: Topology aware content centric networking for mobile ad hoc networks. In *2013 19th IEEE International Conference on Networks (ICON)*, pages 1–6.

Kim, Y., Kim, U., and Yeoml, I. (2013b). The impact of large flows in content centric networks. In *IEEE International Conference on Network Protocols - ICNP*, pages 1–2.

Nasserala, A. and Moraes, I. M. (2015). Artigo 1. In *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pages 641–654.

Nasserala, A. and Moraes, I. M. (2016a). Artigo 2. *IEEE Latin America Transactions*, 14(6):3003–3010.

Nasserala, A. and Moraes, I. M. (2016b). Artigo 3. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 849–852.

Smetters, D. and Jacobson, V. (2009). Securing network content. Technical Report TR-2009-1, Xerox Palo Alto Research Center - PARC.

Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. (2004). Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16.

Xie, M., Widjaja, I., and Wang, H. (2012). Enhancing cache robustness for content-centric networking. In *IEEE Conference on Computer Communications - INFOCOM*, pages 2426–2434.

 © 2017 Sociedade Brasileira de Computação

Privacy-aware web authentication protocol with recovery and revocation

Lucas Boppre Niehues[1], **Ricardo Custódio**[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brazil

lucasboppre@gmail.com, ricardo.custodio@ufsc.br

Abstract. *Password-based authentication is the de facto standard for web services, usually linked to email addresses for account recovery. However, this scheme has several serious known drawbacks. We present a simple pseudonym authentication scheme for users to register and authenticate to online services, by using cryptographic key pairs stored in a mobile device. In contrast to other similar schemes, our proposal requires minimal user interaction, allows for easy account recovery and revocation, and requires no trusted third-party. We created a prototype to evaluate the proposal and concluded that it is viable and has good security, privacy, and usability properties.*

Resumo. *Autenticação baseada em senha é a mais utilizada para serviços web, comumente associada a endereços de email para recuperação das credenciais. Porém, este esquema apresenta várias desvantagens de segurança e privacidade. É apresentado um simples esquema de autenticação pseudônima para serviços online, através de chaves criptográficas assimétricas armazenadas em dispositivo móvel. Em comparação com outros esquemas similares, nossa proposta demanda mínima interação do usuário, permite recuperação e revogação simples das contas, e não exige nenhuma terceira-parte. É criado um protótipo para avaliar a proposta e concluímos que ela é viável e tem boas propriedades de segurança, privacidade e usabilidade.*

1. Introduction

User authentication on the web is a hard problem. It is under competing pressures for usability and security, especially when facing user errors and sophisticated attacks. The de facto standard for general web authentication is the combination of username and password, usually linked to an email address for password recovery purposes [Bonneau et al. 2012].

However, much has been said about the problems of password schemes [Bonneau and Preibusch 2010, Yan et al. 2012]. For example, users have trouble creating and remembering random, high-entropy unique passwords [Bonneau 2012]. Coupled with the difficulty of implementing secure password schemes on the service side, these issues multiply the severity of data breaches.

Numerous alternative authentication schemes have been proposed [Borchert and Günther 2013, Lindemann et al. 2014, Xu et al. 2015]. Although these schemes bring advantages, they usually make sacrifices to security, usability, or privacy. For instance, OpenID and Facebook Login both depend on trusted third parties.

 © 2017 Sociedade Brasileira de Computação

A popular proposal to solve these problems of web authentication is SQRL [Gibson 2016, van Dijk 2014]. However, this scheme has a disaster recovery protocol that is too complex, and the use of deterministic key pairs creates some security concerns.

To remedy these issues we propose a new web authentication protocol with no trusted third parties and no values to memorize, and includes the possibility of account recovery and revocation. Our proposal relies on a mobile device carried by the user to store cryptographic keys for each registered service. This scheme has good security properties and does not require the service to keep any secrets, and neither requires the user to type any name, code or password.

Authentication schemes can also imply privacy concerns. This has been clearly evidenced by the Ashley Madison data breach [Mansfield-Devine 2015], where users were identified by email addresses. Ideally, an authentication scheme should identify users in that service without linking to accounts in different services. We call this property ”pseudonymity” and implement it in our proposed scheme.

This article is organized as follows. The protocol proposed is presented in Section 2. We describe in detail the entire protocol life cycle, considering setup, data storage, the authentication process itself, and disaster recovery (namely account recovery and revocation). In Section 3, we show some scenarios of possible attacks to the proposed protocol. Section 4 compares it to password schemes and to SQRL, which is the closest alternative in literature. Section 5 contains an analysis in terms of strengths and weaknesses of the proposed protocol. We present our conclusions in Section 6.

2. Proposed protocol

This section presents the new privacy aware web authentication protocol. The stages of the protocol life cycle are described, such as setup, data storage, login, the recovery of user data in the event of a disaster, and the revocation, if desired, of the user account in a given service.

The proposed authentication protocol requires minimal user interaction, allows for easy account recovery and revocation, requires no trusted third-party, and implements pseudonymity properties. This is achieved by creating a unique identifier and cryptographic key pair each time the user registers on a new service, plus a disaster recovery key for account recovery and revocation.

The scheme is designed to have three computer systems communicating: the user’s mobile device, the web browser to be authenticated, and the web server hosted by the service. Figure 1 shows the components of the proposed protocol and the corresponding flow of data involved in its execution.

The server authenticates the browser by sending a challenge to be signed by a corresponding private key stored in the mobile device. The offline safe site stores data for recovery and revocation.

2.1. Setup

The first step is to install the mobile application that implements the protocol. When first run, the application generates an asymmetric key pair, referred as *master keys*, and

 © 2017 Sociedade Brasileira de Computação

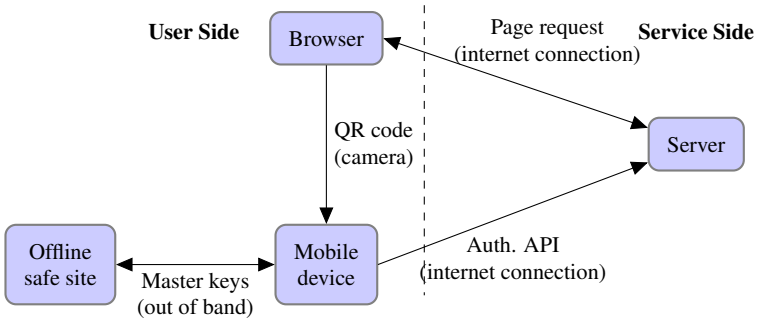


Figure 1. Authentication components.

requests the user to back it up in an offline safe site.

The public key is used to generate user ID, and encrypt the disaster recovery data. The private key is highly sensitive and used only during recovery and revocation. Therefore it is removed from the device once it is backed up.

We label the private key as *disaster recovery key* and the public key as *online master key*. We note that the online master key should be kept secret despite being the ”public” half of the key pair, though it is less sensitive than the private key.

Once the master key pair is created and backed up, the user is ready to register on a service. The protocol steps are shown in Figure 2. For simplicity, we specify this and the following protocols at a high level, without including the exact choice of cryptographic primitives or the message encoding.

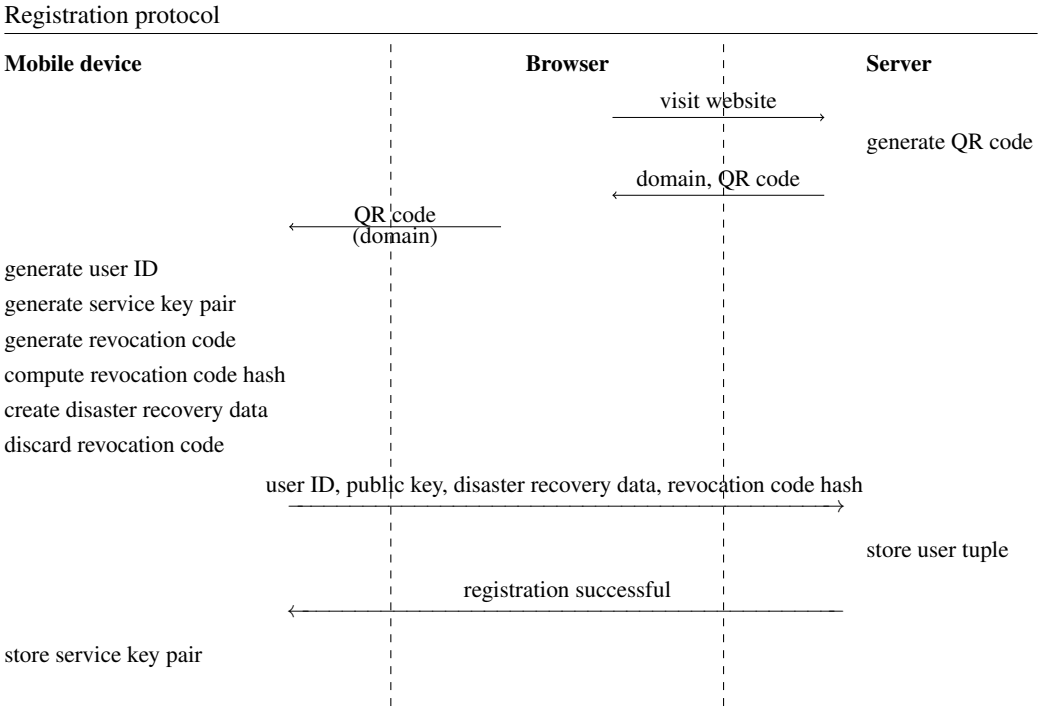


Figure 2. Registration protocol.

The protocol starts with the user visiting the service’s page, which displays a QR

© 2017 Sociedade Brasileira de Computação

code embedding the service domain name. The mobile device then scans the QR code, acquiring the domain name used to uniquely identify the service. The mobile device then generates the following set of credentials using the online master key and the service's domain name:

User ID: identifies the user on a service. The ID is the hash of the *online master key* concatenated with the *service domain name*, that is: $user\,ID = \text{hash}(online\ master\ key \parallel domain)$.

Revocation code: a large random number, to be presented to the service when revoking a key pair. Stored encrypted inside the *disaster recovery data*. This value is created during the registration process but subsequently discarded, requiring the use of the disaster recovery key to recover it. Its hash is stored in the server to verify revocations.

Service key pair: the asymmetric key pair to be used when authenticating to the service. A copy of the key pair is stored encrypted inside the disaster recovery data.

Disaster Recovery Data: It is the *service key pair* together with the *revocation code* encrypted with the *online master key*.

The user then sends the following set of credentials to the server: $\langle$ *user ID*, *disaster recovery data*, *service public key*, and *revocation code hash* $\rangle$. The mobile device only needs to keep the *domain* and *service private key*.

Before accepting the registration, the server must verify if the *user ID* is actually new. The server must not allow a user to register over another user, or replace the credentials stored.

2.2. Storage

Our proposed authentication scheme involves four different components as shown in Figure 3.

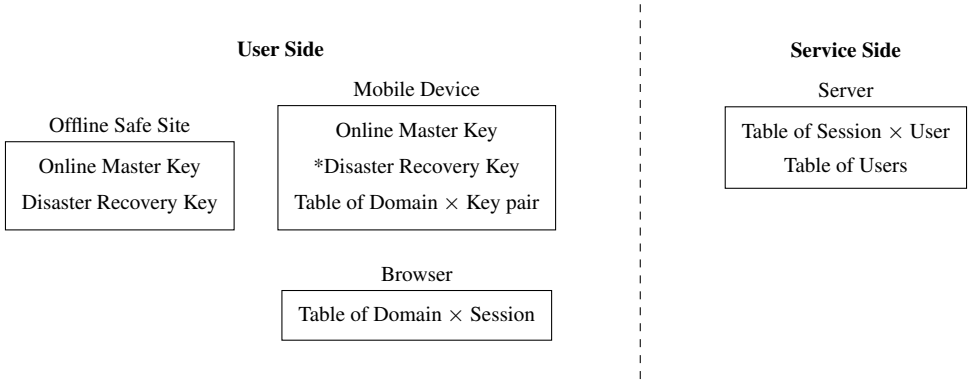


Figure 3. Components of the authentication scheme and the data stored.

The user side has three components: browser, offline safe site, and mobile device. The browser possesses a *session id* cookie for each *domain*. The offline safe site contains the *master key pair* (the *disaster recovery key* and a copy of the *online master key*).

The mobile device stores the *online master key*. The online master key is used to generate the *user ID* based on the *domain*, and to encrypt the *disaster recovery data*

 © 2017 Sociedade Brasileira de Computação

during registration. The online master key should be kept secret to avoid other people registering themselves with it in new services, preventing the original user from using the same id.

The mobile device also stores the table of credentials, which maps the *domain* of each service to the corresponding *key pair*. This table is used during login, using the private key to perform a proof-of-possession protocol. Optionally, it may contain a list of the sessions authenticated by this device, useful to spot attacks and allowing remote log-out.

During revocation and recovery, the mobile device also contains the *disaster recovery key*. Due to its sensitiveness, this key should be discarded from the mobile device after use, and kept only in the *offline safe site*.

The service contains only one component, the web server. To authenticate users it must keep two tables: *users* and *sessions*. The session table associates *hashed session ids* with *user IDs*. This indicates which users are logged-in, and their active sessions (possibly more than one). The session id is sent to the browser as cookie upon its first visit, and discarded from the server afterwards. Note that an attacker with knowledge of a session id can impersonate the corresponding user during that session, which is why only the cryptographic hash of the session id is stored.

The *table of users* maps *user IDs* to their *public key*, *disaster recovery data* and *revocation code hash*. The mobile device supplies all four values during registration, and is subject to change during a revocation. None of these values are secret.

2.3. Login

The login protocol allows the user to authenticate a browser by scanning a QR code using their mobile device camera. Figure 4 shows the messages exchanged during a successful authentication.

The protocol starts when the user points an unauthenticated web browser to the service's website. The server creates a new session id, and responds the browser by sending the session id as a cookie, along with a QR code. The QR code embeds the service domain and a hash of the session id. It is necessary to hash the session id to avoid onlookers from stealing this value.

When scanning the QR code with their mobile device, the user must confirm the service domain. This confirmation prompt is required to protect the user from Man-in-the-Middle attacks. Suppose the user visited the website for a social network, but this service is malicious and wants access to the user's bank service. The malicious server can connect to the bank, posing as a client, and get a QR code for logging in the bank. When the user's browser visits the malicious social network server, it is tricked into displaying the QR code for the bank login, instead of the expected social network QR code. If the mobile device continues with this protocol they will be authenticating the bank session under the malicious server's control. By manually confirming the domain contained in the QR code the user can be sure only the expected service key pair will be used, and no unintended authentications happen.

Once the app has scanned the session id hash and confirmed the domain from the QR code, it fetches the corresponding private key from its local storage. The pri-

© 2017 Sociedade Brasileira de Computação

vate key then signs the session id hash, and the user ID is derived by hashing the online master key concatenated with the domain. The mobile device then sends the triple $\langle$*user ID*, *session id hash*, *signature*$\rangle$ directly to the server, via Wi-Fi or cellular network, through a TLS channel [Dierks and Rescorla 2008].

The server consults its table of users, indexed by the user ID, and retrieves the public key associated. The public key is used to verify the session id hash signature. If the signature is valid, then it represents the user wishes to allow the session (controlled by the browser) to access their data on this service. The server responds with a confirmation and stores the association $\langle$*session id hash*, *user ID*$\rangle$ in the table of sessions. The browser's session is then considered authenticated.

The login may fail if the user ID is not found or if the signature does not match. In this case, the server response to the mobile device indicates the failure, to be displayed to the user. After a successful authentication, the browser will contain a shared secret with the server (session id), the server will have an association between the shared secret and the user (sessions table), and the signature provides unambiguous intent to authenticate that session.

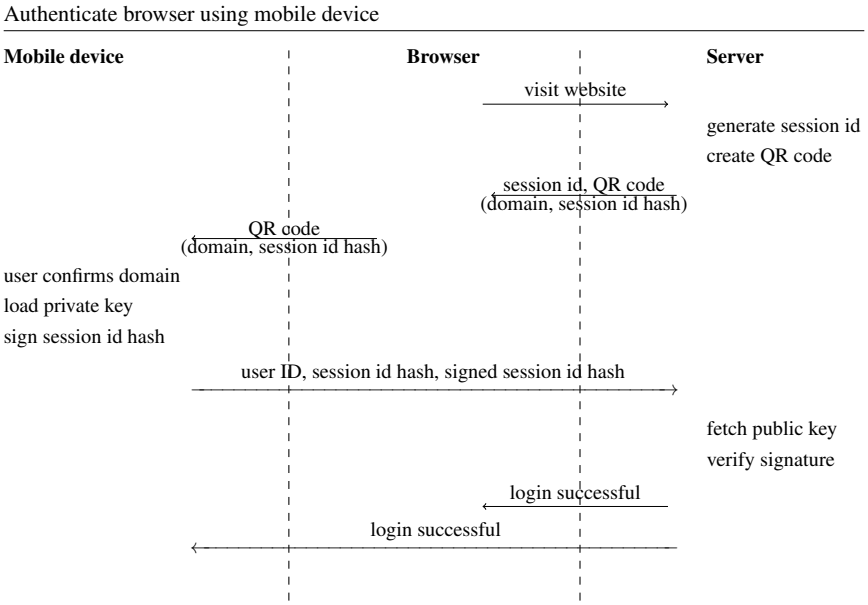


Figure 4. Login protocol.

2.4. Recovery

The authentication protocol depends on data stored in the mobile device. If this data is lost (memory wipe, device destruction or loss) the user must somehow recover access to the services. The recovery process is two step: first, the master key pair must be loaded, then the service key pairs must be recovered from each service. Figure 5 is a diagram of the process.

The master key pair was stored in the offline safe site, set up by the user during the initial enrollment. The user must retrieve this key pair and load the keys in the mobile device. The app can be used to scan a QR code of the keys, or allow the user to type their encoded forms.

 © 2017 Sociedade Brasileira de Computação

Once the online master key and the disaster recovery key (both temporarily loaded into the device) are available, the user inputs the domains of the services they want to recover. For each service the mobile device will generate the *user ID* by hashing the online master key concatenated with the domain, then send a request asking for the *disaster recovery data* associated with this user ID. This value is unique and encrypted, so no authentication is required.

Recall that the disaster recovery data contains the *service key pair* encrypted with the *online* master key, to be decrypted only by the *offline* master key. Since the disaster recovery key is available in the device at the moment, the service key pair is recovered.

During this process, the disaster recovery key is kept loaded in the device. The app must clearly indicate to the user that it's on a recovery state, and that it should be exited as soon as possible. Exiting the recovery state means discarding the disaster recovery key, which brings all components to the same state as before the data was lost.

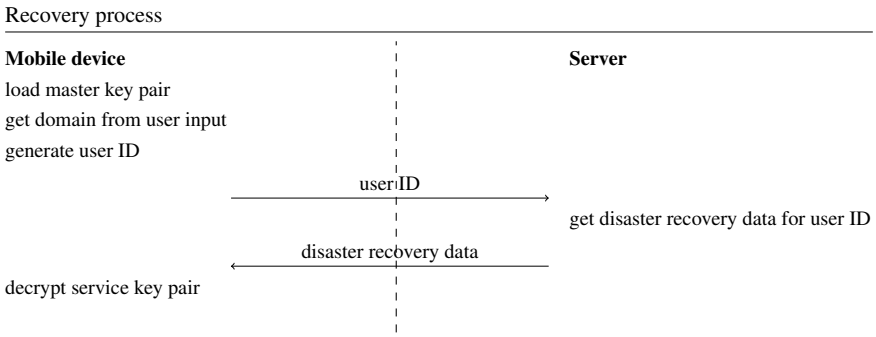


Figure 5. Recovery process.

2.5. Revocation

If the service private keys are *compromised* instead of lost, the user must revoke them. The revocation process is very similar to a recovery followed by a registration. Figure 6 shows the steps required. Due to the similarity of the processes, the steps in common are discussed more superficially.

The revocation process starts with the user loading the *disaster recovery key* in the mobile device and inputting the service domain (if more than one, just repeat the process). The mobile device requests the *disaster recovery data* from the server, and extracts the *revocation code* that was encrypted in it. Up to this point, the process is analogous to a recovery. While the mobile device waits for the disaster recovery data, it also generates the new service key pair, similarly to a registration.

To complete the revocation, the mobile device sends the following tuple to the server: $\langle$ *user ID*, *new public key*, *new disaster recovery data*, *new revocation code hash*, *old revocation code* $\rangle$ (i.e. the same tuple as registration, plus the exposed revocation code). The server can verify the *old revocation code* by hashing it and comparing to the *revocation code hash* stored during registration. If they are the same then the public key, disaster recovery data, and revocation code hash are replaced by the new values received. Additionally, any active session is closed.

© 2017 Sociedade Brasileira de Computação

At this point, the mobile device has a new set of key pairs, and the server values have been updated. The revocation is complete and the *disaster recovery key* is discarded. The security of this process is based on the security of the *disaster recovery key*, which is why it must be stored in an offline safe site.

It is also possible that the *online master key* is compromised, since it is always present in the mobile device. This key is only used to generate user IDs and disaster recovery data, neither of which allows the attacker to impersonate the user. However, the ability to generate user IDs for services allows the attacker to register themselves on new services, with the same user ID that the legitimate user would have, and a set of credentials that the legitimate user will not be able to revoke. This attack blocks the user from using this new service.

The online master key can be replaced by performing the revocation process and, when the mobile device sends the new credentials, including a ”new user ID” generated from a new master key pair. The server would effectively replace all fields for that user (user ID, public key, disaster recovery data, revocation code hash), but since the *old revocation code* is verified this is a valid process.

The only key compromise not recoverable is the compromise of the *disaster recovery key*. In this case, the attacker is able to recover the private key for all services, or revoke them and lock the legitimate user out. This is an unfortunate consequence, but unavoidable for authentication schemes; there is always some ultimate trust. The protocol has been designed to only use the disaster recovery key during emergencies (recovery and revocation), and to not require constant backups of new keys. This enables the disaster recovery key to be kept in a more secure location that is harder to access.

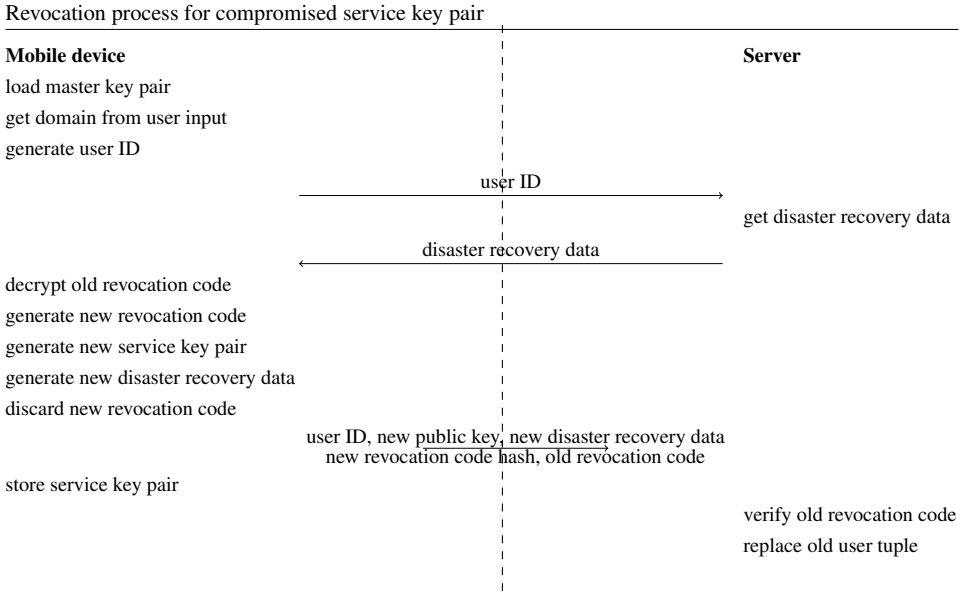


Figure 6. Revocation process to replace credentials.

3. Attack scenarios

Cryptographic schemes are designed with an attacker model in mind, and our proposal is no different. Broadly speaking there are three computer systems (mobile device, browser,

 © 2017 Sociedade Brasileira de Computação

server) plus two environmental factors (network and physical environment of the user). Different assumptions are made for each.

The mobile device is assumed to be secure, but the scheme should have some resilience to physical theft and software compromise. This means that in case one of these events happens, then some security properties may be lost but only temporarily. For example, physical theft may allow the attacker to extract the private keys from the device, and authenticate on the corresponding servers. However, the user is still capable of revoking those keys and taking control back.

The browser is only assumed secure enough to be trusted with one session. Since the user will be using the browser to interact with the service, this assumption must, unfortunately, always be present. Still, the browser is not able to identify the user on unrelated services, or open new sessions by itself in the future.

The server is assumed to be completely untrustworthy for authentication purposes. It may pretend to belong to a different service, forward authentication attempts to steal sessions on a different service, try to identify the user (link accounts), or any other behavior. Additionally, a database leak should not allow attackers to impersonate or identify users.

The network is assumed to also be untrustworthy. The authentication scheme relies on TLS to ensure confidentiality, integrity, and authenticity of the server (domain name matching), even under active attackers. Similarly, the environment around the user is also considered untrustworthy. The scheme should uphold its security properties even if all user interactions are seen by the attacker (e.g. video surveillance recording key presses). Active attacks on the physical environment are not considered, and left as an exercise for spy thrillers.

4. Comparing with other authentication protocols

In this section we compare our proposal to two other authentication protocols: username and password, which is the de facto standard for most of the web; and SQRL, which is the closest protocol to our proposal.

4.1. Username and password

The most common authentication scheme on the web is the username and password combo [Bonneau et al. 2012]. This method requires the user to provide a name, uniquely identifying his account, and a memorized secret to authenticate him. Common additions include registering an email to recover lost passwords, and using the possession of a certain mobile device.

Password authentication schemes have usability problems. For example, account creation is an onerous process, requiring the user to come up with a password that is secure, memorable, unique, and obeys the service's policy, which is often contradictory among different services. Incapable of fulfilling all requirements, users choose low-entropy passwords, which are easy to guess and vulnerable to brute force attacks. Analyzing a dataset of 70 million passwords, [Bonneau 2012] concluded: "As a rule of thumb for security engineers, passwords provide roughly equivalent security to 10-bit random strings against an optimal online attacker trying a few popular guesses for large list of accounts."

© 2017 Sociedade Brasileira de Computação

Additionally, some groups of people have trouble with the creation, memorization, and typing of passwords. For instance, elderly users face special difficulties because of memory and motor skills disabilities [Renaud 2006], while blind users may have trouble interacting with password forms and complying with policies [Saxena and Watt 2009]. These accessibility challenges are active areas of research and improvements are being made, but due to the myriad of individual implementations, many of the password scheme implementations will certainly remain faulty.

Passwords also imply complexity on the service side. Passwords must not be logged, have to be hashed with a special-purpose algorithm before storage (e.g. bcrypt [Provos and Mazières 1999], scrypt [Percival and Josefsson 2016]), and verified with a constant time algorithm. Updating the hash algorithm creates further difficulties, and more advanced security requirements such as "must be different than the last 10 passwords" are common sources of additional complexity. Worsening the problem, there is a plethora of resources online that teach wrong or outdated practices.

Password based authentication schemes also suffer from several security issues. Weak or reused passwords create trivial security holes, and phishing is rampant [Bonneau et al. 2012]. If a user has forgotten their password, the usual alternatives are receiving a password-resetting link by email (creating a single point of failure for all services in the same inbox), or security questions about public or easily guessable personal data [Stuart Schechter 2009]. And if the service mishandles the passwords it receives, attackers can use data breaches to impersonate legitimate users.

4.2. SQRL

There have been a number of proposals for authentication protocols that seek to mitigate these security and privacy challenges. The best representative of these protocols is the SQRL. SQRL [Gibson 2016] is an authentication system with very similar goals and methods, and that ultimately inspired the creation of our scheme. From the point of view of the user, the setup and authentication processes are virtually identical between the two schemes; however, our proposal and SQRL diverge in the underlying cryptographic techniques used, and therefore security properties achieved. We will briefly describe the SQRL scheme, then compare to our proposal.

The SQRL authentication scheme also relies on users scanning QR codes with their mobile devices, and having the mobile device store service key pairs to perform proof-of-possession protocols. The two main differences to our scheme are in the creation of the service key pairs and the revocation facilities. There are plenty of other smaller differences, but they are usually in the level of detail of the client implementation (e.g. exact algorithm, how to protect secret values with passwords, how URLs are processed), or cryptographic differences that do not result in dissimilar security properties (e.g. SHA-256 vs. SHA-512).

In the SQRL protocol all service key pairs are derived deterministically using as seed the value $\mathrm{HMAC}_{\mathrm{SHA256}}(\mathrm{key} = \textit{master key},\ \mathrm{msg} = \textit{domain})$. The (symmetric) master key is derived from a password and a 256-bit pre-master-key value stored in the device. The password is passed through a suitably slow Key Derivation Function, the result of which is XOR'ed with the pre-master-key. A variation of the pre-master-key is also backed up offline to allow account recovery.

 © 2017 Sociedade Brasileira de Computação

The Key Derivation Function used is PBKDF, with different parameters for the pre-master-key in the device and its variation in the offline backup. The online version is protected with parameters that should take half a second to unlock on a standard mobile device, while the offline backup uses parameters for about 60 seconds of processing time. This protection is to avoid brute force attacks in case an adversary acquires access to the physical device or backup.

When first introduced, SQRL lacked a revocation feature. This led to several complaints, and eventually was patched by adding the concept of an ”id lock”. Six keys are involved in the process, forming three asymmetric key pairs, one pair of which is derived from the other two using Elliptic Curve Diffie-Hellman Key Agreement (DHKA). To aid in their description we will use the following notation:

[M]: stored in the user's *mobile* device.
[S]: stored in the service's *server*.
[B]: stored in offline *backup* (analogous to our *offline safe site*).
[-]: not stored. This value is used to derive other data and then safely discarded.

Grouped by key pair ⟨ public, private ⟩, the keys used in the id lock are:

⟨**Identity Lock [M], Identity Unlock [B]**⟩**:** fixed, created during setup.
⟨**Server Unlock [S], Random Lock [-]**⟩**:** created randomly for each service.
⟨**Verify Unlock Key [S], Unlock Request Signing Key [-]**⟩**:** generated by performing either DHKA(Identity Lock [M], Random Lock [-]) or DHKA(Server Unlock [S], Identity Unlock [B]).

During account creation on a new service, the user generates the ⟨Server Unlock [S], Random Lock [-]⟩ pair. Using the Identity Lock [M] key stored on the client it then generates Verify Unlock Key [S] through DHKA(Identity Lock [M], Random Lock [-]). As indicated by the brackets notation the Random Lock [-] key is discarded, while the Verify Unlock Key [S] and Server Unlock [S] are stored in the server.

To perform a revocation the user first loads the Identity Unlock [B] from offline backup and requests Server Unlock [S]. Together they form DHKA(Server Unlock [S], Identity Unlock [B]), which is able to generate the Unlock Request Signing Key [-], whose public part is the Verify Unlock Key [S] and was stored in the server. Thus only the owner of the offline backup can sign revocation requests, and the server can verify them.

4.3. Comparison of SQRL to our proposal

Both protocols provide pseudonymous authentication in different services without third parties, mainly by using a mobile device to read a QR code, and allow for account management in those services. Also, both protocols require the backup of some important secret key in offline storage.

The main security difference is that SQRL's service key pairs are generated deterministically based on a master key that is stored in the device. We propose random service key pairs, later stored encrypted in the server (namely inside the *disaster recovery data*).

If the device is compromised, SQRL forces the user to revoke the master key. This means revoking all accounts in all services. Our proposed scheme requires revocation of

© 2017 Sociedade Brasileira de Computação

only the keys that were present in the device. It is still suggested to revoke the master key pair to avoid account collisions, but this is a nuisance instead of a security danger.

Worse still, the compromise of SQRL's master key, that is kept in the device (albeit under a password) compromises all past and future key pairs based on that master key. This means that a one-time attack that goes undetected will expose accounts created after the compromise. An attacker would have total control on all services until the user changes the master key for other reasons, potentially never.

Our equivalent key is the *online master key* (the one kept in the device). If it is compromised, the only attack on future accounts is to pre-register that user ID. This does not compromise those future accounts, and it reveals to the legitimate user that their online master key was attacked. Similarly, service key pairs not stored in that mobile device are also safe. This is because the only way to recover (or take over) an account is having access to the disaster recovery key.

Additionally, SQRL does not allow changing one individual service key pair, since all such key pairs are derived deterministically from the fixed master key. This property discourages the user from using the scheme in more flexible ways, such as sharing less sensitive service key pairs, or storing them in simpler devices (e.g. thumb drives).

The drawback of our proposed method is that if the user wishes to use more than one mobile device (or a mobile device and a desktop client), the synchronization of service key pairs must be done actively. In SQRL this is accomplished by simply sharing the master key, from which all service key pairs are derived. Naturally, the value of this feature must be weighed against the security repercussions.

An important property, unrelated to security, is the simplicity of the scheme. SQRL's scheme uses one symmetric master key, service key pairs, plus three distinct key pairs for revocation, and a non-trivial application of Elliptic Curve Diffie-Hellman Key Agreement. Our proposal has one master key pair, service key pairs, and a revocation code. We believe this simplicity is fundamental for achieving a wider adoption.

5. Analysis of the proposed protocol

An evaluation of the proposed protocol was made, considering several properties of cryptographic protocols commonly used in the analysis of authentication protocols. Table 1 shows the list of desirable properties that our proposal meets.

Table 1: Desirable Properties of our Proposed Protocol.

Property	Analysis
Memoryless	There are no values to be memorized by the human user, secret or otherwise.
Leak resistant	Access to a server's leaked database confers no advantage in impersonating them to the service it was leaked from, or identifying users across services.
Should-surfing and keylogger resistant	Secrets are handled inside the trusted device. The user never interacts directly with any secret.
Pseudonymity	The same user has unrelated identities in each service.
Timeless backups	The user does not have to update the backup after registering or revoking a service. This allows the disaster recovery key to be stored in a way that is safer but harder to access.

 © 2017 Sociedade Brasileira de Computação

Updatable keys	The keys to a service can be changed without changing the master key. This is a trivial property for most schemes, but hard to achieve without updating backups.
No third-party	The scheme does not rely on any trusted third party, even during revocation.
Phishing resistant	It is resistant to Phishing attacks since the domain is explicitly part of the authentication process.
Scalable	Authenticating with a service has the same difficulty independent from how many accounts the user has. Similarly, authenticating one user costs the same for the server regardless of the total number of users registered.
Low friction	Users don't have to type any name, password, or code, and once authorized the login is nearly instantaneous.
Flexible	The user may self-impose a PIN, password or biometric check; detached keys allow usability-security trade-offs; key pairs can be stored with different security levels. This also applies to the disaster recovery key.

Unfortunately, it also has a few downsides. Table 2 shows these drawbacks.

Table 2: Undesirable Properties of our proposed Protocol.

Property	**Analysis**
Device carry	The cryptographic key pairs have to be kept in a personal trusted device. The loss of this device can bring difficult problems.
Backup required	The user has to safely back up the disaster recovery key.
User unfamiliarity	The authentication process has less friction, but it's unfamiliar to users. This increases the rate of errors and makes users more susceptible to social engineering attack.

6. Conclusion

Web authentication is usually based on passwords, and to some extent, emails. These protocols, however, have security, privacy and usability flaws. The requirement of having high-entropy unique passwords is at odds with human memory, enabling various kinds of attacks, and the manual input aspect is responsible for the creation of innumerous types of phishing. The use of an email address for disaster recovery leads to single-point-of-failure scenarios and needlessly reveals the user's identity. Lastly, passwords are a constant source of headaches for developers, administrators and the users themselves.

The SQRL protocol is a popular proposal to this problem, making use of ubiquitous smartphone possession. By using a mobile device to scan a QR code, the protocol enables users to log in without any usernames or passwords, and a much higher level of security. By using an offline backup instead of email account, privacy and single-point-of-failure concerns are also addressed.

However, the SQRL protocol creates cryptographic key pairs deterministically based on a master key, which is kept on the mobile device. This makes the protocol inflexible, and creates security concerns. Additionally, the disaster recovery protocol (revocation + recovery) is extremely complex, complicating its analysis and implementation.

To remedy these flaws, a new protocol for authentication on the web was introduced. From the point of view of a user, our proposal is similar to the SQRL protocol. However, our proposal creates random key pairs and is much simpler during disaster recovery.

© 2017 Sociedade Brasileira de Computação

Compared with other authentication schemes, we find that our proposal is one of the best candidates for wide adoption on the web. It has vastly superior security properties than pure password schemes, several advantages against more sophisticated schemes, and improves significantly on similar proposals.

The proposed protocol, described in this work, was implemented and verified in practice. The code containing its implementation can be obtained in `https://github.com/boppreh/frango-app`.

References

Bonneau, J. (2012). The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552. IEEE.

Bonneau, J., Herley, C., Van Oorschot, P. C., and Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy*, pages 553–567. IEEE.

Bonneau, J. and Preibusch, S. (2010). The password thicket: Technical and market failures in human authentication on the web. In *WEIS*.

Borchert, B. and Günther, M. (2013). Indirect nfc-login. In *ICITST*, pages 204–209.

Dierks, T. and Rescorla, E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard). Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685.

Gibson, S. (2016). Sqrl. `https://www.grc.com/sqrl/sqrl.htm`.

Lindemann, R., Baghdasaryan, D., and Tiffany, E. (2014). Fido universal authentication framework protocol. *Version v1. 0-rd-20140209, FIDO Alliance, February*.

Mansfield-Devine, S. (2015). The ashley madison affair. *Network Security*, 2015(9):8–16.

Percival, C. and Josefsson, S. (2016). The scrypt Password-Based Key Derivation Function. RFC 7914 (Informational).

Provos, N. and Mazières, D. (1999). Bcrypt algorithm. In *Proceedings of 1999 USENIX Annual Technical Conference*. USENIX.

Renaud, K. (2006). A visuo-biometric authentication mechanism for older users. In *People and Computers XIX—The Bigger Picture*, pages 167–182. Springer.

Saxena, N. and Watt, J. H. (2009). Authentication technologies for the blind or visually impaired. In *Proceedings of the USENIX Workshop on Hot Topics in Security (HotSec)*, volume 9, page 130.

Stuart Schechter, A.J. Brush, S. E. (2009). It's no secret: Measuring the security and reliability of authentication via 'secret' questions. In *Proceedings of the 2009 IEEE Symposium on Security and Privacy*, Berkeley, CA, USA. IEEE Computer Society.

van Dijk, J. (2014). A closer look at sqrl. Technical report, University of Amsterdam.

Xu, F., Han, S., Wang, Y., Zhang, J., and Li, Y. (2015). Qrtoken: Unifying authentication framework to protect user online identity. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pages 368–373. IEEE.

Yan, Q., Han, J., Li, Y., and Robert DENG, H. (2012). On limitations of designing usable leakage-resilient password systems: Attacks, principles and usability. In *19th Network and Distributed System Security Symposium (NDSS)*.

 © 2017 Sociedade Brasileira de Computação

Análise de Vulnerabilidade de Esquemas de Segredo Compartilhado Considerando um novo Modelo de Ameaça

Rick Lopes de Souza, Martin Vigil, Ricardo Felipe Custódio

[1]Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

Abstract. *Secret sharing schemes are cryptographic primitives used to distribute parts of a secret among a set of participants in such a way that only an authorized subset can rebuild the secret. Traditionally, most secret sharing schemes work with only two entity definitions: The Dealer and Participants. For these schemes, several threat models have been proposed considering only these two entities. However, in the literature it is not clear who should play the role of rebuilding the secret and who would be the keeper of the secret. These two new entities modify the existing threat models that consider: (i) the distributor initializes the system and splits and shares the secret, and (ii) the participants cooperate or not with the secret reconstruction. In this work we propose a new threat model considering the new entities that represent the roles involved in shared secret schemes. Considering this new model, we show that some of the best-known schemes are vulnerable. We make evaluations checking which points can be safely maintained and at which points vulnerabilities will emerge according to the new entities and new threat model.*

Resumo. *Esquemas de segredo compartilhado são primitivas criptográficas utilizadas para distribuir partes de um segredo entre um conjunto de participantes de tal forma que apenas um subconjunto autorizado consiga reconstruir o segredo. Tradicionalmente, grande parte dos esquemas de segredo compartilhado trabalham com apenas duas definições de entidades: Distribuidor e Participante. Para esses esquemas, diversos modelos de ameaça foram propostos considerando apenas essas duas entidades. Todavia, na literatura não está claro quem deveria exercer o papel de reconstruir o segredo e quem seria o detentor do segredo. Essas duas novas entidades modificam os modelos de ameaça existentes que consideram o Distribuidor a entidade responsável por inicializar o esquema, gerar parâmetros do sistema, gerar o segredo, criar as partes e distribuir, e Participantes que cooperam ou não com a reconstrução do segredo. Neste trabalho propomos um novo modelo de ameaça considerando as novas entidades que representam todos os papéis envolvidos em esquemas de segredo compartilhado. À luz desse novo modelo, alguns dos esquemas mais conhecidos mostram-se vulneráveis. Também são feitas avaliações verificando quais pontos consegue-se manter segurança e em quais pontos surgirão vulnerabilidades de acordo com as novas entidades e modelos de ameaça definidos.*

1. Introdução

Um esquema de segredo compartilhado é um método para distribuir um segredo para um conjunto de participantes dando para cada um apenas uma parte, de tal forma que

 © 2017 Sociedade Brasileira de Computação

apenas um conjunto específico de participantes possa reconstruir o segredo reunindo as partes. Esquemas de segredo compartilhado são considerados serviços básicos em diversos protocolos criptográficos. Podem ser utilizados para procedimentos de autenticação, replicação de arquivos, contingência de dados sensíveis e computação distribuída.

Esses esquemas envolvem diferentes entidades que possuem funcionalidades específicas dependendo do uso. A literatura normalmente considera apenas duas entidades: Distribuidor e Participantes. O Distribuidor tem a responsabilidade de receber um segredo, quebrar em partes e distribuir de forma segura para um conjunto de Participantes. Os Participantes devem guardar de forma segura e seguir os protocolos honestamente para fornecer e reconstruir o segredo.

Partindo das entidades definidas anteriormente, os esquemas de segredo compartilhado clássicos como o de Shamir [Shamir 1979] consideram um modelo de ameaça tradicional, onde o Distribuidor é uma entidade confiável, que gera os parâmetros, o segredo e distribuir de forma segura para os Participantes. Os Participantes comportam-se de maneira polarizada. Ou seja, ou são completamente honestos (participam da reconstrução do segredo de forma correta) ou são maliciosos (não participam na reconstrução do segredo, atrapalhando o protocolo).

O modelo de ameaça mencionado anteriormente apresenta algumas premissas razoáveis para algumas situações, entretanto, não necessariamente reflete os diferentes cenários de aplicações existentes. Por isso, diversos outros trabalhos foram propostos descrevendo novos modelos de ameaças e novos esquemas de segredo compartilhado para garantir a segurança com essas variações.

Problema: Mesmo com diversos trabalhos propostos na literatura variando os modelos de ameaça, encontra-se uma limitação na definição das entidades. Refere-se na grande maioria que o Distribuidor é quem gera o segredo e os parâmetros, fazendo com que seja uma entidade parcialmente confiável. Mas sabe-se que esse nem sempre é o cenário encontrado nas aplicações, onde existe um Detentor do Segredo que deseja compartilhar um determinado segredo com mais participantes utilizando o protocolo de esquema compartilhado sem necessariamente confiar no Distribuidor. Além disso, em grande parte da literatura não se encontra de forma clara qual entidade deve ser responsável pela reconstrução do segredo.

Motivação: A confiabilidade de esquemas de segredo compartilhado dependem dos procedimentos de segurança que utilizam. Logo a avaliação dos modelos de ameaça nos protocolos de segredo compartilhado que são utilizados para os mais diversos motivos são essenciais. A literatura não comenta de forma mais específica sobre qual entidade deve ser responsável por reconstruir o segredo, assim como não comenta sobre qual entidade deveria gerar o segredo além do Distribuidor. A criação de definição de novas entidades muda alguns modelos de ameaça, fazendo com que alguns dos principais esquemas existentes possam não ser mais confiáveis em determinadas situações. Dessa forma, faz-se necessário uma avaliação desses esquemas em um novo modelo de ameaça para verificar se estariam seguros ou não.

A partir do problema e da motivação, conclui-se que é necessário um estudo nos esquemas de segredo compartilhado para avaliar qual seria o impacto da criação de uma novo modelo de ameaça baseado na criação de novas entidades.

 © 2017 Sociedade Brasileira de Computação

Contribuição: Este artigo tem como principais contribuições:

1. Criação de uma nova entidade para os modelos de ameaça
2. Criação de um novo modelo de ameaça com a nova entidade
3. Avaliação do impacto nos protocolos de segredo compartilhado com base no novo modelo de ameaça
4. Possíveis correções e medidas de controle para os problemas encontrados a partir da avaliação feita

Este artigo está organizado da seguinte forma: primeiro alguns dos esquemas tradicionais de segredo compartilhado são introduzidos, depois as princpais entidades dos esquemas de segredo compartilhado são detalhadas. A partir disso, define-se o modelo de ameaça tradicional, assim como os modelos de ameaça não tradicionais. Como parte das contribuições do artigo, um novo modelo de ameaça é definido e avaliado. Ao final do trabalho estão as considerações finais.

2. Esquemas tradicionais de Segredo Compartilhado

Esquemas de segredo compartilhado são considerados serviços básicos em determinados protocolos criptográficos. Podem ser utilizados para procedimentos de autenticação, replicação de arquivos, contingência de dados sensíveis e etc.

Esses esquemas normalmente definem duas entidades: Distribuidor e Participantes. Os Distribuidores tem como principal função a geração dos parâmetros, quebrar o segredo em partes e distribuí-los para os Participantes. Os Participantes então devem guardar em segurança e cooperar no momento da reconstrução.

Os Participantes fazem parte de um conjunto chamado Estrutura de Acesso do esquema de segredo compartilhado. Beimel [Beimel 2011] provê uma definição formal das Estruturas de Acesso:

Definição 2.1 (Estrutura de Acesso)**.** Seja $P = \{p_1, \ldots, p_n\}$ um conjunto de participantes. Uma coleção $\mathcal{A} \subseteq 2^{\{p_1,\ldots,p_n\}}$ é monotônica se $B \in \mathcal{A}$ e $B \subseteq C$ implica que $C \in \mathcal{A}$. Uma estrutura de acesso é uma coleção monotônica $\mathcal{A} \subseteq 2^{\{p_1,\ldots,p_n\}}$ de subconjuntos não vazios de $\{p_1, \ldots, p_n\}$. Conjuntos pertencentes a $\mathcal{A}$ são chamados autorizados e conjuntos não pertencentes a $\mathcal{A}$ são chamados não autorizados.

As duas principais propriedades de um esquema de segredo compartilhado podem ser definidas da seguinte forma. Seja P um conjunto de participantes, seja $\mathcal{A}$ a estrutura de acesso monotônica em P, seja K o conjunto de possíveis segredos. Dessa forma, os esquemas devem possuir as duas seguintes propriedades:

1. **Corretude**. Qualquer subconjunto autorizado consegue reconstruir o segredo.
2. **Privacidade Perfeita**. Qualquer subconjunto não autorizado não consegue obter nenhuma informação a respeito do segredo.

Existem diferentes construções de esquemas de segredo compartilhado ([Beimel 2011] para um survey). Um dos mais importantes esquemas já propostos é o de Shamir [Shamir 1979] descrito a seguir.

O esquema clássico de Shamir possui as propriedades de Corretude e Privacidade Perfeita como estabelecida na definição (referenciar definição). O esquema estabelece

© 2017 Sociedade Brasileira de Computação

dois parâmetros t e n, tal que $t, n \in \mathbb{Z}$, onde $0 < t \leq n$. O segredo $k \in \mathbb{Z}_p$, onde $\mathbb{Z}_p$ é um anel com p primo, é dividido em n partes de tal forma que ao menos t é possível reconstruir o segredo. O esquema é baseado em interpolação de polinômios, onde dado t diferentes pontos $(x_1, y_1), \ldots, (x_t, y_t)$, existe apenas um único polinômio $Q(x)$ de grau $t-1$ onde $Q(0) = k$.

O esquema funciona como descrito a seguir. Dados t, n e um segredo $k \in \mathbb{Z}_p$, a entidade chamada Distribuidor precisa escolher um polinômio $Q(x) = a_0 + a_1 x + ... + a_{t-1}x^{t-1}$ tal que $a_0 = k$ e $a_1, \ldots, a_{t-1} \in \mathbb{Z}_p$ são coeficientes escolhidos de forma aleatória utilizando uma distribuição uniforme.

As partes são compostas por (x_i, y_i), para $i = (1, 2, \ldots, n)$, onde x_i é gerado aleatoriamente e cada avaliação $y_i = Q(x_i)$ é calculada. Após esta etapa, diferentes partes são distribuídas para cada Participante. Recebendo quaisquer t ou mais partes, o Reconstrutor pode encontrar o polinômio $Q(x)$ por meio de interpolação e então recuperar o segredo com a operação $k = Q(0)$.

3. Entidades

O esquema de segredo compartilhado depende de algumas entidades para funcionar de maneira correta. Isto significa que cada entidade deve ter suas funções corretamente definidas, assim como suas limitações. Não se encontra na literatura definições muito concretas sobre cada entidade, assim como suas limitações. Na grande maioria dos casos nota-se falta de informação sobre a entidade que gera o segredo e a entidade que reconstrói o segredo. Abordaremos aqui todas as possíveis entidades envolvidas em um esquema de segredo compartilhado.

1. **Detentor do Segredo:** O Detentor do Segredo é uma entidade que gera o segredo e tem como principal função gerar um segredo de forma aleatória e segura. Após essa geração, deve-se repassar o mesmo para que uma outra entidade possa quebrar e compartilhar com outras entidades. Partimos da premissa que o Detentor do Segredo deve ser sempre confiável, pois ele é o Detentor do Segredo que quer compartilhando utilizando algum esquema de compartilhamento.
   Opcionais:
   (a) *Quebrar o segredo e distribuir:* O Detentor do Segredo pode também ser um Distribuidor (que será apresentado a seguir). Esta entidade irá quebrar o segredo e compartilhar com os participantes de forma segura.
   (b) *Confiável:* O Detentor do Segredo é confiável. Ou seja, o mesmo sempre gerará o segredo de uma forma aleatória seguindo alguma distribuição uniforme e entregará ao Distribuidor utilizando um canal de comunicação seguro.
2. **Distribuidor:** O Distribuidor é uma entidade que tem como principal função receber o segredo de um participante ou uma terceira parte, quebrá-lo e distribuí-lo para os participantes por meio de um canal seguro. **Opcionais:**
   (a) *Gerar o segredo* - Dado um conjunto de possíveis segredos K, irá sortear de forma aleatória um segredo $k \in K$.
   (b) *Gerar os parâmetros do sistema* - Em determinados tipos de esquemas de segredo compartilhado, deve-se gerar os parâmetros para o compartilhamento. Entre eles:

 © 2017 Sociedade Brasileira de Computação

- Tamanho total do grupo
- Tamanho do limiar
- Quantidade de níveis do esquema

(c) *Confiável:* O Distribuidor é completamente honesto. Ou seja, não pode ser corrompido por ninguém e seguirá corretamente os protocolos e regras.

(d) *Malicioso:* Esta entidade irá se comportar de forma maliciosa. Ou seja, irá gerar partes falsas que não deixarão recuperar o segredo. Poderá também manipular a geração de partes tal que menos participantes consigam recuperar o segredo. O Distribuidor irá se comportar como mencionado na definição 2 com os opcionais 1 e 2.

3. **Participante:** Um participante é um membro que pertence a uma estrutura de acesso do esquema de segredo compartilhado. Este membro recebe uma parte do segredo e tem como principal função guardá-lo de forma segura. Deve também seguir os protocolos de maneira honesta na reconstrução do segredo.
   Opcionais:
   (a) *Agir como um Distribuidor* - Agindo como um Distribuidor, o participante pode receber o segredo de uma terceira entidade ou gerá-lo sozinho. O participante então quebra esse segredo e distribui para os outros participantes.
   (b) *Agir como um Reconstrutor* - O participante consegue receber um conjunto de partes dos participantes para reconstruir o segredo.
   (c) *Confiável:* Os participantes irão executar os protocolos de maneira correta. Ou seja, recebem suas partes, mantém em segurança e participam do processo de reconstrução quando necessário.
   (d) *Malicioso:* O participante pode ser comprometido por um adversário e tentará subverter o protocolo.
4. **Reconstrutor:** Um reconstrutor tem como principal função receber as partes dos participantes e reconstruir o segredo. Este segredo será então utilizado para o seu propósito final.
   Opcionais:
   (a) *O Reconstrutor é também um Distribuidor* - Em determinadas situações necessita-se que exista apenas uma entidade que distribua e receba as partes. Neste caso, centraliza-se a segurança do esquema em apenas uma entidade que possui dois papéis.
   (b) *O Reconstrutor é também um Participante* - Existem casos onde é necessário que um dos participantes assuma o papel de reconstrutor. Ou seja, todos as partes serão reveladas para esta entidade reconstruir o segredo.
   (c) *O reconstrutor é uma terceira parte* - Uma terceira parte, que não é o Distribuidor ou Participantes, pode agir como um reconstrutor para receber as partes e reconstruir o segredo.
   (d) *Confiável:* O Reconstrutor é confiável e recebe todos as partes necessárias para reconstruir o segredo. Após a reconstrução, o reconstrutor deve encaminhar o segredo para a entidade Detentor do Segredo e destruir o segredo reconstruído localmente.
   (e) *Malicioso:* O reconstrutor é malicioso e pode querer corromper o esquema enviando o segredo recuperado para um adversário.

© 2017 Sociedade Brasileira de Computação

5. **Adversário:** Um adversário tem como principal objetivo obter informação a respeito do segredo. Ou seja, esta entidade quer de alguma forma atrapalhar o protocolo estabelecido para ganhar acesso ao segredo. Isto pode ser feito de diversas formas, dependendo do tipo de papel que o adversário tem no esquema.
   Opcionais:
   (a) *O adversário é o Distribuidor - Um Distribuidor malicioso pode ser o adversário do esquema. Tal entidade pode gerar parâmetros errados, assim como partes incorretas ou inserir dados suspeitos na criação dos esquemas. Caso tais informações não sejam geradas seguindo de forma correta os protocolos, o esquema pode ser comprometido.*
   (b) *O adversário é um Participante* - Um participante pode ser um adversário para o esquema. Tal participante pode ter sido comprado por uma entidade externa ao esquema e tentar prejudicar a recuperação do segredo. Assim como pode também tentar descobrir o segredo sem a autorização do resto do grupo.
   (c) *O adversário é uma Terceira Parte* - Uma terceira parte é uma entidade que não faz parte do esquema ou da estrutura de acesso, mas pode tentar obter alguma informação sobre as partes ou do segredo.
   (d) *Passivo:* Um adversário pode capturar as partes, entretanto, o protocolo é executado de forma correta e as partes não são corrompidas.
   (e) *Ativo:* Pode tomar a forma de um participante e submeter uma parte falsa durante o processo de reconstrução. Isso previne os participantes de conseguir reconstruir o segredo correto.

4. Modelo de Ameaça Tradicional

Para melhor entender os esquemas, precisa-se primeiramente especificar o modelo de ameaça tradicional. Após essa especificação, pode-se então especificar os modelos de ameaça diferentes e que não estão bem documentados na literatura.

O modelo de ameaça tradicional é o esquema em que todas as entidades se comportam de maneira honesta [Martin 2008]. Ou seja, todos irão gerar seus dados de forma aleatória utilizando uma distribuição uniforme e vão seguir os protocolos de maneira correta. Existem adversários que vão tentar obter as partes buscando por falhas de segurança nas comunicações ou vão tentar comprometer algum participante para subverter o protocolo. O Distribuidor se comporta de forma honesta, gera os parâmetros, segredo e distribui as partes para os Participantes. Os Participantes se comportam de maneira polarizada. Ou seja, recebem as partes, guardam de maneira segura e participam ou não no processo de reconstrução. O Reconstrutor não é bem definido pelo modelo, ficando aberto se um dos participantes poderia assumir esse papel ou uma terceira parte confiável. A tabela 1 ilustra a relação do modelo de ameaça tradicional com as entidades envolvidas.

Tabela 1. Modelo de Ameaça Tradicional

	Confiável	**Malicioso**	**Passivo**	**Ativo**
Distribuidor	✓			
Participante	✓			
Reconstrutor	✓			
Adversário			✓	

 © 2017 Sociedade Brasileira de Computação

Os esquemas tradicionais de Shamir [Shamir 1979] e Blakley [Blakley 1979] suportam esse modelo de ameaça e conseguem garantir segurança nos protocolos. Ambos protocolos definem como premissas que as entidades são confiáveis e que todo o protocolo funcionará de maneira correta, sem interferências.

5. Modelos de Ameaça Não Tradicionais

Sabemos que o modelo de ameaça tradicional não é sempre possível devido às diferentes circunstâncias de uso dos esquemas de segredo compartilhado. Nem sempre é possível garantir total segurança em todas as etapas em cenários reais. Um dos primeiros ataques publicados na literatura, um adversário passa a ter um comportamento ativo para explorar o modelo de ameaça tradicional do esquema de Shamir [Tompa and Woll 1989]. Esse ataque explora a ação de um adversário tomar a forma de um Participante que maliciosamente submete partes falsas no processo de reconstrução do segredo. Após a mudança no modelo de ameaça, algumas perguntas surgem e que precisam ser respondidas:

1. Quem reconstrói o segredo?
2. As partes são reveladas durante o processo de reconstrução?
3. Os adversários são estáticos ou dinâmicos?
4. Quais são os objetivos dos adversários?

Portanto, necessita-se descrever diferentes modelos de ameaça para se adequar aos novos cenários de uso e tentar responder essas perguntas. Descreve-se a seguir os modelos de ameaças alternativos encontrados na literatura.

5.1. Modelo de Ameaça Alternativo 1 - Distribuidor Malicioso

Este modelo considera que os Participantes são honestos (que vão seguir os protocolos de maneira correta) e que o Distribuidor é malicioso de forma limitada. Definimos aqui que existe algum tipo de acordo entre o Distribuidor malicioso e um adversário externo. Devemos ressaltar que o Distribuidor não é uma entidade ”toda poderosa”, onde poderia fazer qualquer coisa. Existem limites que precisamos definir. O Distribuidor malicioso gerará os parâmetros, sorteará um segredo e o enviará as partes para os Participantes. Entretanto, o Distribuidor malicioso terá como um dos principais objetivos prejudicar a reconstrução do segredo distribuindo de forma proposital partes que não pertencem ao esquema. Outra ameaça do Distribuidor é tentar de alguma forma alterar a geração do polinômio, não sendo completamente aleatório como os protocolos tradicionais recomendam. Esses detalhes podem comprometer o segredo no momento da reconstrução, inviabilizando a obtenção correta do parte ou fazendo com que determinados pontos possam ser favorecidos. Os Participantes receberão as partes, guardarão de forma segura e irão participar de forma honesta na reconstrução do segredo enviando de forma segura as partes para o Reconstrutor. Não fica claro quem deve assumir o papel do Reconstrutor, mas o mesmo irá reunir as partes para finalmente reconstruir o segredo. Caso o Reconstrutor seja uma terceira parte, as partes do segredo não são reveladas, caso seja um dos participantes, as partes são reveladas, o segredo é reconstruído, utilizado e então as partes devem ser descartadas. A tabela 2 ilustra a relação do modelo de ameaça alternativo 1 com as entidades.

© 2017 Sociedade Brasileira de Computação

Tabela 2. Modelo de Ameaça 1 - Distribuidor Malicioso

	Confiável	Malicioso	Passivo	Ativo
Distribuidor		✓		
Participante	✓			
Reconstrutor	✓			
Adversário				✓

5.1.1. Esquemas Seguros para este Modelo

Os esquemas de segredo compartilhado verificáveis (SCV) de Pedersen [Pedersen et al. 1991] e Feldman [Feldman 1987], que são baseados no esquema de Shamir e criptografia homomórfica, são protocolos que conseguem manter segurança nesse tipo de modelo de ameaça. Esses protocolos necessitam de um algoritmo chamado "Verificar"que permite aos Participantes verificar a validade de suas partes (antes de tentar qualquer reconstrução). Ao final do algoritmo, cada participante decide se aceita ou rejeita sua parte. O algoritmo precisa verificar: Consistência: um subgrupo autorizado de participantes que aceita suas partes poderá reconstruir o segredo k. Corretude: Se o Distribuidor for honesto, então o valor k de segredo é genuíno.

5.2. Modelo de Ameaça Alternativo 2 - Participante Malicioso

Este modelo de ameaça considera que um adversário pode tomar a forma de um participante e começar a atuar de forma ativa. Ou seja, o participante tentará de alguma forma comprometer o esquema submetendo partes incorretas no momento da reconstrução, fazendo com que os participantes honestos não consigam reconstruir o segredo e também falha em alertar os outros participantes que eles não reconstruíram o segredo correto. Considera-se aqui o Distribuidor uma entidade honesta que gera os parâmetros, o segredo e distribui as partes de forma segura para os Participantes. Não fica claro quem deve ser o Reconstrutor, entretanto, o modelo de ameaça deixa em aberto que um Participante malicioso poderia assumir esse papel. Ou seja, uma entidade não confiável pode prejudicar a reconstrução do segredo. A tabela 3 ilustra a relação do modelo de ameaça alternativo 2 com as entidades.

Tabela 3. Modelo de Ameaça 2 - Participante Malicioso

	Confiável	Malicioso	Passivo	Ativo
Distribuidor	✓			
Participante		✓		
Reconstrutor	✓			
Adversário				✓

5.2.1. Esquemas Seguros para este Modelo

Considerando o modelo de ameaça 2, onde um participante pode ser malicioso, o esquema de Bellare e Rogaway [Rogaway and Bellare 2007] propõe um framework para esquemas

 © 2017 Sociedade Brasileira de Computação

de segredo compartilhado que mantém um Distribuidor confiável e participantes polarizados, ou seja, existem participantes honestos e maliciosos. O framework consegue garantir a segurança nesse modelo de ameaça propondo um protocolo chamado HK2, redefinindo o esquema de Krawczyk's. O framework estende a ideia de adversários classificando como poderiam comprometer as partes. Esses adversários são classificados da seguinte forma:

- Erasure (0): Adversário não consegue corromper as partes (apenas visualizar e prevenir o uso no processo de reconstrução).
- Recoverability (1): Adversário pode corromper todas as partes, exceto uma (isso corresponde o caso onde o Reconstrutor é um participante honesto).
- Recoverability (2): Um adversário pode corromper todas as partes (isso corresponde ao caso onde o reconstrutor é uma terceira parte).

Outros esquemas que também tentam garantir segurança contra Participantes maliciosos são os trabalhos de Carpentieri [Carpentieri 1995] e Rabin e Ben-Or [Rabin and Ben-Or 1989], onde propõe um esquema que os Participantes possuem canais de comunicação segura entre eles para troca de mensagens. Esses participantes podem se comunicar secretamente que podem verificar as partes com uma maioria honesta.

Dado um conjunto total de participantes n, sendo necessário no mínimo t para reconstruir o segredo, o protocolo se baseia em um Distribuidor confiável que gerará um segredo k, partes $s_1, \ldots, s_n$ que serão distribuídos para os Participantes. Além desses valores, o Distribuidor enviará também um vetor para verificação v_i para cada Participante i. Ao final do processo, os Participantes devem se reunir para tentar recuperar o segredo com o Reconstrutor obtendo novamente um valor y. Os Participantes devem então verificar esse valor utilizando suas partes s_i juntamente com o vetor v. Cada um ao final da operação de verificação deverá obter um valor que seria o segredo k. Caso uma maioria honesta consiga obter o valor y, isso significa que $y = k$ e recuperaram de forma correta o segredo inicial.

6. Novo Modelo de Ameaça

O novo modelo de ameaça descrito neste trabalho consiste no uso de mais uma entidade definida na seção 3 chamada de Detentor do Segredo. Além disso, este trabalho definirá o Reconstrutor como uma terceira parte. Ou seja, será uma entidade que não exerce o papel de Participante, podendo ser tanto o Distribuidor quanto uma terceira parte exercendo exclusivamente este papel.

Neste modelo de ameaça, o Detentor do Segredo honesto cria um determinado segredo k de acordo com suas regras locais, cria os parâmetros de limiar (n, m) e envia de maneira segura para o Distribuidor. O Distribuidor malicioso então gerará as partes e distribuirá para os Participantes. Essa geração das partes pode estar comprometida, pois o Distribuidor pode gerar partes falsas com o objetivo de atrapalhar a reconstrução do segredo ou gerar as partes de tal forma que comprometa a segurança do sistema, possibilitando que menos partes poderiam ser utilizadas para recuperar o segredo. Além disso, a distribuição pode não ser feita de maneira segura para os Participantes, pois não se sabe como essa comunicação é feita. Os Participantes são considerados honestos, pois guardarão suas partes com segurança e participarão de maneira correta do momento de

© 2017 Sociedade Brasileira de Computação

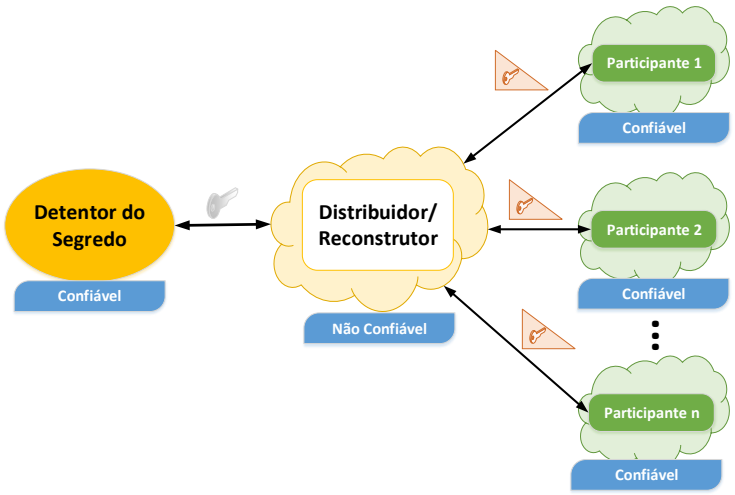


Figura 1. Ilustração de um exemplo do novo modelo de ameaça a esquemas de segredo compartilhado.

reconstrução. O Reconstrutor é uma terceira entidade ou o próprio Distribuidor e se comporta de maneira maliciosa. Ou seja, o mesmo pode não recuperar o segredo de maneira deliberada ou pode tentar divulgar o segredo e as partes sem a permissão do Detentor do Segredo.

Um exemplo seria uma aplicação que gera parâmetros secretos e deseja utilizar um dispositivo criptográfico disponibilizado em nuvem para replicar os dados de maneira segura e tentar garantir disponibilidade desses dados. O Detentor do segredo (que seria a aplicação local) utilizaria uma terceira parte não confiável para como Distribuidor e Reconstrutor para distribuir e reconstruir as partes antes de obter novamente o segredo. Essas partes seriam enviadas para outros provedores de nuvens separados para garantir segurança e disponibilidade. Dessa forma, seria interessante que apenas o Detentor do Segredo tivesse posse do Segredo em claro para garantir a segurança em todo o processo. A figura 1 ilustra o exemplo aqui comentado. A tabela 4 ilustra a relação do novo modelo de ameaça com as entidades.

Tabela 4. Novo Modelo de Ameaça - Distribuidor e Reconstrutor Maliciosos

	Confiável	Malicioso	Passivo	Ativo
Detentor do Segredo	✓			
Distribuidor		✓		
Participante	✓			
Reconstrutor		✓		
Adversário				✓

7. Avaliação Novo Modelo de Ameaça

Para realizar a avaliação dos esquemas no novo modelo de ameaça vamos seguir as etapas:

1. Mostrar um protocolo de segredo compartilhado com a entidade Detentor do Segredo.

 © 2017 Sociedade Brasileira de Computação

2. Analisar quais seriam os pontos de vulnerabilidade considerando o Novo Modelo de Ameaça.
3. Criar uma tabela comparativa mostrando a vulnerabilidade de diferentes esquemas de segredo compartilhado frente à algumas das principais ameaças.
4. Criar uma tabela mostrando quais esquemas de segredo compartilhado ainda estão vulneráveis para os diferentes modelos de ameaça.
5. Propor uma possível solução para contornar o problema de segurança encontrado.

O seguinte protocolo ilustra um exemplo de como fica um esquema de segredo compartilhado com uma nova entidade. A partir disso, pode-se avaliar o conhecimento de cada entidade e verificar a segurança conforme o modelo de ameaça adotado.

$$
\begin{array}{llcll}
M_1 & Dt & \xrightarrow[\text{interna}]{} & Dt & : \; k, t, n \\
M_2 & Dt & \xrightarrow[\text{externa}]{} & D & : \; k, t, n \\
M_3 & D & \xrightarrow[\text{externa}]{} & P_1 \dots P_n & : \; (s_1, \dots, s_n) \\
M_4 & P_1 \dots P_n & \xrightarrow[\text{externa}]{} & R & : \; (s_1, \dots, s_n) \\
M_5 & R & \xrightarrow[\text{interna}]{} & R & : \; k \\
M_6 & R & \xrightarrow[\text{externa}]{} & Dt & : \; k
\end{array}
$$

Considerando Dt como Detentor do segredo, D como o Distribuidor malicioso, P_i tal que $i \in \{1, \dots, n\}$ são Participantes, R é o Reconstrutor malicioso, a seta $\rightarrow$ representa a comunicação entre Entidades, podendo ser $interna$ ou $externa$ (ou seja, mensagens para criação própria de parâmetros ou mensagens para outras entidades com ações do esquema), s_i tal que $i \in \{1, \dots, n\}$ são partes do segredo, k é o segredo original gerado a partir de partes enviadas pelos participantes.

A análise do esquema é o seguinte: No protocolo anterior, a mensagem M_1 que o Detentor do segredo gera os parâmetros k, t, n, sendo esses os necessários para fazer o esquema funcionar. Os mesmos são então repassados de maneira segura para o Distribuidor na mensagem M_2. A partir deste momento, o Distribuidor tem conhecimento do segredo e pode tomar atitudes suspeitas como a divulgação do segredo ou alteração dos parâmetros, não gerando de forma correta e honesta. Considerando que o mesmo seja malicioso, o esquema estará parcialmente comprometido a partir dessa mensagem M_2. A mensagem M_3 é o envio seguro das partes para todos os Participantes do esquema. Esses Participantes são honestos e não subverterão o protocolo. Participarão de forma honesta na reconstrução do segredo. A mensagem M_4 é o processo de recuperação do segredo, onde os Participantes enviam suas partes para o Reconstrutor de forma segura. A mensagem M_5 é o processo onde o Reconstrutor recupera o segredo e pode tomar atitudes suspeitas, como a divulgação do mesmo. A mensagem M_6 é onde o Reconstrutor envia de volta o segredo para o Detentor do Segredo.

Ao avaliar o protocolo anterior, existem diversas partes onde entidades maliciosas podem tomar atitudes suspeitas como a divulgação do segredo, mudança nas partes e mudança no segredo. A tabela 5 ilustra a quais ameaças estão expostas cada um dos principais esquemas de segredo compartilhado.

Na tabela 5, as ameaças são descritas a seguir:

© 2017 Sociedade Brasileira de Computação

Tabela 5. Ameaças aos quais cada esquema de segredo compartilhado está exposta.

	A_1	A_2	A_3	A_4	A_5
[Shamir 1979]	✓	✓	✓	✓	✓
[Feldman 1987]	✓	✓			
[Pedersen et al. 1991]	✓	✓			
[Rabin and Ben-Or 1989]	✓	✓			
[Rogaway and Bellare 2007]	✓	✓			

- A_1- **Divulgar Segredo:** Quando o Distribuidor não é totalmente confiável, o segredo que é repassado para ele não cifrado pode ser divulgado sem a autorização do Detentor do Segredo para um adversário. Esse adversário pode ser uma terceira parte fora do esquema ou para algum Participante malicioso.
- A_2- **Alteração dos Parâmetros:** Ao utilizar um Distribuidor malicioso como gerador de parâmetros do esquema de segredo compartilhado, pode-se ter alterações nos parâmetros originais que foram repassados por um Detentor de Segredo ou uma terceira parte confiável. Essa alteração pode consistir em:
  1. Não utilizar um gerador de números aleatórios confiável
  2. Gerar propositadamente parâmetros fracos
  3. Não utilizar os parâmetros corretos e possibilitar a recuperação com menos partes do que o necessário
- A_3- **Alterar Partes:** Participantes podem receber corretamente suas partes, entretanto, no momento da reconstrução, Participantes maliciosos podem enviar partes incorretas para prevenir a geração correta do segredo.
- A_4- **Subverter Protocolo:** Podem existir Participantes maliciosos que tentarão atrapalhar o protocolo enviando partes erradas ou simplesmente não participando das reconstruções. Essas atitudes podem comprometer a reconstrução do segredo, evitando que saibam quem realizou os ataques.
- A_5- **Partes Reveladas na Reconstrução:** Esse tipo de ameaça depende do tipo do modelo de ameaça e da entidade Reconstrutor. Se essa entidade é uma terceira parte confiável, as partes normalmente não são reveladas, entretanto, ao colocar um participante, deve-se revelar as partes e descartar após o procedimento de reconstrução. Caso o reconstrutor seja uma entidade maliciosa, o mesmo pode querer divulgar as partes para adversários.

A tabela 6 faz um comparativo dos diferentes esquemas de segredo compartilhado e para quais modelos de ameaça são vulneráveis. Nota-se que para o novo modelo todos estariam vulneráveis.

Tabela 6. Quais esquemas de segredo compartilhado estão vulneráveis para diferentes modelos de ameaça.

	Tradicional	Modelo 1	Modelo 2	Novo Modelo
[Shamir 1979]	✓			✓
[Feldman 1987]				✓
[Pedersen et al. 1991]				✓
[Rabin and Ben-Or 1989]				✓
[Rogaway and Bellare 2007]				✓

 © 2017 Sociedade Brasileira de Computação

7.1. Possível Solução

Ao analisar o novo modelo de ameaça aqui proposto, nota-se que existem diversos pontos de falha, como a divulgação do segredo e alteração dos parâmetros. Para que essas propriedades sejam protegidas, o Detentor do Segredo pode tomar algumas providências iniciais para garantir uma maior segurança.

Uma das atitudes iniciais é considerar que existe essa entidade a mais, fazendo com que o Distribuidor não seja uma entidade totalmente confiável. Partindo dessa premissa, pode-se então cifrar o segredo antes de enviar para o Distribuidor com uma chave simétrica Ks. O seguinte protocolo ilustra essa possível solução.

$$
\begin{array}{llcll}
M_1 & Dt & \xrightarrow[\text{interna}]{} & Dt & : \; k, t, n, Ks \\
M_2 & Dt & \xrightarrow[\text{interna}]{} & Dt & : \; E(k)_{Ks} \\
M_3 & Dt & \xrightarrow[\text{externa}]{} & D & : \; (k)_{Ks}, t, n \\
M_4 & D & \xrightarrow[\text{externa}]{} & P_1 \dots P_n & : \; (s_1, \dots, s_n) \\
M_5 & P_1 \dots P_n & \xrightarrow[\text{externa}]{} & R & : \; (s_1, \dots, s_n) \\
M_6 & R & \xrightarrow[\text{interna}]{} & R & : \; (k)_{Ks} \\
M_7 & R & \xrightarrow[\text{externa}]{} & Dt & : \; (k)_{Ks} \\
M_8 & Dt & \xrightarrow[\text{interna}]{} & Dt & : \; D(k)_{Ks}
\end{array}
$$

Considerando Dt como Detentor do segredo, D como o Distribuidor malicioso, P_i tal que $i \in \{1, \dots, n\}$ são Participantes, R é o Reconstrutor malicioso, a seta $\rightarrow$ representa a comunicação entre Entidades, podendo ser $interna$ ou $externa$ (ou seja, mensagens para criação própria de parâmetros ou mensagens para outras entidades com ações do esquema), s_i tal que $i \in \{1, \dots, n\}$ são partes do segredo, k é o segredo original gerado a partir de partes enviadas pelos participantes.

Neste protocolo pode-se ver que na mensagem M_1 além de criar os parâmetros, o Detentor do Segredo agora também cria uma chave simétrica Ks para cifrar o segredo. A mensagem M_2 ilustra o procedimento de cifragem do segredo k com a chave Ks. A partir da mensagem M_3 até a mensagem M_7 é feita da mesma forma como o protocolo anterior, com a diferença de que as outras entidades tem conhecimento apenas do segredo cifrado. Na mensagem M_8 o segredo é então decifrado pelo Detentor do Segredo.

8. Considerações Finais

Com a análise feita nos modelos de ameaças tradicionais e alternativos, pode-se detectar que esses modelos não refletem a realidade de muitas aplicações utilizadas. Grande parte das aplicações utilizam uma entidade separada para gerar o segredo e os parâmetros utilizando o Distribuidor somente para quebrar o segredo e compartilhar as partes. Dessa forma, este trabalho propôs a criação de uma nova entidade chamada Detentor do Segredo. Após a criação dessa nova entidade, os modelos de ameaça mudam e precisam ser novamente analisados. Por isso, um novo modelo de ameaça também foi proposto para analisar os principais esquemas de segredo compartilhado.

© 2017 Sociedade Brasileira de Computação

Após o levantamento dos principais esquemas de segredo compartilhado na literatura para os diferentes modelos de ameaças, atingiu-se o objetivo de detectar vulnerabilidades para esses esquemas utilizando o novo modelo de ameaça aqui proposto. A análise foi baseada na avaliação do protocolo tradicional de segredo compartilhado, utilizando as entidades normalmente encontradas na literatura. Com isso, observou-se alguns pontos de falhas para algumas ameaças aqui citadas. Com essas novas ameaças, este trabalho também propôs uma possível solução para contornar as vulnerabilidades encontradas.

Como trabalho futuro, pretende-se aplicar o novo modelo de ameaça descrito neste trabalho para introduzir um novo assunto chamado de estruturas de acesso escondidas. Essas estruturas de acesso escondidas tem como uma das premissas o uso de Distribuidores maliciosos e diferentes modelos de ameaça de tal forma que seja possível recuperar o segredo com menos partes que o liminar definido no protocolo de Shamir.

Referências

Beimel, A. (2011). Secret-sharing schemes: a survey. In *Coding and cryptology*, pages 11–46. Springer.

Blakley, G. R. (1979). Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317.

Carpentieri, M. (1995). A perfect threshold secret sharing scheme to identify cheaters. *Designs, Codes and Cryptography*, 5(3):183–187.

Feldman, P. (1987). A practical scheme for non-interactive verifiable secret sharing. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 427–438. IEEE.

Martin, K. M. (2008). Challenging the adversary model in secret sharing schemes. *Coding and Cryptography II, Proceedings of the Royal Flemish Academy of Belgium for Science and the Arts*, pages 45–63.

Pedersen, T. P. et al. (1991). Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto*, volume 91, pages 129–140. Springer.

Rabin, T. and Ben-Or, M. (1989). Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM.

Rogaway, P. and Bellare, M. (2007). Robust computational secret sharing and a unified account of classical secret-sharing goals. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 172–184. ACM.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Tompa, M. and Woll, H. (1989). How to share a secret with cheaters. *journal of Cryptology*, 1(3):133–138.

 © 2017 Sociedade Brasileira de Computação

Authentication of Identity Documents Using DNSSEC, Digital Signatures and QR Codes

Luiz Fernando Ribeiro Amaral[1],
Jorge Guilherme Silva dos Santos[1], **Mateus Almeida Rocha**[1],
Joseph Gersch[2], **Georges Daniel Amvame-Nze**[1],
Robson de Oliveira Albuquerque[1], **Rafael Timóteo de Sousa Júnior**[1]

[1]Decision Technologies Laboratory - LATITUDE
Electrical Engineering Department
University of Brasília (UnB)
Brasília, DF, Brazil

[2]Department of Computer Science
Colorado State University
Fort Collins, CO, USA

{luiz,jorge,mateus,robson}@redes.unb.br,

jgersch@colostate.edu,

{georges,desousa}@unb.br

Abstract. *This paper describes a system for authenticating identity documents by digitally signing the data and embedding it in a 2D code to be printed with the document. In the proposed scheme, a message is digitally signed using a digital signature block which is stored in a QR Code. The code is later scanned by the user and, after validation, the corresponding digital information can be compared with the printed information. The authenticity of the message is guaranteed using RSA digital signatures and a secure DNS implementation based on DANE and DNSSEC for the certificate distribution. A proof of concept was implemented using the ISC BIND DNS Server and OpenSSL to create and distribute the certificates and a Telegram Bot for signature verification.*

1. Introduction

Document counterfeiting is on the rise as sophisticated printing and scanning technologies become cheap, enabling criminals to take advantage of digital technology to produce high quality fraudulent documents. Society relies on experts for verifying the authenticity of these documents using special tools and document properties, an almost impossible job for entities dealing with thousands of documents daily [Garain and Halder 2008].

The system proposed in this paper addresses the verification of an identity document by enabling any person to verify the information contained in this document using a smartphone application. There is no need for specialized people or physical security features in the document itself.

Compared to previous work [Warasart and Kuacharoen 2012], this system does not rely on a certificate authority to provide valid certificates, nor does it need to have the certificates preloaded by the application used for signature validation. Using DNSSEC

 © 2017 Sociedade Brasileira de Computação

(Domain Name System Security Extensions) enabled servers, the certificates can be published in the signing entity zone and the authenticity and integrity of the data can be guaranteed by the chain of trust provided by DNSSEC signatures [Arends et al. 2005].

Digital signatures are a standard for guaranteeing authenticity and integrity nowadays. *A digital signature algorithm allows an entity to authenticate the integrity of signed data and the identity of the signatory. The recipient of a signed message can use a digital signature as evidence in demonstrating to a third party that the signature was, in fact, generated by the claimed signatory* [FIPS 2013]. In our proposed scheme, a message is digitally signed using a digital signature block which is stored in a QR Code which is printed on the identity document, such as for instance a driver's license containing a QR Code with the message and its signature as shown in Figure 1.

Figure 1. Driver's license example. Adapted from [DMV 2013]

This paper is organized so the complete system architecture is first described and then a proof of concept implementation of the system is presented.

2. The System Architecture

The architecture is composed of signing entities, which can be governmental institutions or companies who issue identification documents, a phone application used for the signature validation and the DNS infrastructure. Figure 2 shows a complete diagram of the system. Each entity can have a signing server, for storing the keys and generating the QR Codes with the signatures and the DNS servers to provide the certificates for signature validation. A user, using his smartphone scans the QR Code using the camera. The phone decodes the data and queries the DNS server to retrieve the certificate and validate the DNSSEC chain of trust. It then validates the signature using the retrieved certificate and if valid, presents the data to the user.

A key feature is that, while other implementations propose the use of optical character recognition (OCR) for obtaining the message for validation [Warasart and Kuacharoen 2012], we rely on a person to validate the document by comparing digitally signed data contained in the QR Code message with the actual data printed on the document.

2.1. DNS Servers

Although the CERT Resource Record specification allows the storage of certificates in insecure DNS zones, it specifies that in these cases the whole certificate chain needs to

 © 2017 Sociedade Brasileira de Computação

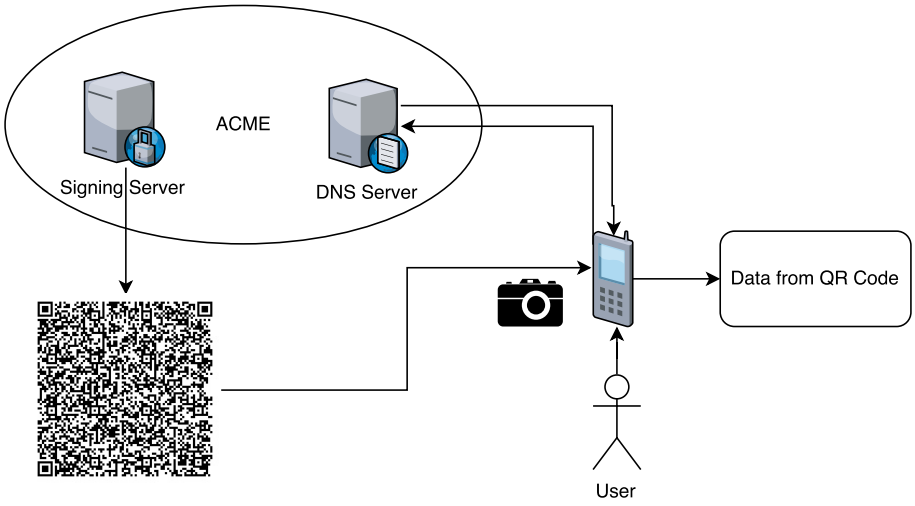


Figure 2. System Architecture Overview[Source: Authors]

be validated, exacerbating the principle of relying on a certificate authority. To be able to trust the certificate without the need of verifying the whole certificate chain, we propose the use of CERT or DANE self-signed certificates in the DNS. In this case, the zone must be secure, i.e, DNSSEC-enabled, and the record authenticity verified by using DNSSEC [Josefsson 2006b].

The DNSSEC specification defines the required behavior for DNSSEC-enabled zones, including serving keys, signatures, secure denial of existence and serving the zone securely in all of the authoritative name servers [Osterweil et al. 2009]. NIST allows the use of 1024-bit RSA keys for DNSSEC, as long as other key management practices are put in place, to avoid key compromise [Chandramouli and Rose 2013].

2.2. Message Format

The messages used by the system consist of JSON objects such as shown in Figure 3. JSON is a lightweight, text-based, language-independent data interchange format [Bray 2014]. This choice brings flexibility to the data format, making it easy to add or remove fields, without the need to completely rewrite data templates or the application itself.

The message object needs two mandatory values: *_root* and *_key*, representing the DNS root and the key identifier, respectively. These fields indicate where to find the certificate needed for signature validation. The validating application still needs to know the DNS zone corresponding to the indicated *_root* to fetch the certificate, but the key field brings more flexibility, as the signing entity can easily add more certificates to the DNS infrastructure without the need of updating the signature validation application.

2.3. Information Signing

The message is signed using the RSA digital signature scheme, with 2048-bit RSA private keys, the SHA-512 hash algorithm and the PSS padding scheme, meeting the recommendations established by NIST in the SP800-57 Part 3 Rev. 1 [NIST 2015].

Figure 4 illustrates the message signature process. The issuer of a document calculates the message SHA-512 hash, encrypts it using the RSA private key and finally

© 2017 Sociedade Brasileira de Computação

```
{
    "_id": "1",
    "_root": "acme",
    "Name": "Motorist M. Michael",
    "DOB": "08/31/1978",
    "ID": "123 456 789",
    "Class": "D",
    "Issued": "08/31/2013",
    "Expires": "08/31/2021"
}
```


Figure 3. Example Message [Source: Authors]

appends it to the end of the message. Message and signature are then published in the form of a QR Code on the printed document. It is valid mentioning that the padding and mask generation operations were removed from the figure for simplicity.

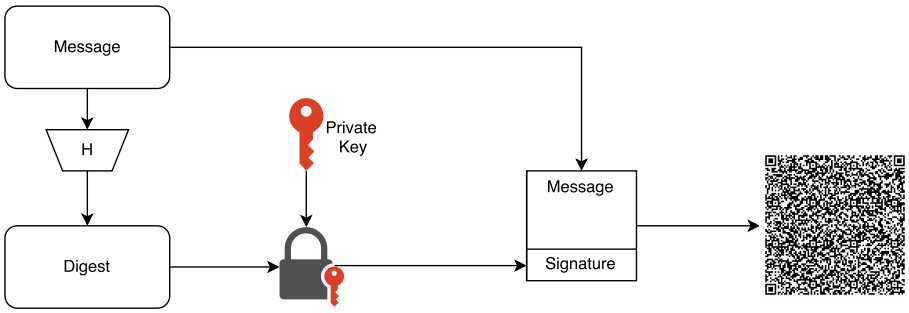


Figure 4. Message signature process. [Source: Authors]

2.4. QR Code

A QR Code is one of the many kinds of 2D codes. It was invented in 1994 to be used in automotive industry and became an ISO standard in 2000. Nowadays it has become widespread, being used in electronic components labeling, aircraft and space industrial data-product identification, bus tickets, betting tickets, patient identification in hospitals and many other applications in all kinds of fields [Soon 2008].

The QR Code specification states that it can store 2,953 characters at maximum when using 8-bit binary mode. Other modes are available and offer higher capacity, but are limited and don't cover our use case. Table 1 shows the capacity of each of the available modes.

The QR Code standard states that the alphanumeric mode, for example, covers only part of the ASCII character table and doesn't include the brackets and colon ({,},:) used in the JSON message object. One way to make our data comply with the requirements of the alphanumeric mode would be encoding it using Base32, but the gain in data capacity between the alphanumeric mode and the binary mode is approximately 45%, while the Base32 encoding adds an overhead of 60%, making it inefficient for use with

 © 2017 Sociedade Brasileira de Computação

Table 1. QR Code capacity for each mode. Adapted from [Soon 2008]

Mode	Capacity
Numerical characters	7,089 characters at maximum
Alphanumerical	4,296 characters at maximum
Binary (8 bit)	2,953 characters at maximum
Kanji characters	1,817 characters at maximum

the alphanumeric mode [Josefsson 2006a]. Our proposed system is designed for the authentication of identity documents, which usually don't carry a large amount of data. For example, a drivers license contains an ID number, name, address, sex, height, DOB, expiration date and issuance date. In the case of an application that needs to carry a big amount of data, compression may be applied to the data and signature which will help to reduce the size of the QR Code printout.

2.5. Storing Certificates on the DNS

The certificates used for signature validation are stored in the signing entity DNS zone under CERT Resource Records. According to [Josefsson 2006b], *it is recommended that certificate CERT Resource Records be stored under a domain name related to their subject, i.e., the name of the entity intended to control the private key corresponding to the public key being certified*. Hence, each document issuer who uses the system will store its certificates under its own DNS domain name and zone.

The structure of a CERT Resource Record is shown in Figure 5. in this structure, the certificate type field defines the type of data contained in the record. In our case, this field is set with type 1 - PKIX. The key tag field is a 16-bit value computed from the certificate public key used for efficient public key selection. The algorithm used for computation is specified in [Arends et al. 2008], Appendix B. The algorithm field identifies the digest algorithm used. This value changes based on the certificate being published in the record and a full list of algorithms and their IDs can be found in [IANA 2017]. Finally, the Base64 encoded certificate goes into the certificate field.

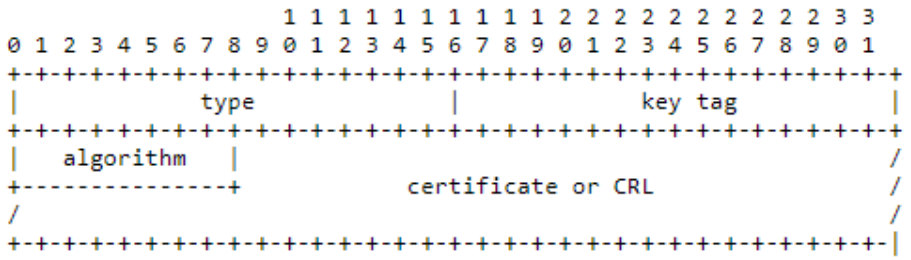


Figure 5. CERT Resource Record Structure (Numbers on top indicate each bit). Source: [Josefsson 2006b].

2.6. Signature Validation

This process, as shown in Figure 6, is performed in a smartphone, beginning with the QR Code capture by the device camera. After scanning the QR Code, the signature validation

© 2017 Sociedade Brasileira de Computação

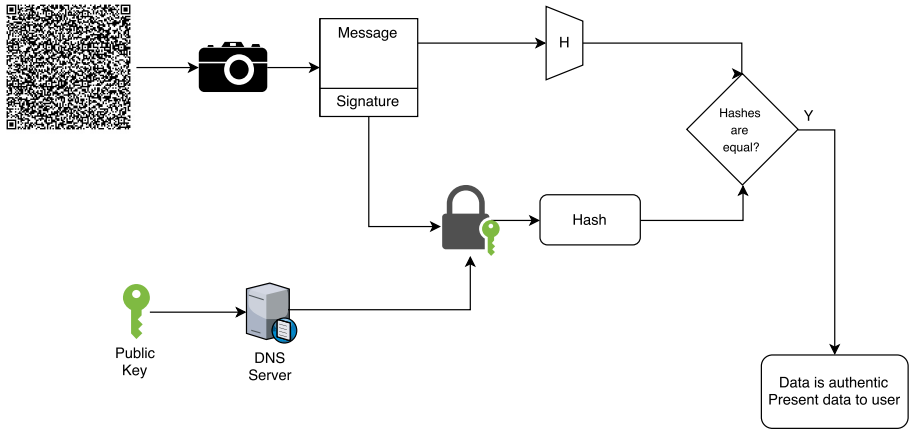


Figure 6. Signature validation process [Source: Authors]

is quickly done, as the process of decryption only takes milliseconds in most contemporary smartphones [Thiranant et al. 2015].

Thus, to validate the signature, the user takes a picture of the identification document QR Code using a smartphone. The application then splits the message and signature and starts two different processes. The first one calculates the SHA-512 hash of the message and keep it for later usage. The second process uses the *_root* and *_key* contained in the message to fetch the certificate from the DNS server and then decrypts the signature data and obtains the hash of the signed message. By comparing the two hashes, the authenticity of the message contained in the QR Code can be guaranteed.

The validation process in Figure 6 authenticates the message contained in the QR Code, but does not guarantee that the data in the printed document is authentic. This second validation needs to be done by the user, by visual comparison between the data presented by the application and the data in the document.

3. Proof of Concept

To evaluate the proposed architecture, a proof of concept was implemented using open-source software tools. A certificate authority was created using OpenSSL, two ISC BIND DNS servers were setup in master-slave operation, a web API was created to execute the signature operation and a Telegram Bot was used to provide an user interface for signature verification.

3.1. Certificate Authority

Using the default OpenSSL settings, an example certificate authority was created and two certificates, for two example companies were issued. The private keys related to the certificates were 2048-bit long, as recommended by NIST.

3.2. DNS Zone

To achieve the stated requirements, the system was implemented using two DNS servers, one master and one slave. Two example zones, representing pseudo-organizations Acme and Globex, respectively acme.luiz.eng.br and globex.luiz.eng.br, were created to hold

 © 2017 Sociedade Brasileira de Computação

the certificates used for the signatures. The zones were implemented with DNSSEC, in a pre-existing DNSSEC-enabled infrastructure.

While the acme.luiz.eng.br zone has a valid DNSSEC signature and valid DS entries in the parent zone, the zone globex.luiz.eng.br only has the DS entries in the parent zone, resulting in a signature validation failure as the zone is in a bogus state, simulating for example, a cache poisoning situation.

To verify the secure status of the ACME zone and bogus status of the Globex zone, we used the Verisign DNSSEC Analyzer [Verisign Inc. 2011]. Figure 7 shows the result of the DNSSEC validation for both zones, where the Globex zone does not contain a DNSKEY record, neither RRSIGs.

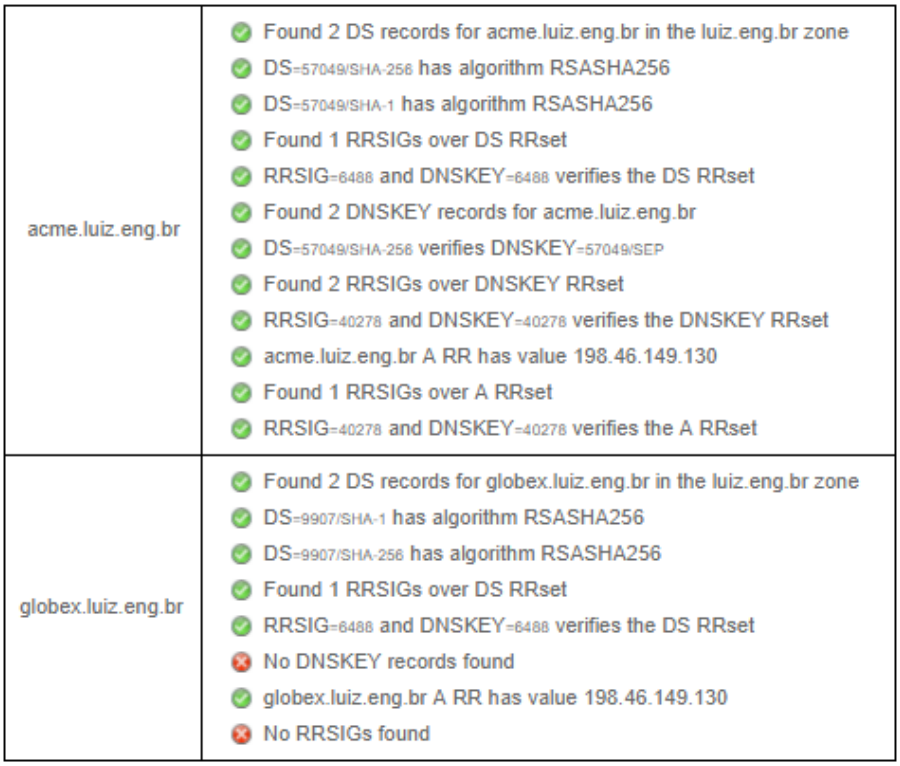


Figure 7. DNSSEC validation of the zones. Adapted from [Verisign Inc. 2011]

After setting up the zone, the certificates were published following the RFC4398 [Josefsson 2006b] specification, except for the naming scheme, as it is just a recommendation. The first certificate, the one for the fictitious Acme organization, was published under 1.acme.luiz.eng.br, and the second, for the fictitious Globex Corporation, was published under 1.globex.luiz.eng.br.

3.3. Message Signing

In our implementation, we used the Tornado Web Framework [The Tornado Authors 2017] to implement a Web API for message signing. With this API, two messages were signed to be used later in the validation process. The messages are JSON objects containing sample data of a driver's license, as shown in Figure 3. A sample output from the API can be seen in Figure 8.

© 2017 Sociedade Brasileira de Computação

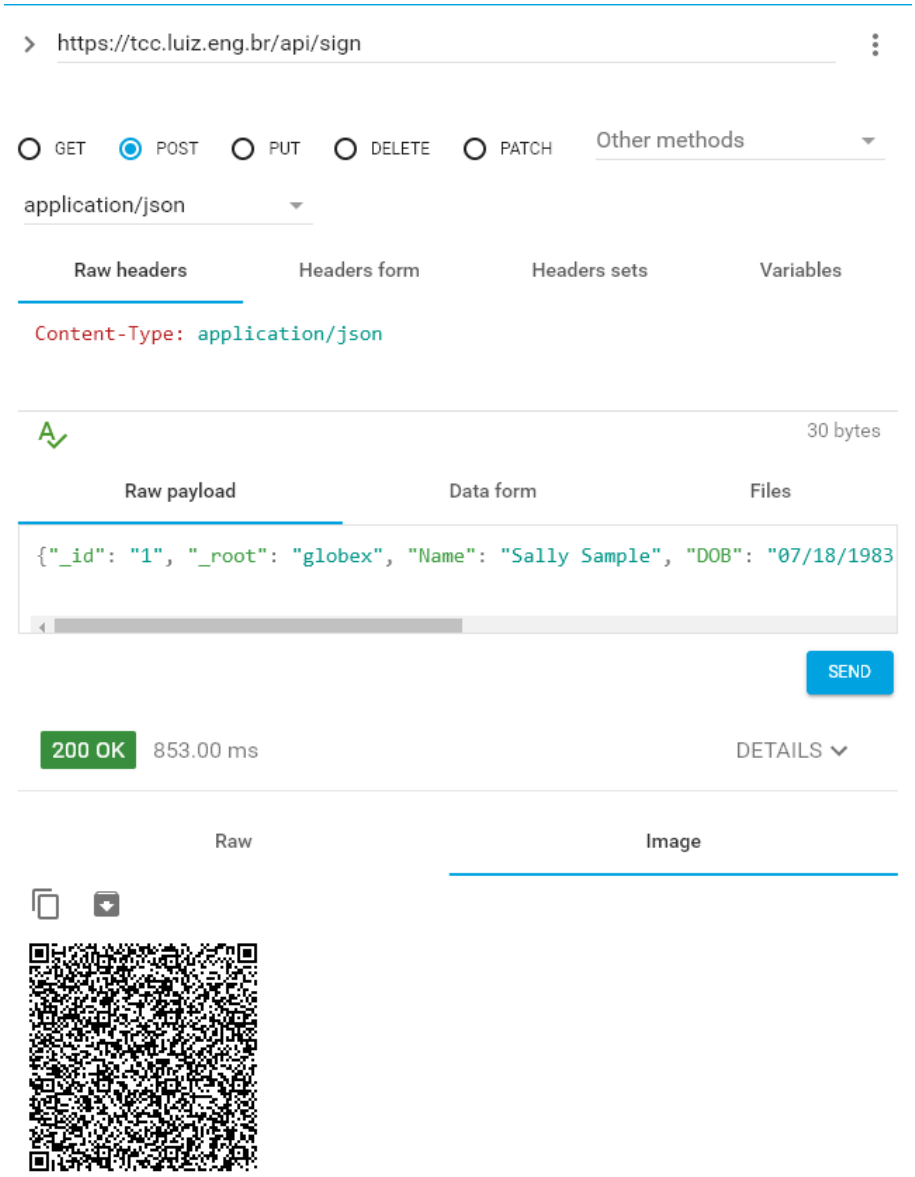


Figure 8. API Signature Request [Source: Authors]

The message signing itself is implemented through the hazardous materials layer of the Python Cryptography [Python Cryptographic Authority 2013] package. The hazardous materials layer contains low-level cryptography primitives, meaning that the functions included with it should be used carefully, as they can result in an insecure implementation.

The binary output of the signature function is encoded in Base64, so we can later separate the message from its signature using the ASCII character 0x1F, without the risk of having this same byte in the signature data.

When using a 2048-bit key, the produced signature is 256 bytes long and after the Base64 encoding, which has a 4:3 ratio [Josefsson 2006a], will result in a 344 bytes long string, leaving room for up to 2609 bytes for the message, although the utilization of this data depends on the QR Code capacity.

 © 2017 Sociedade Brasileira de Computação

Despite all the capacity left for the message, a QR Code to store all this data, would be too big to fit on an identity document. A solution is to employ the feature of QR Code linking, allowing the division into symbols enabling the printing even if there is not enough space available for the complete QR Code. Figure 9 shows an example of the linking technique.

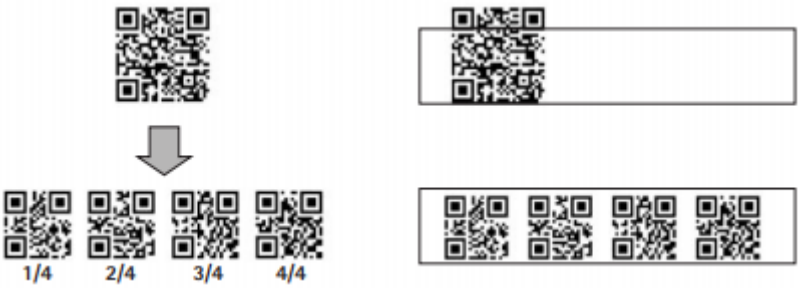


Figure 9. An example of QR Code linking [Soon 2008]

Regarding the example message in Figure 3, which has only 154 bytes and is small enough for its resulting QR Code to fit in an identity document, as the example shown in Figure 1, the QR Code linking feature was not employed in our proof of concept.

To prove that the system works, two different messages were signed and a QR Code was created for each of them. The two created QR Codes can be seen in Figures 10 and 11.

Figure 10. QR Code containing the message with a valid signature [Source: Authors]

© 2017 Sociedade Brasileira de Computação

Figure 11. QR Code containing the message with an invalid signature [Source: Authors]

3.4. Signature Validation

The signature validation is done through a Telegram Bot [Telegram 2017], as we considered it a simple way to implement a nice mobile user interface using Python. This decision makes our solutions multi-platform, so that the users can interact with the Bot using the Telegram app on different devices, including iPhones and Android phones.

To validate the signature, the Telegram Bot decodes the QR Code and splits the resulting data into message and encoded signature, considering the 0x1F character as their separator. The Bot then maps the *_root* field of the message to a list of zones already known to it and uses the *_key* field to build the complete DNS name to fetch the certificate.

When fetching the certificate, the validity of the DNSSEC signatures is checked, to establish if the returned data can be trusted or not. If the returned response has an invalid DNSSEC signature or no DNSSEC at all, the certificate cannot be trusted and, by consequence, the signature will be considered invalid.

If the certificate fetching is successful, the message is validated against the signature and, if the data matches, the information contained in the message is presented to the user. Figure 12 shows the resulting output for the two example QR Codes mentioned before.

4. Conclusion and Future Work

The proposed architecture can provide a distributed, flexible yet safe way of guaranteeing the authenticity of identity documents without the need of relying completely on a trusted third-party. By using the DNS infrastructure for certificate retrieval, identity document issuers can create and manage their own keys, making it easier to execute key rollovers for example, as the keys don't need to be preloaded by the verifying application or party.

The decentralization of the keys makes it harder for an attacker to compromise the whole system. As the keys are distributed, an attacker would have to breach into each of the subsystems to acquire all the keys and the DNSSEC guarantees that the keys stored on the DNS are served in a secure manner, preventing attacks like cache poisoning and alteration of the keys stored in the zone. By means of DANE, we are able to use and trust self-signed certificates, removing the need of a Public Key Infrastructure and reducing the costs related to a third-party Certificate Authority.

 © 2017 Sociedade Brasileira de Computação

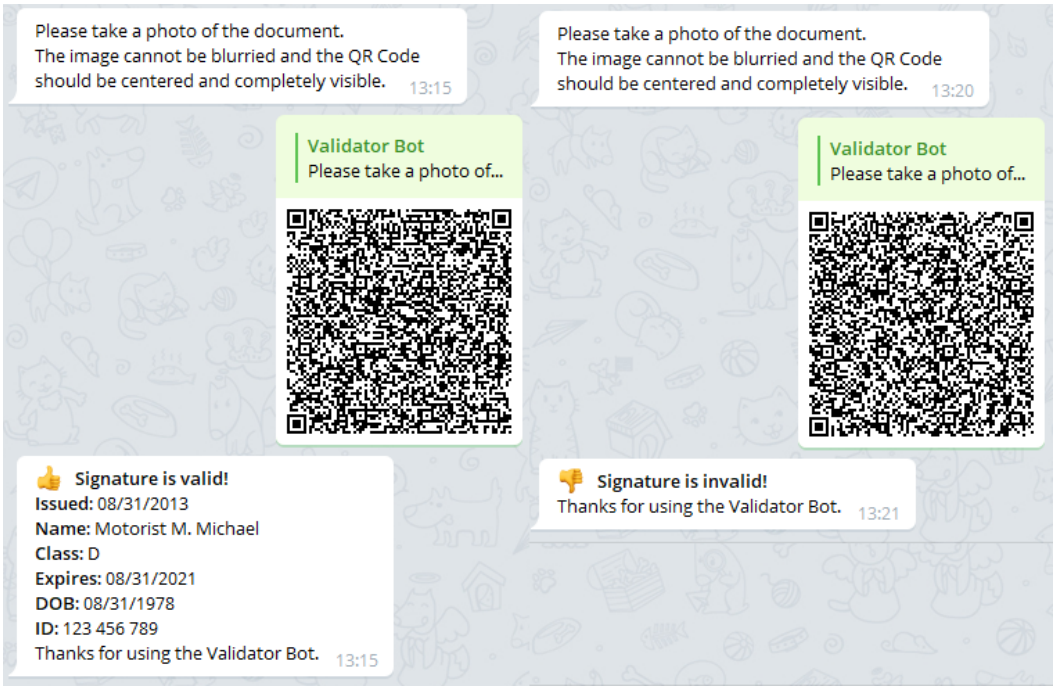


Figure 12. Bot messages after validating the two signatures [Source: Authors]

In the future, optical character recognition techniques can be used to automate the validation of identity documents, removing the need of a visual comparison done by the user. The security can be increased by including a timestamp to prevent replay attacks and including a serial number to provide an identifier to enable traceability by the issuer.

Acknowledgments

This research work has the support of the Brazilian research and innovation Agencies CAPES – Coordination for the Improvement of Higher Education Personnel (Grant 23038.007604/2014-69 FORTE – Tempestive Forensics Project) and CNPq – National Council for Scientific and Technological Development (Grant 465741/2014-2 Science and Technology National Institute – INCT on Cyber Security), as well as the Brazilian Ministry of Planning, Development and Management (Grants 005/2016 DIPLA – Planning and Management Directorate, and 11/2016 SEST – State-owned Federal Companies Secretariat), as well as Secure64 Software Corporation, for technical information on DNSSEC deployment and behavior.

References

Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S. (2008). Rfc 4034-resource records for the dns security extensions (2005).

Arends, R., Austein, R., Larson, M., Massey, D., and Rose, S. W. (2005). Dns security introduction and requirements, rfc 4033. *Internet Engineering Task Force (IETF)*.

Bray, T. (2014). The javascript object notation (json) data interchange format.

Chandramouli, R. and Rose, S. (2013). Secure domain name system (dns) deployment guide. *NIST Special Publication*, 800:81–2.

© 2017 Sociedade Brasileira de Computação

DMV, N. Y. S. (2013). Sample New York State DMV Photo Documents. https://dmv.ny.gov/id-card/sample-photo-documents. (Accessed on 2017-07-10).

FIPS, P. (2013). 186-4. *Digital Signature Standard (DSS)*.

Garain, U. and Halder, B. (2008). On automatic authenticity verification of printed security documents. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pages 706–713. IEEE.

IANA (2017). Domain Name System Security (DNSSEC) Algorithm Numbers. https://www.iana.org/assignments/dns-sec-alg-numbers/dns-sec-alg-numbers.xhtml. (Accessed on 2017-07-10).

Josefsson, S. (2006a). The base16, base32, and base64 data encodings.

Josefsson, S. (2006b). Storing certificates in the domain name system (dns).

NIST, S. (2015). 800-57 part 3 rev 1, recommendation for key management: Part 3: Application-specific key management guidance.

Osterweil, E., Massey, D., and Zhang, L. (2009). Deploying and monitoring dns security (dnssec). In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 429–438. IEEE.

Python Cryptographic Authority (2013). Welcome to pyca/cryptography — Cryptography 2.0.dev1 documentation. https://cryptography.io/en/latest/. (Accessed on 2017-07-10).

Soon, T. J. (2008). Qr code. *Synthesis Journal*, 2008:59–78.

Telegram (2017). Bots: An introduction for developers. https://core.telegram.org/bots. (Accessed on 2017-07-10).

The Tornado Authors (2017). Tornado web server - tornado 4.5.2 documentation. http://www.tornadoweb.org/en/stable/. (Accessed on 2017-07-10).

Thiranant, N., Lee, Y. S., and Lee, H. (2015). Performance comparison between rsa and elliptic curve cryptography-based qr code authentication. In *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*, pages 278–282. IEEE.

Verisign Inc. (2011). DNSSEC Analyzer. https://dnssec-debugger.verisignlabs.com/. (Accessed on 2017-07-10).

Warasart, M. and Kuacharoen, P. (2012). Paper-based document authentication using digital signature and qr code.

 © 2017 Sociedade Brasileira de Computação

Uma Avaliação das Prevenções de Phishing em Navegadores Web

Carlo M. R. da Silva[1], **Eduardo L. Feitosa**[2], **Vinícius C. Garcia**[1]

[1]CIn – Universidade Federal de Pernambuco (UFPE)

[2]IComp – Universidade Federal do Amazonas (UFAM)

{cmrs,vcg}@cin.ufpe.br, efeitosa@icomp.ufam.edu.br

Abstract. *Web browsers are tools of utmost importance when it comes to data consumption on the Internet, since they enable the interaction and consumption of information provided by various services available over the Web. Several daily companies adapted their services to be used on the Web in order to obtain advantages over competitors. However, it is clear the difficulty of these tools in preventing their users from being victims of phishing attacks. Such attacks if put into effect result in devastating and often irreversible consequences for business. Through this article, we present a study that evidences the lack of effective solutions in analyzing and minimizing such problematic submit to the protection mechanisms designed for the most popular Web browsers available.*

Resumo. *Navegadores Web são ferramentas de extrema importância no que diz respeito ao consumo de dados na internet, pois possibilitam a interação e consumo de informações providas por diversos serviços disponíveis na Web. Diversas empresas cotidianas adequaram seus serviços para serem utilizados na Web no intuito de obter vantagens competitivas entre seus concorrentes. Em contrapartida, é nítida a dificuldade destas ferramentas em evitar que seus usuários sejam vítimas de ataques de* phishing*. Tais ataques quando concretizados resultam em consequências devastadoras, e muitas vezes irreversíveis aos negócios. Através deste artigo, apresentamos um estudo que evidencia a carência de soluções eficazes em analisar e minimizar tal problemática sobre os populares navegadores Web disponíveis para os usuários.*

1. Introdução

A *Web* se consolidou como a plataforma mais eficiente para a distribuição de serviços. Desde seu advento, diversas organizações e segmentos, a exemplo de bancos e lojas virtuais, motivaram-se a migrar suas atividades e funcionalidades para ela, e com o passar dos anos essa tendência ganhou ainda mais força. Contudo, para que essas operações sejam realizadas pela *Web* é necessário, muitas vezes, que sejam fornecidas informações sensíveis de pessoas físicas ou jurídicas, tais como senhas de contas e cartões de crédito, dados cadastrais, entre outros. Com isso, cresce também o número de crimes cibernéticos relacionados à fraudes eletrônicas, em especial *phishing* ou *Phishing Scam*[1], onde atacantes visam interceptar essas informações.

[1]O termo SCAM é uma sigla para *Scheming Crafty Aggressive Malicious*

 © 2017 Sociedade Brasileira de Computação

De acordo com o Cert.BR, *phishing* é o tipo de fraude por meio da qual um golpista tenta obter dados pessoais e financeiros de um usuário, pela utilização combinada de meios técnicos e engenharia social [Cert.BR 2012]. Um ponto importante sobre *phishing* é que sua investida emerge de forma situacional, como, por exemplo, o aumento expressivo de ataques no Brasil devido a liberação de saques do FGTS de contas inativas[2]. Não obstante, o *phishing* também impulsiona o crime cibernético, pois apresenta-se como uma ferramenta para propagação de *malware* na Internet [Khonji et al. 2013]. Um bom exemplo é o *ransonware wannacry*, que gerou problemas em escala global.

Para ilustrar a escala, a Figura 1 registra o número de sites contendo *phishing* confirmados e ainda disponíveis nos últimos 5 anos, no repositório do *PhishTank*[3].

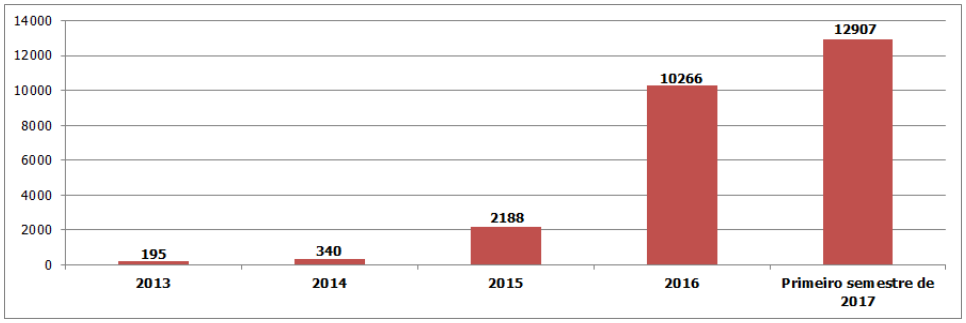


Figura 1. Histórico de registros no *PhishTank* nos ultimos 5 anos

No que tange a detecção de sites com *phishing*, os navegadores *Web* são a primeira linha de defesa e utilizam diferentes mecanismos nativos para identificar e impedir que os usuários caiam em sites maliciosos. A Figura 2 ilustra como esses mecanismos interceptam a ação do usuário, visando impedir o acesso ao *phishing* com base em uma lista negra, sobrepondo a página maliciosa por um página de alerta de perigo.

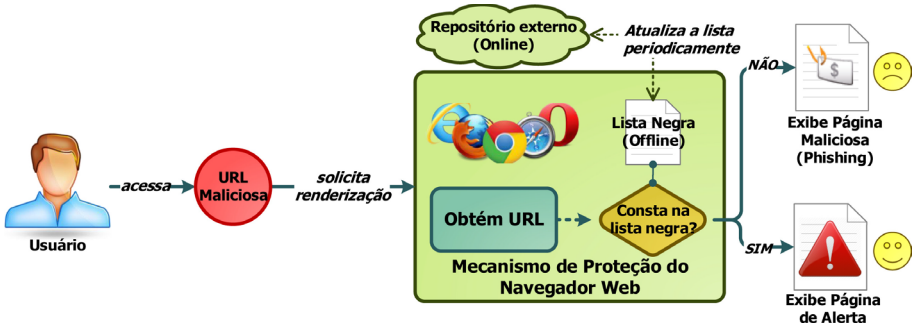


Figura 2. Exemplo do mecanismo de proteção dos navegadores contra *phishing*

Todavia, essas soluções possuem lacunas como, por exemplo, a brevidade na sincronização da lista negra, ou seja, o tempo mínimo de reconhecimento de um *phishing* recém detectado (*zero-day*), além de falhas em recursos do navegador, como *plug-in* e extensões. É neste cenário que este estudo objetiva analisar a eficiência das proteções de *phishing* existentes nos navegadores *Web* mais populares.

[2]Ataques de *phishing* em massa visam roubar informações do FGTS: https://goo.gl/nMS3Wi
[3]PhishTank: http://www.phishtank.com/

© 2017 Sociedade Brasileira de Computação

Para tanto, uma metodologia, dividida em três perguntas, foi elaborada com o objetivo de **investigar a proteção contra *phishing* dos navegadores *Web***. Para tanto, 16.683 URL maliciosas foram submetidas aos mecanismos de proteção nativos dos navegadores, onde **foi observada a propagação e a semântica das respectivas URL**, considerando também o perfil mais explorado por parte dos criadores de *phishing*, através da análise do padrão de propagação presentes em cada URL.

Este artigo está dividido da seguinte forma: na Seção 2 é descrita a problemática, detalhando o cenário de atuação do *phishing* e trabalhos correlatos ao presente estudo. Na Seção 3 é apresentada a proposta de análise, descrevendo escopo, caminhos e justificativas quanto as decisões levantadas. A Seção 4 apresenta os resultados obtidos com a proposta de análise, incluindo um refinamento dos dados extraídos e interpretações. Por fim, na Seção 5 são apresentadas as considerações finais e perspectivas futuras sobre o desenvolvimento da proposta.

2. Cenário de ataque

De acordo com a *Kaspersky* [Kaspersky 2014], *phishing* é um ataque caracterizado por tentativas fraudulentas contra usuários da Internet. Nesse cerne, o atacante desenvolve uma página falsa que apresenta-se como um ambiente confiável, induzindo suas vítimas à submeterem dados sensíveis, como, por exemplo, o preenchimento de formulários com credencias de acesso a um determinado serviço genuíno [Mohammad et al. 2015].

Conforme ilustrado na Figura 3, tudo se inicia através da engenharia social, que apesar de ter um contexto amplo, é o mecanismo que não se restringe apenas a vulnerabilidades técnicas, uma vez que também explora a confiança de suas vítimas. O atacante faz um estudo clínico do recurso computacional e do fator humano envolvido.

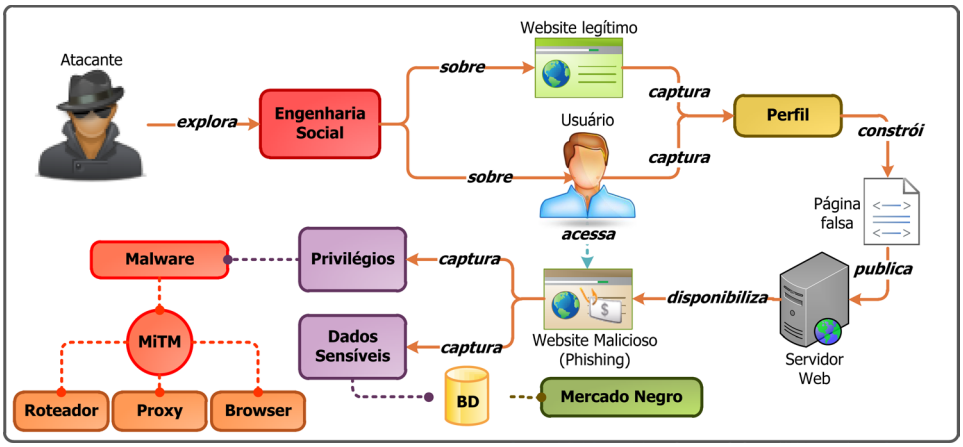


Figura 3. Fluxograma padrão de um *phishing*

Através da engenharia social, o atacante mapeia comportamentos tanto do serviço legítimo (saber que uma determina senha de acesso possui 6 ou 8 dígitos, por exemplo) quanto dos respectivos usuários finais (saber seus interesses e meios de acesso), resultando em uma extração de perfil em cada alvo [Khonji et al. 2013]. Esse perfil se traduz como um conjunto de comportamentos que servem de subsídio para a construção de um site ou aplicativo malicioso. A qualidade desse perfil garante maior similaridade entre o site legítimo e falso, diminuindo as suspeitas de fraude.

© 2017 Sociedade Brasileira de Computação

Diante disso, o atacante começa a atuar como um "homem do meio", do inglês "*Man in The Middle*" (MiTM), e pode ter objetivos divididos em dois segmentos: uma vez que a vítima caia no golpe, a mesma poderá conceder privilégios de acesso em seu dispositivo, através da execução de um *proxy*, extensão do navegador, redirecionamentos maliciosos no roteador ou demais aplicativos maliciosos [Whittaker et al. 2010]. Também pode usar o computador infectado para propagar o site malicioso entre os contatos da vítima ou obter dados sensíveis, como dados de cartão de crédito, os quais o criminoso possa vender como um produto no mercado negro.

2.1. Novos caminhos e tendências

Ao contrário da exploração de vulnerabilidades em aplicações, a exploração por engenharia social é direcionada ao descuido no fator humano, seja de forma direta, induzindo técnicas que interceptem o fluxo do usuário, ou indireta, em que o atacante analisa vulnerabilidades nos recursos computacionais envolvidos. A seguir são descritas algumas técnicas nesse segmento que exploram tecnologia e o fator humano envolvido.

Exploração por *punycode*: Conforme o RFC 3492 [Costello 2003], o *punycode* é um protocolo que permite a conversão de caracteres com *unicode* específico, como o chines ou russo, em uma versão compatível para nomes de domínios DNS. Essa cadeia de caracteres é sempre precedida do prefixo *"xn–"* e a responsabilidade da conversão é atribuída ao navegador *Web*. Por exemplo, ao utilizar o alfabeto cirílico (alfabeto para linguas eslavas), a palavra *apple* em *unicode* cirílico [Jovanovic 2009] convertida em *punycode* resulta em *xn–80ak6aa92e*, ou seja, uma versão ASCII aceita em nomes DNS. Portanto, ao digitar a URL *xn–80ak6aa92e.com* no navegador *Web* serão exibidos os caracteres *apple.com* na barra de endereço, mas o mesmo será apresentado como um domínio não registrado, ou seja, diferente do domínio oficial *apple.com* em ASCII[4].

Outra maneira de sofisticar o golpe é a possível combinação de *unicode* distintos para resultar em uma cadeia de caracteres idêntica ao site genuíno. O fato é que alguns idiomas, a exemplo do cirílico, não possuem caracteres como *g* ou *l* minúsculos, o que dificulta o infrator forjar um conjunto de caracteres que se assemelhem a um domínio como o *google.com*. O mais aproximado seria algo como *GooGle.com*, porém, suscetível à suspeitas em usuários mais atentos. Contudo, nada impede que o infrator use combinações de *unicode* diferentes, como por exemplo usar o *g* e *l* minúsculos do alfabeto latim básico e os demais caracteres do alfabeto cirílico, resultando na cadeia de caracteres *xn–ggl-tdd6ba.com*, que convertida pelo *punycode*, resultaria em *google.com*[5].

Exploração por Domínio de Topo (*gTLD*): Outro meio de exploração de *phishing* é mascarar URL de domínios enganosos através de domínios genéricos [TrendMicro 2017]. Proposto em 2012 pela ICANN, esse recursos possibilitou que organizações proporcionassem URL amigáveis de acordo com o segmento de atuação, a exemplo de empresas como o Bradesco, que atua no segmento bancário e oferecem endereços *http://bradesco.b.br* ou *http://banco.bradesco*. Esse recurso, apesar dos benefícios, também serve como oportunidade para mal intencionados publicarem URL falsas com modestas variações em comparação ao legítimo, fazendo um jogo de palavras ou trocando letras por números, para que a URL falsa se assemelhe a genuína.

[4]Exemplo extraído da notícia publicada em "The Hackers News": https://goo.gl/USO9Vo

[5]Conversor *online* de *punycode* para ASCII: http://idna-converter.com/

 © 2017 Sociedade Brasileira de Computação

Exploração por *Spear Phishing*: Essa modalidade de ataque usa os mesmos princípios de um ataque de *phishing*, contudo, nesse cenário o golpe ocorre de forma direcionada [TrendMicro 2014]. Diferente do meio convencional, com um escopo mais amplo e disperso, o *spear* phishing direciona em um determinado grupo ou organização. O conteúdo do *phishing* costuma ser mais elaborado, além de fazer uso de táticas inteligentes para torna-se mais convincente a vítima da organização em questão. Em suma, essa modalidade tem um análise bem mais minuciosa sobre a extração de perfis das vítimas consideradas como alvos, trazendo assim maiores possibilidades.

Exploração por política regional e localização geográfica: Outro fator oportuno para os crimes cibernéticos é a diversidade na conduta quanto a punições por crimes digitais que são ocasionadas pelas políticas de um determinado estado ou país [ZDNet 2016]. Diversos servidores que armazenam páginas maliciosas encontram-se geograficamente em países onde não existe quase ou nenhuma legislação que responda por crimes dessa natureza. Isso justifica o fato de existir *phishing* que encontram-se ativos desde 2013.

2.2. Trabalhos Relacionados

Akhawe e Felt [Akhawe and Felt 2013] apresentam uma avaliação empírica sobre a ineficiência dos alertas de seguranças quando os navegadores *Web* notificam seus usuários. A pesquisa é direcionada aos avisos de Malwares e *phishing* que são ou que deveriam ser exibidos durante a navegação, trazendo como conclusão a importância de uma eficiente política de alerta destes incidentes, além de evidenciar um significativo impacto destes alertas ao comportamento do usuário final. Na mesma linha, Mohammad et al. [Mohammad et al. 2015] apresentam uma abordagem sobre os principais métodos de ataques por *phishing*, considerando mecanismos de proteção e detecção de *phishing*.

Mazher et al [Mazher et al. 2013] realizam uma avaliação das barras de ferramentas que podem ser instaladas nos navegadores no intuito de proteger o usuário. Esse tipo de solução vem como uma opção adicional de proteção e, apesar de não serem mecanismos nativos, possuem um certo privilégio sobre a navegação do usuário. Esse viés acaba se tornando uma preocupação, porque ferramentas dessa natureza são desenvolvidas por terceiros, dificultando avaliar com precisão a real intenção dessas ferramentas durante a navegação. Jackson et al [Jackson et al. 2007] apresentam uma avaliação dos mecanismos nativos do navegador sobre a proteção de *phishing* considerando a usabilidade em um estudo de caso que envolveu 27 usuários e 12 sites entre as categorias malicioso ou legítimo. O estudo avalia o certificado https e casos de falsos positivos e foi os primeiros passos para o aperfeiçoamento do Chrome em relação a proteção de *phishing*.

O diferencial do estudo desse artigo em comparação aos demais correlatos é que pouco se aprofundou e discutiu nos trabalhos relacionados sobre o meio de propagação, ciclo de vida e morfologia do *phishing*. A proposta também apresenta um grande número de *phishing* e, por se enveredar em uma análise quantitativa para avaliá-los, investiga os repositórios disponíveis, bem como o mecanismo nativo no navegador, além de levantar padrões sobre esse crime através da análise de propagação.

3. Proposta

Essa Seção descreve a metodologia elaborada para um diagnóstico sobre a eficiência dos principais mecanismos de proteção à *phishing* existentes nos navegadores *Web*. É importante frisar que nesta metodologia não foram considerados mecanismos externos ou

© 2017 Sociedade Brasileira de Computação

de terceiros, como extensões ou antivírus, apenas os recursos nativos do navegador. A metodologia proposta é baseada em três questionamentos:

- (Q1) Como **obter** um número considerável de *phishing* disponíveis na *Web*?
- (Q2) Como **submeter** esse número de *phishing* aos mecanismos de proteção dos navegadores *Web*?
- (Q3) Como **analisar** o padrão de propagação e semântica de cada *phishing*?

3.1. Q1: Extrair URL maliciosas dos repositórios de *phishing*

Para responder o Q1 foi realizada uma busca por repositórios de *phishing* com as seguintes características: **atualização constante** e **disponível gratuitamente (pública)**. O intuito desses critérios é possibilitar a criação do cenário de avaliação que retrate o mundo real com um número razoável de *phishing*, para ser capaz de responder aos questionamentos com o máximo de precisão. Para o experimento, foram utilizados dois repositórios: *PhishTank*[6] e *OpenPhish*[7]. O *PhishTank* além de atender os requisitos, possui um número expressivo e detalhado de registros em sua base de dados. O repositório *OpenPhish*, embora restrinja o total acervo da sua base através de um acordo comercial, ainda permite baixar uma versão gratuita. Vale destacar que os repositórios do *SafeBrowsing* e *VirusTotal* foram avaliados e descartados por não disponibilizam sua base de registros abertamente.

Conforme ilustrado na Figura 4, é definido o fluxograma **F1**, utilizado pelo estudo para realizar a extração de *phishing* nos repositórios do *PhishTank* e *OpenPhish*. Como primeiro passo é realizar a leitura dos catálogos *offline* disponíveis pelos repositórios, os arquivos *verified_online.json* e *feed.txt*[8], respectivamente do *PhishTank* e do *OpenPhish*. Em seguida, cada URL é lida, observando se a mesma já se encontra cadastrada no banco de dados do cenário da experimentação. Se a mesma já foi cadastrada, observa-se se o *phishing* encontra-se em atividade (ativo e online) e se sofreu algum tipo de modificação através de uma análise pelo *hash* do *response*. Maiores detalhes dessa atividade serão descritos mais adiante nos fluxogramas **F4** e **F5**.

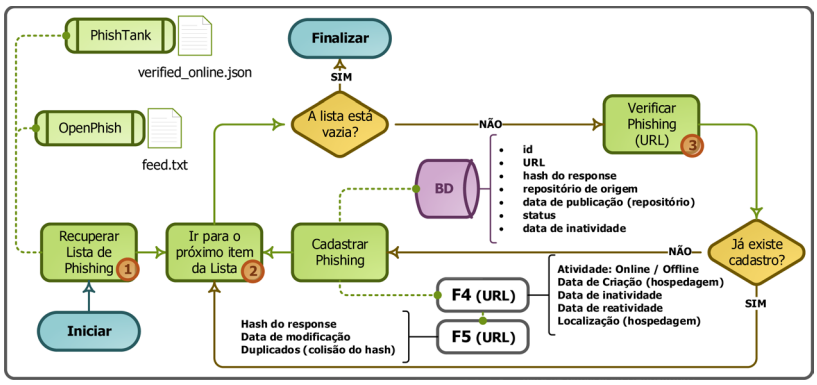


Figura 4. Fluxograma da extração de *phishing* nos repositórios F1

[6]https://www.phishtank.com
[7]https://openphish.com/
[8]*Download* dos arquivos: https://goo.gl/gFCYfJ

© 2017 Sociedade Brasileira de Computação

3.2. Q2: Avaliar os Mecanismos de proteção dos navegadores *Web*

Para responder o questionamento Q2 foi realizada uma análise sobre as proteções nativas do navegador. Para esse estudo, foi considerado o uso dos seguintes navegadores: Google Chrome, Internet Explorer (IE), Mozilla Firefox, Opera e Safari (**Quais versões**). De acordo com a documentação oficial de cada navegador, foi observado que Chrome, Firefox e Safari compartilham o mesmo mecanismo de proteção, intitulado *SafeBrowsing* API, portanto esses representam um conjunto com mesmo resultado. Já o Opera e o IE terão seus resultados individualizados, pois utilizam, respectivamente, o *PhishTank* API e o *Windows Defender SmartScreen*. Com isso, foi considerado para o testes as APIs do *PhishTank* e *SafeBrowsing*. O fato do IE não ter um mecanismo público foi um problema a ser analisado posteriormente, porém, sanado e descrito na Seção 4.

Conforme ilustrado na Figura 5, o primeiro passo do fluxograma **F2** é garantir a posse e sincronização dos dados extraídos nos repositórios de *phishing*. Ao obter a lista de registros, é verificado se o mesmo encontra-se *online*. Esse passo foi incluído para garantir uma redução de falsos negativos para o caso do *phishing* estar hospedado em um servidor que apresente baixa disponibilidade. Ao identificar como *online*, é observado qual seu repositório de origem, uma vez que ele seja do *OpenPhish*, estará candidato a ser avaliado na API do *PhishTank* e do *SafeBrowsing*, caso contrário, será necessário ser avaliado apenas no *SafeBrowsing*.

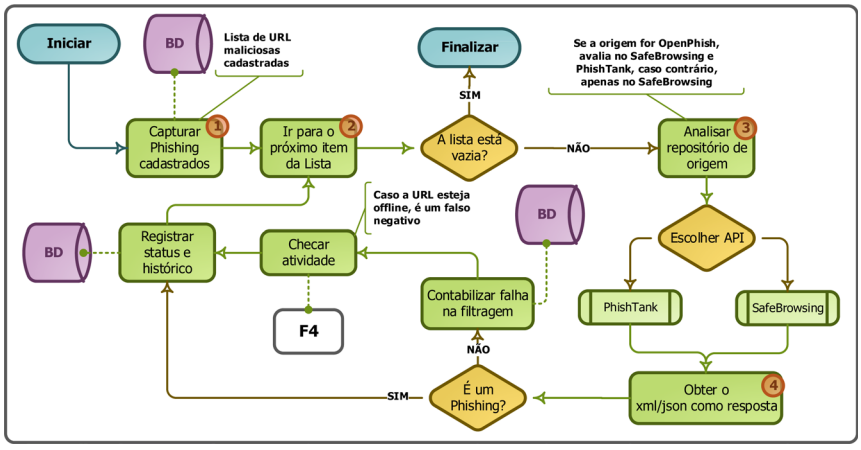


Figura 5. Fluxograma da avaliação da API *PhishTank* e *SafeBrowsing* (F2)

3.3. Q3: Analisar os padrões de propagação e semântica do *phishing*

Por fim, o último passo foi estabelecer uma análise para identificar os padrões de propagação aplicados pelo *phishing*. Como já mencionado na Seção 1, os criadores de *phishing* muitas vezes baseiam-se em determinadas situações eventuais e regionais, como o caso da liberação de FGTS no Brasil. Nesse ponto, o criminoso explora com maior profundidade os perfis mencionados na Figura 3. Para tal objetivo, as URL obtidas no Q1 foram armazenadas em um banco de dados relacional. A justificativa é que posteriormente se fez possível realizar buscas, através de consultas SQL, para a extração de comportamentos com base no conteúdo da URL.

4. Resultados Obtidos

Nessa Seção serão debatidos os resultados obtidos na avaliação proposta pelo estudo.

© 2017 Sociedade Brasileira de Computação

4.1. Resultados de Q1

Conforme detalhado na Tabela 1, foram coletados 25.896 *phishing* no *PhishTank* e 3.776 no *OpenPhish*, totalizando 29.672 *phishing*. Contudo, para a realização do teste, será considerado o primeiro semestre de 2017 com base na data de confirmação do *phishing* como ameaça, resultando em 12.907 registros de *phishing*. Como o repositório do *OpenPhish*, por ser uma versão gratuita, não informa data de publicação e fornece uma lista limitada de registros, foram considerados todos os 4.122 *phishing* disponibilizados nessa versão. A soma final dos repositórios resultou em 16.683 *phishing* para o experimento. É importante observar que **346** *phishing* estavam presentes em ambos os repositórios, estes foram descartados do *OpenPhish*, reduzindo para 3.776 ocorrências nesse repositório.

Tabela 1. Phishing obtidos nos repositórios

Repositório	**Total de registros**	**Registros de 2017**	**Registros *Online* de 2017**	**Registros *Offline* de 2017**
PhishTank	25.896	12.907	12.390	517
OpenPhish	3.776	3.776	3.748	28
-	**29.672**	**16.683**	**16.138**	**545**

4.2. Resultados de Q2

Não são incomuns os casos em que os repositórios disponibilizam, além de seus registros, uma interface programável (API) para diagnosticar se uma determinada URL trata-se de um *phishing*. Esse tipo de mecanismo favorece cenários de testes da natureza desse estudo, pois representam uma drástica redução de tempo na tarefa de avaliar milhares de URL suspeitas, podendo o trabalho ser realizado em um laço de repetição de um *software*.

A API do *PhishTank* disponibiliza suas consultas sem qualquer tipo de limitação e de forma gratuita. Assim como *PhishTank*, o *SafeBrowsing* também disponibiliza sua API de forma aberta, apenas estabelece uma cota generosa de 12 mil requisições diárias por cada *API key*. Já as APIs do *VirusTotal* e o *OpenPhish* restringem o acesso através de uma licença comercial, o que inviabilizou a utilização das mesmas nesse estudo. O intuito do estudo foi, com base nos registros extraídos em repositórios, avaliar a taxa de acerto de cada API disponível para o teste. O resultado da análise das API com relação a detecção dos *phishing* pode ser observado na Tabela 2. O fato de utilizar *phishing online* (12.390 ao invés de 12.907 no caso do *PhishTank* e 3.748 ao invés de 3.776 no caso do*OpenPhish* é devido aos falsos negativos apresentados pelas API quando checam uma URL *offline*.

Tabela 2. Avaliação das API em confronto aos repositórios

Repositório	**API**	**Total de *phishing***	**Identificados**	**% Acerto**
PhishTank	*SafeBrowsing* API	12.390	7.824	63,15%
OpenPhish	*PhishTank* API	3.748	2.572	68,62%
OpenPhish	*SafeBrowsing* API	3.748	2.265	60,43%

Com esse resultado, teoricamente, era possível concluir que qualquer navegador que utilizasse o *SafeBrowsing*, uma vez submetido aos 12.390 *phishing* (online) do *PhishTank*, também teria uma percentual de acerto de 63,15%, pelo fato da API em questão ser

 © 2017 Sociedade Brasileira de Computação

a mesma adotada pelo seu mecanismo de proteção. Contudo, durante os testes essa teoria não se concretizou devido a uma série de divergências identificadas no experimento.

Primeiramente, foi observado que uma URL na API do *PhishTank* não era reconhecida como *phishing*, mas quando aberta no Opera, era reconhecida como maliciosa. O mesmo ocorreu entre o *SafeBrowsing* e o *Firefox*. O que explica essa diferença de comportamento entre a API e o navegador é a utilização da lista negra adotada pelo navegador, conforme já ilustrado na Figura 1. Por exemplo, o Opera armazena uma lista negra com as URL maliciosas. Periodicamente, o navegador consulta na API do *PhishTank* por novas URL a serem inseridas na lista. Uma vez atualizada, o navegador fará uso da lista para realizar a filtragem, esteja a máquina do usuário *online* ou *offline*.

Já no caso da API, a mesma, aparentemente, confere se o *phishing* está disponível (online), caso não esteja, não é considerado como uma ameaça. Tal comportamento justifica a divergência nos resultados dos testes entre testar diretamente na API e no navegador. Outra questão é a divergência entre o filtro de cada API. Durante o estudo, certas URL foram reconhecidas como *phishing* pela API do *PhishTank*, mas na API do *SafeBrowsing* não eram identificadas [9]. Outro fator é que as API não tratam situações onde URL maliciosas são encapsuladas por URL curtas, já no navegador essas URL seriam detectadas quando o site encurtador redirecionasse o usuário para o destino.

Por fim, também foi observado que algumas URL tinham comportamentos distintos entre o *Chrome*, *Firefox* e *Safari*, mesmo estes compartilhando um único mecanismo de proteção, sugerindo uma possível proteção adicional em cada navegador. O motivo desse comportamento não pode ser identificado por esse experimento, mas diante essas observações, o estudo motivou o desenvolvimento de uma avaliação adicional que considerou a checagem diretamente no navegador. Outro fato que reforça essa decisão é que a proteção do IE, a *SmartScreen*, não disponibiliza uma API externa para consultas nem tampouco um repositório de *phishing* registrados.

Para tanto, foi proposto um novo fluxograma **F3** de coleta de informações, conforme ilustrado na Figura 6. Ele possui um fluxo bem similar ao fluxograma F2, contudo, ao identificar um *phishing* como *online*, é criada uma instância de um determinado navegador e o fluxo captura o corpo da página carregada durante o acesso ao *phishing*. Diferentemente do teste com as API, pelo fato do navegador filtrar um *phishing* independente do mesmo estar *online* ou *offline*, os *phishing offline* também foram considerados no fluxograma F3, checando um total de 12.907 para os testes confrontados com o repositório *PhishTank* e 3.776 quando confrontados com o *OpenPhish*.

Considerando que o cenário irá acessar milhares de *phishing* simultaneamente, foi preciso estabelecer um cenário de testes automatizados. A ferramenta escolhida para proporcionar essa automação foi a *Selenium Server*[10], que oferece automação de testes diretamente no navegador. Também foi utilizada a biblioteca *Selenium WebDriver*, um mecanismo que realiza os testes definidos em uma instância do navegador, dando suporte a praticamente todos os navegadores disponíveis. Para garantir maior precisão nos resultados, após a execução dos testes, foi realizada uma triagem no *LOG* resultante da conclusão da aplicação que invoca as instâncias do navegador, verificando assim o motivo

[9] Por exemplo, a URL http://notariato.info/tmp/mod_cust/ (Testado em 17/07/2017)
[10] http://docs.seleniumhq.org/download/

© 2017 Sociedade Brasileira de Computação

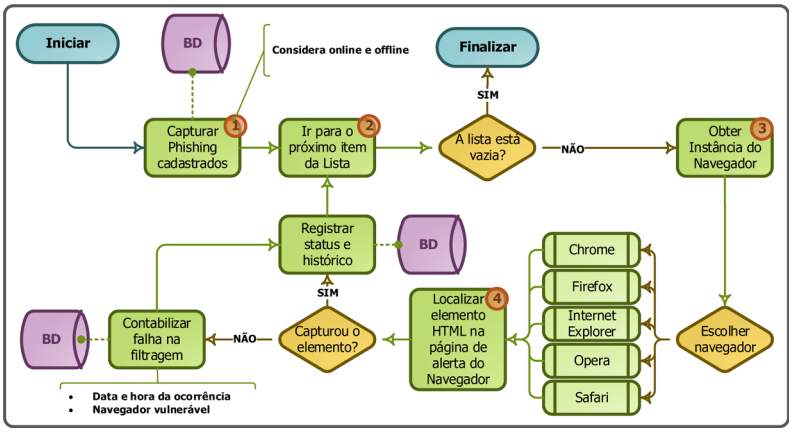


Figura 6. Fluxograma de avaliação nos navegadores F3

das falhas nos métodos, para confirmar se o teste ocorreu conforme o esperado.

O próximo passo é identificar se, ao invés da página do *phishing*, foi exibida a página de alerta do navegador. Esse tratamento precisou ser adaptado de acordo com o navegador utilizado, pois cada um possui uma representação diferente diante o acesso ao *phishing*. Porém, ambos compartilham o mesmo comportamento de sobrepor a página maliciosa com uma página de alerta. É possível obter elementos dessa página de alerta com a biblioteca Selenium WebDriver, tal busca pode ser feita pelo atributo ID de algum elemento HTML existente. Como exemplo, é possível localizar um botão intitulado "Me tire daqui!" que possui *ID "getMeOutButton"* no Firefox ou *"goToHomePage"* no IE.

Já no Opera a checagem era feita pelo texto *"Aviso de Fraude"* no título da página. Da mesma forma, Chrome e Safari apresentam como título o texto *"Erro de Segurança"*[11]. Por fim, se o elemento ou título fosse reconhecido, significaria que a tela de alerta havia sido exibida, indicando que o filtro foi bem sucedido. Caso contrário, seria registrada uma ocorrência de falha na filtragem do navegador. O resultado desse experimento pode ser observado nos gráficos da Figura 7.

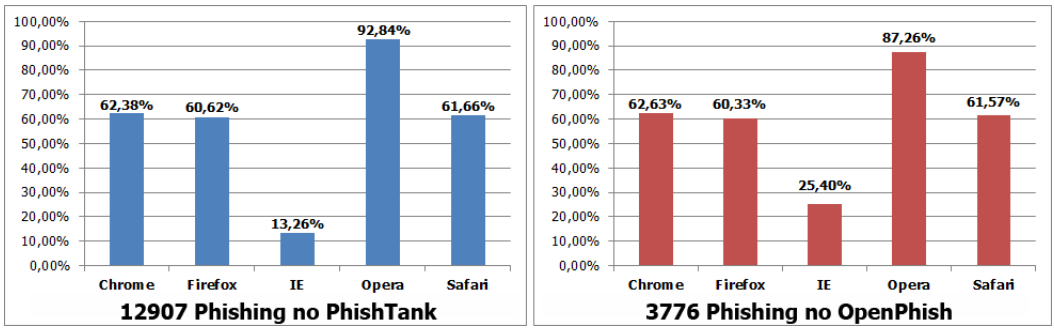


Figura 7. Resultados obtidos na filtragem dos navegadores

4.3. Resultados de Q3

Por fim, foi realizada a análise de padrões nas URL dos *phishing* obtidos na Q1. Foi observado uma grande ocorrência de palavras-chave, revelando o perfil de maior interesse

[11]A busca foi feita pelo título da página porque o navegador impedia o acesso aos detalhes sobre o conteúdo da página de alerta.

© 2017 Sociedade Brasileira de Computação

pelos mal intencionados, que se envereda em transações financeiras e redes sociais. Esse comportamento pode ser extraído através de uma análise de propagação na URL com base dos repositórios através de consultas SQL na base de dados com os *phishing* extraídos de Q1. O resultado do experimento pode ser observado na Tabela 3.

Tabela 3. Análise dos padrões dos *phishing* (na URL)

Palavra-chave	Ocorrências	Palavra-chave	Ocorrências	Palavra-chave	Ocorrências
login	3.596	domínios .br	1.590	domínios .org	1.345
https	1.266	google	916	paypal	596
card	529	mobile	368	boleto	302
banco	291	facebook	287	cadastr*	269
download	217	payment	204	billing	182
cert*	168	santander	154	banking	136

Observando a tabela 3, é possível perceber a quantidade de domínios que teoricamente possuem um processo de registro mais restrito, a exemplo dos domínios *.br* e *.org.*, mas ainda sim são amplamente explorados e, muitos, até o momento, sem uma postura corretiva sobre o fato. O primeiro caso evidencia o interesse dos mal intencionados em atuar no Brasil. Já o segundo pode representar a vista grossa em determinados países devido a postura política sobre o caso, conforme mencionado a Seção 2. Outra teoria seria os casos de criminosos que conseguem quebrar a segurança dos sites governamentais através de vulnerabilidades existentes, conseguindo realizar *upload* de páginas maliciosas em servidores oficiais, dificultando identificar o *phishing* com base no domínio.

4.4. Ameaças da avaliação

Como em qualquer tipo de experimento, a automação pode ocasionar em riscos na precisão dos resultados obtidos. Contudo, o estudo considera que a confiabilidade da metodologia adquiriu considerável precisão com a decisão de eliminar o caráter subjetivo da avaliação, no caso, a intervenção manual. E diante do contexto desse estudo não há nada que um ser humano possa fazer que a automação não seja capaz de reproduzir.

Quanto à **limitação de escopo**, no estado atual, foram considerados os *phishing* com o status "confirmado" no primeiro semestre de 2017, contudo, foi suficiente para ter um considerável número de URL maliciosas para análise. A justificativa de considerar os *phishing* mais recentes é que a grande relevância do filtro se dar por sua rápida resposta sobre um novo *phishing* que surge, ou seja, o menor tempo da janela de vulnerabilidade.

Em relação aos **falsos positivos** e **falsos negativos**, cada mecanismo avaliado tem suas particularidades. Primeiramente, é preciso considerar que os repositórios dependem da intervenção humana (denúncia) para cada *phishing*. Diante disso, é notória a dificuldade de responder à *phishing* recém-criados ou que não foram denunciados. Com o estado atual do estudo, seria possível fazer uma varredura pela *Web* em busca de *phishing* através da colisão de *hash*, identificando *phishing* não denunciados, porém, replicados em outro servidor. Contudo, o *phishing* poderia apresentar pequenas mudanças em suas variações, que apesar de sutis, seriam suficiente para modificar o *hash*.

Para tanto, como trabalho futuro, é pretendido desenvolver uma **análise morfológica** para acompanhar o comportamento de cada *phishing* durante um determinado

© 2017 Sociedade Brasileira de Computação

período. Não obstante, em certos casos, reconhecer um *phishing* através de seu *hash* pode ter suas limitações, já que existem diversas formas de atingir um *layout* semelhante a página genuína. Diante disso, um reconhecimento por imagem poderia ser interessante. Em suma, tais técnicas visam minimizar a dependência com os repositórios existentes ou mesmo contribuir com denuncias, melhorando-os.

Outro fato que reforça a preocupação dessa dependência é que existem casos de *phishing* publicados no repositório que ainda não passaram pelo crivo da administração da plataforma, representando uma janela de vulnerabilidade no repositório e, consequentemente, na API. Por exemplo, o *PhishTank*, por ser uma plataforma colaborativa onde qualquer pessoa denuncia de forma irrestrita, o *phishing* uma vez cadastrado, é classificado como "não confirmado", pois será submetido à uma análise manual por parte dos administradores do *PhishTank* para então ser uma ameaça "confirmada". Esse fato constata um atraso significativo da plataforma sobre um *phishing* que representa um perigo real ao usuário final, evidenciando o problema de falsos negativos. Contudo, a análise manual é um preço que se paga para evita falsos positivos.

Diante o estado atual do estudo, seria possível comparar a data do inicio de atividade do *phishing* no servidor de hospedagem (*online*) com a data de publicação do *phishing* na plataforma. Enquanto não lhe for atribuído o *status* "confirmado", nenhuma URL é reconhecida como uma ameaça, seja pela API ou navegador, proporcionado a janela de vulnerabilidade. Durante o experimento, também foi observado que a API não reconhecia uma URL como um *phishing* diante uma das situações: erro 404 (*offline*), conta suspensa ou *timeout* no servidor. Contudo, é sensato considerar que o site malicioso *offline* eventualmente poderá voltar a atividade, seja por correção na execução da página (*timeout*) ou contas reativadas no servidor. Com base nesses aspectos, é pretendido desenvolver uma **análise do ciclo de vida** de cada *phishing* como trabalhos futuros.

O problema também versa no contexto dos navegadores. Por exemplo, na API do *PhishTank*, determinados *phishing* foram de fato confirmados como positivos no repositório, mas ainda não estavam sendo reconhecidos como uma ameaça quando acessados através do navegador. O motivo seria o atraso na sincronização da lista negra do navegador com os registros do respectivo repositório, representando assim uma janela de vulnerabilidade através do uso do navegador. Contudo, para maior precisão sobre o atraso na sincronização da lista negra, seria necessário comparar em cada *phishing*, a data e hora da falha contra a data e hora da confirmação no repositório, obtendo uma diferença.

Outro ponto, é que os testes foram baseados em navegadores para *desktop*, teoricamente, pode ser considerado que os resultados seriam aplicados à dispositivos móveis. De toda forma, talvez fosse interessante comparar entre as versões de cada plataforma. Todavia, existem certos aspectos da computação móvel, como repositório de aplicativos, que merecem análises voltadas para esse cerne. Por fim, uma limitação da análise de propagação é que a mesma inspecionou apenas a composição da URL, o interessante seria considerar elementos textuais e da páginas, uma vez que nem todo *phishing* contem em sua URL palavras-chave que façam referência aos objetivos do site alvo.

5. Conclusão

Dentre os 5 navegadores avaliados, nenhum teve mais que 90% de acerto quando comparado em um repositório diferente de seu mecanismo de proteção. Os casos em que o

 © 2017 Sociedade Brasileira de Computação

navegador foi avaliado em um repositório distinto ao do seu mecanismo, como no teste do Opera com o *OpenPhish*, o navegador teve um desempenho de 87,26%. Já a API do repositório do Opera, *PhishTank*, quando confrontada com o *OpenPhish*, resultou em 68,62% de acerto, evidenciando uma possível proteção adicional no navegador. Contudo, o Opera atingiu 92,84% de acerto enquanto confrontado por seu respectivo repositório, o que indica possíveis falhas de falsos negativos. De fato, essas evidências sugerem uma pesquisa com maior profundidade na investigação sobre essa proteção adicional.

Em termos de baixa ineficiência da filtragem, o que chama atenção é a taxa do IE com o *SmartScreen*, 13,26% no *PhishTank*, bem abaixo dos concorrentes. Apesar das limitações mencionadas sobre o *SmartScreen*, um ponto positivo é que o mesmo pode, adicionalmente, ser integrado aos aplicativos do pacote *Microsoft office*, a exemplo do cliente de *e-mail Outlook*, podendo assim minimizar as explorações por *e-mail* mencionadas na Seção 2. Contudo, essa funcionalidade não reduz os riscos atuantes no navegador. Considerando os resultados dos testes sobre as proteções disponíveis, foi observado que o mecanismo nativo dos navegadores apresentam sérias limitações, sugerindo aos seus usuários a necessidade de considerar soluções *endpoint*, a exemplo de antivírus.

Como trabalhos futuros, o estudo pretende preencher lacunas observadas na Seção 4.4. Conforme ilustrado no fluxograma **F4** da Figura 8, o próximo passo do estudo é realizar uma **análise do ciclo de vida** de cada *phishing*, visando um monitoramento do tempo de criação e atividades posteriores. Na mesma linha, com o fluxograma **F5** da Figura 9 é pretendido realizar uma **análise morfológica** para monitorar as modificações sofridas pelo *phishing* durante um período. Para facilitar a organização e estruturação, diversos dados extraídos nos repositórios de *phishing* (Q1) foram armazenados na base de dados do experimento.

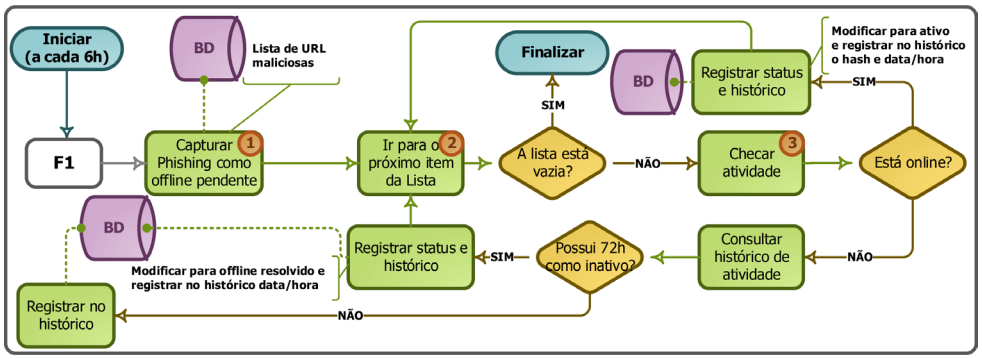


Figura 8. Fluxograma da análise de ciclo de vida do *phishing* F4

Para o ciclo de vida, será registrado datas de eventos do *phishing*. Uma delas é a data de publicação no repositório, obtida no repositório de origem. Porém, a mesma não sugere o inicio da atividade do *phishing*. Para isso, a informação pode ser obtida através do protocolo *WHOIS*[12], que oferece informações sobre o proprietário do domínio, data de publicação, expiração e última atualização do domínio. Por fim, a data de inatividade auxilia na análise do ciclo de vida do *phishing* durante sua atuação na Web.

Quanto a sua morfologia, serão consideradas informações como o conteúdo rece-

[12]WHOIS: https://tools.ietf.org/html/rfc3912

© 2017 Sociedade Brasileira de Computação

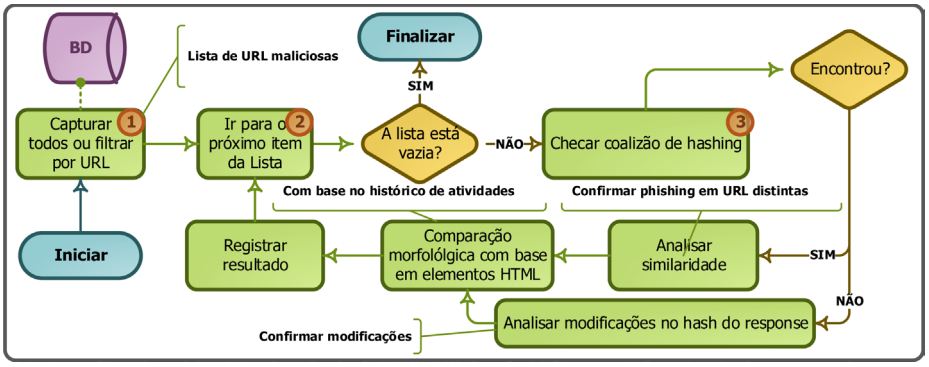


Figura 9. Fluxograma da análise de morfologia do *phishing* F5

bido como resposta ao requisitar a URL maliciosa, que será convertida em um calculo de *hash SHA-256* para facilitar na identificação de URL que apontem para um mesmo *phishing*, através de colisão de *hash*.

Referências

Akhawe, D. and Felt, A. P. (2013). Alice in warningland: A large-scale field study of browser security warning effectiveness. In *22Nd USENIX*, SEC'13, pages 257–272.

Cert.BR (2012). Golpes na internet. *Disponível em: https://cartilha.cert.br/golpes/*.

Costello, A. M. (2003). Punycode: A bootstring encoding of unicode for internationalized domain names in applications (idna). *Disponível em: https://tools.ietf.org/html/rfc3492*.

Jackson, C., Simon, D. R., Tan, D. S., and Barth, A. (2007). An evaluation of extended validation and picture-in-picture phishing attacks. In *Financial Cryptography*, volume 4886 of *Lecture Notes in Computer Science*, pages 281–293. Springer.

Jovanovic, G. (2009). Standardization of the old church slavonic cyrillic script and its registration in unicode. *Disponível em: http://www.unicode.org/charts/PDF/U0400.pdf*.

Kaspersky (2014). O que é phishing? *Disponível em: https://goo.gl/4EEtxk*.

Khonji, M., Iraqi, Y., and Jones, A. (2013). Phishing detection: A literature survey. *IEEE Communications Surveys and Tutorials*, 15(4):2091–2121.

Mazher, N., Ashraf, I., and Altaf, A. (2013). Which web browser work best for detecting phishing. In *2013 5th ICICT*, pages 1–5.

Mohammad, R. M., Thabtah, F., and McCluskey, L. (2015). Tutorial and critical analysis of phishing websites methods. *Comput. Sci. Rev.*, 17(C):1–24.

TrendMicro (2014). O que é spear phishing? *Disponível em: https://goo.gl/GhzUFY*.

TrendMicro (2017). Domínios genéricos e sites fraudulentos: uma conexão perigosa. *Disponível em: https://goo.gl/r9gAZm*.

Whittaker, C., Ryner, B., and Nazif, M. (2010). Large-scale automatic classification of phishing pages. In *NDSS '10*.

ZDNet (2016). Phishing-as-a-service is making it easier than ever for hackers to steal your data. *Disponível em: https://goo.gl/6UST6u*.

 © 2017 Sociedade Brasileira de Computação

Marca d'água estruturada

Lucila Maria de Souza Bento[3] **, Davidson Rodrigo Boccardo**[3]**,**
Raphael Carlos Santos Machado[*4,6] **, Vinícius Gusmão Pereira de Sá**[1] **e**
Jayme Luiz Szwarcfiter[†1,2,5]

[1] Instituto de Matemática – Universidade Federal do Rio de Janeiro (UFRJ)

[2]COPPE – Universidade Federal do Rio de Janeiro

[3]Clavis Segurança da Informação

[4]Instituto Nacional de Metrologia, Qualidade e Tecnologia (INMETRO)

[5]IME – Universidade do Estado do Rio de Janeiro (UERJ)

[6]PPCIC - CEFET/RJ – Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

lucila, davidson@clavis.com.br, rcmachado@inmetro.gov.br,
vigusmao@dcc.ufrj.br, jayme@nce.ufrj.br

Abstract. *A digital object watermark corresponds to embedded identification data which can be timely retrieved to reveal authorship/ownership, with aims at discouraging piracy. Software watermarking schemes based on immersing an encoded key into control flow graphs have been suggested in the literature. Considerable efforts have lately been put up into improving such schemes with respect resilience to attacks. We propose a novel graph-based watermarking scheme with two distinctive features: first, our encoding algorithm employs randomization; second, and more importantly, our watermarks conform to structured code. The ability to encode the key in distinct forms and the absence of awkward goto-like substructures lend our watermarks greater diversity and stealthiness, making them more resilient to subtraction and distortion attacks. A linear-time implementation is given.*

Resumo. *Uma marca d'água em artefato digital corresponde a uma informação de identificação embarcada naquele objeto de forma oculta, podendo ser usada para comprovar autoria/propriedade, com o objetivo de desencorajar a distribuição ilegal de software. Dentre as técnicas de marca d'água de software apresentadas na literatura, destacam-se os esquemas baseados em grafos, nos quais uma chave secreta é codificada e inserida no grafo de fluxo de controle do programa. Recentemente têm sido realizados diversos esforços para melhorar esses esquemas com relação à resiliência a ataques. Neste artigo, apresentamos um novo esquema de marca d'água baseado em grafos com duas características principais: nosso algoritmo de codificação emprega aleatoriedade; e, o que é mais importante, nossas marcas d'água estão em conformidade com códigos estruturados. A capacidade de codificar uma mesma*

[*]Autor apoiado por CNPq e FAPERJ.
[†]Autor apoiado pelo CNPq.

 © 2017 Sociedade Brasileira de Computação

chave de distintas formas e a ausência de subestruturas semelhantes a peculiares goto*'s conferem maior diversidade e furtividade às nossas marcas d'água, tornando-as mais resilientes a ataques de subtração e distorção. Apresentamos também uma implementação em tempo linear.*

1. Introdução

Marcas d'água vêm sendo utilizadas ao longo dos séculos para estabelecer autenticidade, autoria ou propriedade de objetos. No início da década de 1990, esse conceito foi introduzido no contexto de proteção de software, como meio de impedir, ou pelo menos desencorajar, a pirataria de software [Davidson and Myhrvold 1996]. Até esse momento, muitas pesquisas sobre marca d'água de software foram realizadas e empregaram técnicas distintas, incluindo predicados opacos, alocação de registradores, interpretação abstrata e caminhos dinâmicos [Arboit 2002, Collberg et al. 2004, Cousot and Cousot 2004, Monden et al. 2000, Nagra and Thomborson 2004, Qu and Potkonjak 1998, Zeng et al. 2012]. A informação exata que é incluída no software pode variar entre identificação de autoria até identificação de propriedade, neste caso para que uma auditoria feita em cópias ilegais do software permita saber de qual cópia autêntica aquelas se originaram. Idealmente, uma marca d'água deve ser de difícil remoção.

Os esquemas de marcas d'água baseados em grafos consistem em algoritmos de codificação/decodificação (*codecs*) que traduzem informações de identificação na forma de um grafo, que em geral pertence a alguma classe especialmente escolhida, e algoritmos de embarcação/extração para inserir/reconhecer marcas d'água existentes em um software. O primeiro esquema de marca d'água para proteção de software logo veio a inspirar a criação de um esquema baseado em grafos, no qual a marca d'água é embarcada no grafo de fluxo de controle (GFC) do software a ser protegido [Venkatesan et al. 2001]. O GFC, que pode ser obtido por ferramentas para análise estática de código, representa todas as sequências de computações possíveis das intruções do programa na forma de um grafo direcionado, cujos vértices são blocos de código estritamente sequenciais e as arestas indicam possíveis relações de precedência entre esses blocos. O algoritmo de embarcação, basicamente, constrói código fictício que será inserido no corpo do programa, a partir de alguma posição secreta predefinida. Este código corresponde exatamente à estrutura da marca d'água pretendida, de modo que o GFC do programa possua um trecho que corresponda exatamente ao grafo que codifica a marca d'água desejada.

Apesar dos algoritmos de embarcação/extração de marcas d'água estarem atualmente num nível razoável de desenvolvimento [Chroni and Nikolopoulos 2012b, Collberg and Nagra 2009], os *codecs* propostos até o momento deixam bastante espaço para melhorias no que se refere à resiliência a ataques. Dentre os modelos de ataque mais estudados, dois deles exigem atenção especial: ataques de distorção e de subtração [Collberg and Thomborson 1999]. No modelo de ataque de distorção, o adversário tenta danificar a estrutura da marca d'água realizando alterações no código do programa, a fim de remover alguns blocos de código ou ligação entre eles, isto é, o adversário tenta indiretamente remover vértices e arestas do GFC do programa marcado. Uma alternativa para mitigar esse tipo de ataque consiste no uso de códigos de correção de erros amplamente conhecidos na literatura [Reed and Solomon 1960, Mann 1968, Purser 1995, Guruswami 2005]. No ataque de subtração, o adversário detecta a presença

 © 2017 Sociedade Brasileira de Computação

da marca d'água e tenta removê-la completamente. Para evitar esse tipo de ataque, a marca d'água deve ser tão parecida quanto possível com o código do software protegido, para que o adversário tenha dificuldade em identificá-la após uma ação de engenharia reversa no binário do software.

Recentemente, Chroni e Nikolopoulos apresentaram um *codec* [Chroni and Nikolopoulos 2012a] em que a chave é codificada em um grafo pertencente à classe dos grafos de permutação redutíveis [Collberg et al. 2003], possuindo alguma resiliência a ataques [Bento et al. 2013]. Tais grafos são uma subclasse dos GFCs, o que os torna passíveis de serem incorporados ao GFC de um programa. No entanto, sua topologia não corresponde à de um GFC de um *programa estruturado*, uma vez que, para fazer surgir algumas de suas arestas, é frequentemente necessária a adição de saltos (*jumps*) artificiais no programa, isto é, acaba-se tendo que recorrer a instruções pouco usuais (e muito pouco recomendáveis), como o *goto* [Dijkstra 1968]. Como já mencionamos, uma propriedade desejada em uma marca d'água é sua capacidade de estar bem disfarçada no código original, impedindo que seja encontrada e removida. Como as marcas d'água produzidas por este *codec* só podem ser inseridas no código do programa por meio de tais instruções peculiares, estas tenderiam a gerar suspeitas, aumentando as chances de sucesso de um adversário.

Neste artigo, propomos um novo *codec* para marca d'água baseada em grafos. O *codec* proposto utiliza randomização para obter *diversidade*, uma propriedade tida como de grande importância pela comunidade [Collberg and Nagra 2009]. Em resumo, a estrutura das marcas d'água produzidas pelo nosso *codec* é definida a partir de escolhas aleatórias realizadas durante a execução do algoritmo de codificação. Esta característica torna menos provável que uma marca d'água possa ser detectada através de ataques de comparação de força bruta (realizados por ferramentas de *diff*) entre diferentes programas marcados pelo mesmo autor ou para o mesmo proprietário. Uma versão rudimentar desse *codec* for apresentada em [Bento et al. 2014]. O diferencial das marcas d'água apresentadas aqui está no fato de poderem finalmente ser embarcadas adicionando-se código fictício *estruturado* — como definido por Dijkstra [Dahl et al. 1972] — no programa original. Em outras palavras, ao se proteger um programa estruturado, o GFC resultante, com a marca d'água embarcada, continuará pertencendo à classe dos grafos que correspondem a programas estruturados (grafos de Dijkstra), como estudado em [Bento et al. 2015].

Na Seção 2, apresentamos o novo *codec*, ilustrado a seguir por um exemplo completo (codificação/decodificação), na Seção 3. Na Seção 4, descrevemos uma implementação em tempo linear para os algoritmos de codificação e decodificação. A Seção 5 contém nossas considerações finais.

Seja G um grafo. Denotaremos por $V(G)$ e $E(G)$, ao longo do texto, o conjunto de vértices e o conjunto de arestas de G, respectivamente. Para $u, v \in V(G)$, uma aresta de u a v é representada $[u \rightarrow v]$, ou, equivalentemente, $[v \leftarrow u]$. Um *caminho* em um grafo G de tamanho k é um sequência de vértices $v_0, v_1, \ldots, v_k$ tal que $[v_{i-1} \rightarrow v_i] \in E(G)$ para $i = 1, 2, \ldots, k$. Um caminho v_0, v_1, l, v_k, com $k > 1$, é um *ciclo* de tamanho $k + 1$ se $[v_0 \rightarrow v_k] \in E(G)$. Um caminho *hamiltoniano* em um grafo G é um caminho no qual todos os vértices de G aparecem exatamente uma vez.

© 2017 Sociedade Brasileira de Computação

2. Marca d'água estruturada

Nesta seção, apresentamos nossa proposta de *codec* para marca d'água de software estruturado. Tendo em vista as características desejadas em um esquema de marca d'água (resiliência, furtividade, diversidade, eficiência em tempo e espaço), adiantamos algumas das propriedades principais do *codec* que será a seguir proposto.

- Os grafos produzidos pelo algoritmo de codificação pertencem à classe dos GFCs de programas estruturados.
- Uma aresta removida da marca d'água proposta pode corromper no máximo um bit da representação binária da chave codificada, não interrompendo o processo de decodificação. Assim sendo, ataques de distorção podem ser detectados *depois* da decodificação através de mecanismos usuais para detecção e correção de erros, de forma que se possa detectar e corrigir uma marca d'água em que até b bits, para b arbitrário, tenham sido corrompidos — desde que, evidentemente, um número adequado $f(b)$ de bits de redundância tenha sido acrescentado à chave anteriormente à codificação.
- O algoritmo de codificação executa algumas ações de maneira randomizada, o que significa que uma mesma chave pode originar marcas d'água diferentes (em execuções distintas do algoritmo).
- Os algoritmos de codificação e decodificação podem ser implementados em tempo e espaço lineares no tamanho (quantidade de bits) da chave.

2.1. Codificação

O Algoritmo 1 descreve o passo-a-passo da codificação. Em primeiro lugar, é criado um caminho hamiltoniano H com os $n + 2$ vértices (iniciais) de G, dispostos em ordem crescente de rótulo ao longo do caminho. Durante a execução do algoritmo, novas arestas (e, possivelmente, vértices) serão acrescentadas. As arestas do grafo obtido podem ser divididas em três grupos: *arestas de caminho*, que são as arestas que compõem H; *arestas de retorno*, que são arestas que ligam um vértice v a algum vértice anterior a v em H; e *arestas de avanço*, que são arestas que ligam um vértice v ao vértice $v + 2$.

O primeiro vértice do caminho hamiltoniano H, de rótulo 1, sempre irá corresponder a um bit '1' em B (pois toda chave, em binário, começa com um bit '1'), e seu grau de saída será sempre 1 (a aresta do caminho hamiltoniano é sua única aresta de saída).

Um vértice é considerado *mudo* quando tem função meramente estrutural, não estando mapeado diretamente a um bit da chave, e sendo portanto ignorado pelo algoritmo de decodificação. Os dois últimos vértices de H são vértices mudos, assim como são mudos os vértices que são destinos de arestas de avanço. A cada vez que um vértice $v+2$ é tornado mudo pelo algoritmo (através da inclusão da aresta $[v \rightarrow v + 2]$), um novo vértice é acrescentado ao grafo (e ao caminho hamiltoniano original, com a adição de nova aresta), de forma que o número de vértices não-mudos do grafo permaneça sempre igual 'a quantidade n de bits da chave.

Uma aresta de retorno $[w \leftarrow v]$ é considerada *admissível* em G quando são satisfeitas as cinco condições seguintes:

(i) $w = v - 1$, caso exista em $E(G)$ a aresta de avanço $[v - 1 \rightarrow v + 1]$;
(ii) não existe em $E(G)$ a aresta de retorno $[w \leftarrow v - 1]$;

 © 2017 Sociedade Brasileira de Computação

Algoritmo 1: Codificação de marca d'água estruturada

Entrada: uma chave inteira ω a ser codificada
Saída: um grafo de Dijkstra G (marca d'água) codificando ω

1. seja B a representação binária de ω, e seja $n = |B|$
2. crie um grafo G, com conjunto inicial de vértices $V(G) = \{1, \ldots, n+1\}$
   e conjunto inicial de arestas $E(G) = \{[v \to v+1]$, para $1 \leq v \leq n+1\}$
3. seja $i = 2$ (o *índice corrente*), e seja $v = 2$ (o *vértice corrente*)
4. **enquanto** $i \leq n$:
5. **se** existe uma aresta de avanço $[v-2 \to v]$:
6. faça $v = v+1$, de forma a pular o vértice mudo v
7. **continue** para a próxima iteração do laço, mantendo o mesmo valor do índice corrente i
8. **se** existe pelo menos uma aresta de retorno admissível com origem em v:
9. **se** $v-1$ é a origem de uma aresta de avanço $[v-1 \to v+1]$:
10. **se** $B[i] =$ '1':
11. adicione a aresta de retorno $[v-1 \leftarrow v]$
12. remova a aresta de caminho $[v \to v+1]$
13. // os vértices $v-1, v, v+1$ corresponderão a // um bloco *while* no grafo de Dijkstra G
14. **senão**:
15. não adicione aresta de retorno ou avanço com origem em v
16. // os vértices $v-1, v, v+1$ corresponderão a // um bloco *if*...*then* no grafo de Dijkstra G
17. **senão**:
18. adicione uma aresta $[v-k \leftarrow v]$ escolhida aleatória e uniformemente dentre as arestas de retorno admissíveis com origem em v
19. // os vértices $v-k, v-k+1, \ldots, v$ corresponderão a // um bloco *repeat* no grafo de Dijkstra G
20. **senão**:
21. **se** $B[i] =$ '1':
22. faça $V(G) = V(G) \cup \{t+1\}$, onde t é o rótulo do maior vértice atualmente em $V(G)$
23. acrescente a aresta de caminho $[t \to t+1]$
24. acrescente a aresta de avanço $[v \to v+2]$
25. // o vértice $v+2$ será um vértice mudo
26. faça $i = i+1$, e faça $v = v+1$
27. **retorne** G

(iii) não existe em $E(G)$ qualquer aresta de retorno $[w' \leftarrow w]$, isto é, com origem em w;

(iv) não existe em $E(G)$ a aresta de avanço $[w-1 \to w+1]$;

(v) toda aresta de retorno $[z' \leftarrow z]$ com $w < z < v$ (isto é, com origem entre w e v ao longo de H) satisfaz $z' > w$ (isto é, a aresta chega em um vértice posterior a w ao longo de H).

© 2017 Sociedade Brasileira de Computação

Com exceção dos vértices mudos, que são origens apenas de arestas de caminho, cada vértice $v > 1$ poderá ser o vértice de origem de até uma aresta de retorno ou de avanço. Mais precisamente:

- para cada bit '1' de índice $i \in \{2, \ldots, n\}$ na representação binária B da chave, teremos uma aresta de retorno indo do i-ésimo vértice não-mudo v_i (na sequência dada por H) até um vértice w tal que a diferença entre os rótulos de v_i e w seja *ímpar*, ou — quando não houver aresta de retorno admissível satisfazendo a paridade desejada da diferença dos rótulos — uma aresta de avanço $[v \rightarrow v + 2]$;
- para cada bit '0' de índice $i \in \{2, \ldots, n\}$ em B, teremos analogamente uma aresta de retorno $[w \leftarrow v_i]$ tal que $v_i - w$ é *par*, ou — quando não houver aresta de retorno admissível satisfazendo a paridade desejada da diferença dos rótulos — não acrescentaremos aresta com origem no i-ésimo vértice não-mudo v_i.

Uma aresta de retorno que não satisfizesse as condições acima daria origem, invariavelmente, a um grafo não pertencente à classe dos grafos de Dijkstra, e que, portanto, não poderia aparecer como GFC de código estruturado. Por construção, as arestas de retorno adicionadas a G pelo Algoritmo 1 são sempre admissíveis, e produzem, de acordo com o caso, blocos básicos associados aos construtos *while* e *repeat* dos grafos de Dijkstra (linhas 9–12 e 18–19).

Note que a adição de uma aresta de avanço $[v_i \rightarrow v_i + 2]$ com origem no i-ésimo vértice não-mudo do caminho hamiltoniano, em qualquer das iterações do Algoritmo 1, é sempre seguida, já na iteração seguinte, de ações que levam à geração de um bloco *while* ou de um bloco *if*...*then*, que são *statement graphs* constituintes dos grafos de Dijkstra [Bento et al. 2015]. O bloco exato a ser gerado dependerá do valor do $(i + 1)$-ésimo bit de B (linhas 9–12 e 15–16). Pela mesma razão, note também que a adição da aresta de avanço $[v \rightarrow v+2]$ implica sempre a remoção da aresta de caminho $[v \rightarrow v+1]$, rompendo o caminho hamiltoniano original.

2.2. Decodificação

O algoritmo de decodificação consiste de quatro etapas. Primeiro, os vértices da marca d'água são rotulados ao longo do caminho hamiltoniano original em ordem crescente: $1, 2, \ldots, |V(G)|$. Tal rotulação sempre pode ser realizada, já que os blocos do caminho hamiltoniano original (ainda que este tenha sido rompido ao longo da codificação) são consecutivos no GFC. Em seguida, definimos o primeiro bit do binário como '1' (o que é sempre verdade, por construção, uma vez que jamais codificamos zeros à esquerda). Passamos então a obter as informações codificadas pelas arestas de retorno e avanço (ou por sua ausência) com origem em cada um dos vértices de rótulo $v \geq 2$:

- se v é destino de uma aresta de avanço (vértices mudo), ignore-o; senão,
- uma aresta de retorno $[w \leftarrow v]$ tal que $v - w$ é ímpar (respectivamente, par) indica que o bit codificado em v é '1' (respectivamente, '0');
- uma aresta de avanço $[v \leftarrow v + 2]$ indica que o bit codificado por v é '1';
- e, finalmente, a não existência de aresta de retorno ou avanço com origem em v indica que o bit codificado em v é '0'.

Ao final, o algoritmo percorre os bits de B calculando o valor inteiro da chave ω da forma usual. O Algoritmo 2 apresenta em pseudo-código o processo de decodificação.

 © 2017 Sociedade Brasileira de Computação

Algoritmo 2: Decodificação de marca d'água estruturada

Entrada: uma marca d'água G
Saída: a chave ω codificada por G

1. rotule os vértices de G em ordem ascendente, partir de 1, ao longo do único caminho hamiltoniano de G
2. crie um vetor B, inicialmente com uma única posição, preenchida com '1'
3. seja $i = 2$ (o *índice corrente*)
4. **para cada** vértice $v \in \{2, |V(G)| - 2\}$:
5. **se** v não for mudo (destino de uma aresta de avanço):
6. **se** v é origem de uma aresta de avanço:
7. $B[i] =$ '1'
8. **senão se** existe aresta de retorno $[w \leftarrow v]$ com $v - w$ ímpar:
9. $B[i] =$ '1'
10. **senão**:
11. $B[i] =$ '0'
12. faça $i = i + 1$
13. seja $n = i - 1$ (o índice do último bit gerado)
14. **retorne** $\omega = \sum_{i=1}^{n} B[i] \cdot 2^{n-i}$

3. Um exemplo completo

A Figura 1 ilustra duas marcas d'água geradas pelo Algoritmo 1 para a chave $\omega = 397$, cuja representação binária, determinada pelo passo 1, é $B = 110001101$, com $n = 9$. A marca d'água apresentada na Figura 1(a) possui $n + 3$ vértices, enquanto a marca d'água da Figura 1(b) possui $n + 4$ vértices. O vértice 2 corresponde ao bit da posição 2 em B, que é um '1', e assim o algoritmo produz a aresta de retorno $[1 \leftarrow 2]$ (um salto ímpar) nas duas execuções. Para o vértice 3, que corresponde ao bit '0' na posição 3 de B, o algoritmo não dispunha de arestas admissíveis (a aresta $[1 \leftarrow 3]$, embora apresente o desejado salto par, viola a segunda condição de admissibilidade). Dessa forma, nenhuma aresta adicional a partir de 3 é acrescentada. O mesmo acontece com o vértice 4. Para o vértice 5, correspondente ao bit '0' na posição 5 de B, o algoritmo dispunha de duas arestas de retorno admissíveis: $[1 \leftarrow 5]$ e $[3 \leftarrow 5]$. A aresta de $[3 \leftarrow 5]$ foi a escolhida na execução da Figura 1(a), e $[1 \leftarrow 5]$ na da Figura 1(b).

Note que, até este ponto, o algoritmo não acrescentou nenhuma aresta de avanço, mantendo os rótulos dos vértices equivalentes aos índices dos bits de B. Para o vértice 6, correspondente ao bit '1' na posição 6 de B, para a marca d'água da Figura 1(a) o algoritmo acrescentou a aresta de retorno $[1 \leftarrow 6]$; no entanto, na execução que levou à marca d'água da Figura 1(b) o algoritmo não dispunha de arestas de retorno admissíveis, e foi portanto levado à adicionar a aresta de avanço $[6 \rightarrow 8]$. Dessa forma, o vértice 8 tornou-se "mudo", implicando a inclusão de um vértice extra e desemparelhando os rótulos dos vértices da marca d'água da Figura 1(b) e os índices dos bits de B.

Dando sequência ao exemplo de codificação, para o vértice 7, que corresponde ao bit '1' na posição 7 de B, o algoritmo emprega as únicas opções disponíveis em cada caso, que são a aresta de avanço $[7 \rightarrow 9]$ na primeira marca d'água, deixando o vértice 9

© 2017 Sociedade Brasileira de Computação

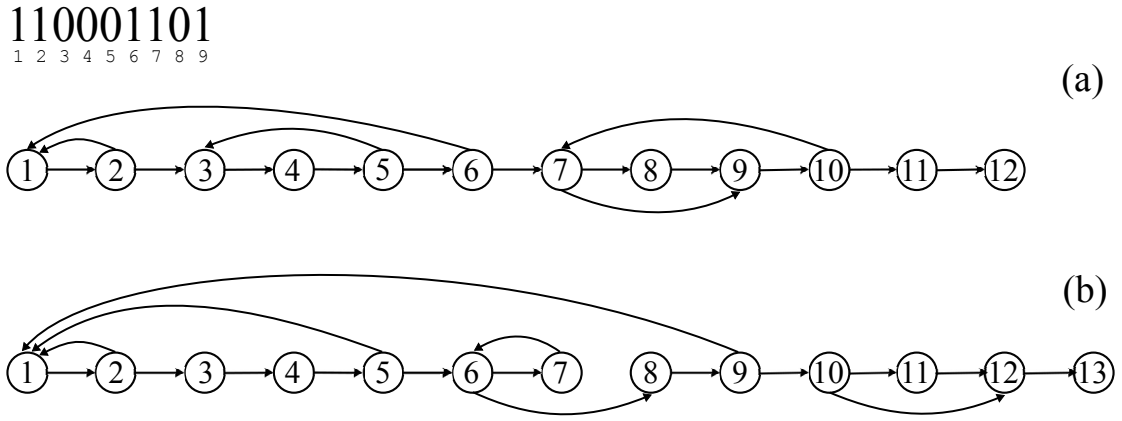


Figura 1. Marcas d'água distintas codificando a chave $\omega = 397$

"mudo" e causando a inclusão de novo vértice, e a aresta de retorno $[6 \leftarrow 7]$, na segunda.

Na Figura 1(b), para que os vértices $\{6, 7, 8\}$ correspondessem a um bloco *while* dos grafos de Dijkstra, a aresta de caminho $[7 \rightarrow 8]$ precisou ser removida.

Na Figura 1(a), o vértice 8 representa o bit '0' da posição 8 de B e, portanto, nenhuma aresta adicional, uma vez que não havia arestas de retorno admissíveis. Por outro lado, na Figura 1(b), o vértice 8 é vertice mudo, não codificando bit. Já o vértice 9, na Figura 1(a), é mudo, enquanto na Figura 1(b) está associado à codificação do bit '0' na posição 8 de B. Para tal, o algoritmo adicionou a aresta de retorno $[1 \leftarrow 9]$.

Dando sequência, o vértice 10 corresponde ao bit '1' na posição 9 de B tanto na Figura 1(a) quanto na Figura 1(b). Para a marca d'água da Figura 1(a), o algoritmo escolheu criar a aresta de retorno $[7 \leftarrow 10]$, dentre as diversas opções admissíveis para a paridade desejada do salto (ímpar); para a marca d'água da Figura 1(b), criou a aresta de avanço $[10 \rightarrow 12]$, dado que não havia arestas de retorno admissíveis.

A decodificação das duas marcas d'água é facilmente verificável.

4. Análise

É fácil ver que a quantidade de arestas de avanço adicionadas ao longo do algoritmo é, no máximo, $\lfloor n/3 \rfloor$. De fato, quando o bit de índice $i \geq 3$ ocasiona a inclusão de uma aresta de avanço (com origem no i-ésimo vértice não-mudo de H), os bits de índices $i+1$ e $i+2$ jamais ocasionarão a inclusão de arestas de avanço, pois as arestas de retorno $[v_{i+1} - 1 \leftarrow v_{i+1}]$ e $[v_{i+2} - 3 \leftarrow v_{i+2}]$ serão necessariamente admissíveis. Dessa forma, a quantidade de vértices do grafo marca d'água satisfaz

$$|V(G)| \leq (n+2) + \lfloor n/3 \rfloor = \lfloor 4n/3 \rfloor + 2.$$

O número máximo de iterações do laço principal do Algoritmo 1 é, portanto, $O(n)$, uma vez que temos uma iteração por vértice — a variável v é incrementada a cada iteração.

Para mostrarmos que o Algoritmo 1 pode ser implementado para rodar em tempo $O(n)$, é suficiente garantir que suas linhas 8 e 18 são executadas em tempo $O(1)$, pois todas as demais linhas rodam em tempo constante de forma trivial (representando o grafo G por meio de usuais listas de adjacências). Ou seja, precisamos garantir que se possa

 © 2017 Sociedade Brasileira de Computação

escolher em tempo constante uma aresta de retorno admissível com origem no vértice corrente v. Para isso, manteremos um registro dos vértices que são a cada momento *candidatos* a ser o destino de uma aresta de retorno (admissível) com origem no vértice corrente v. Ora, tal conjunto de candidatos é constituído por todo vértice $w < v$ que satisfaça as cinco condiçoes de admissibilidade vistas na Seção 2.1, e além disso possua a paridade desejada, isto é, a mesma paridade de v, caso o bit corrente, de índice i, seja um '0', ou paridade distinta da de v, caso o bit corrente seja um '1'.

A condição (i) pode ser verificada trivialmente antes mesmo do sorteio do destino admissível w, forçando-se a escolha $w = v-1$ se existir a aresta de avanço $[v-1 \rightarrow v+1]$. O mesmo é feito com relação à condição (ii), removendo-se o vértice w do conjunto de candidatos caso exista a aresta de retorno $[w \leftarrow v-1]$.

Para atender às demais condições, o que fazemos é manter duas pilhas de candidatos, uma que conterá apenas vértices de rótulo par (inicialmente vazia), e outra que conterá apenas vértices de rótulo ímpar (inicialmente contendo apenas o vértice 1). Fazemos também com que, ao final de cada iteração, o vértice corrente $v \geq 2$ seja acrescentado à pilha correspondente caso v não tenha se tornado a origem de uma aresta de retorno, satisfazendo assim a condição (iii), e não haja aresta de avanço $[v-1 \rightarrow v+1]$, respeitando-se assim também 'a condição (iv). Até aqui, sempre em tempo constante. Finalmente, ao acrescentarmos uma aresta de retorno $[w' \leftarrow v]$, removemos da pilha das pilhas de candidatos todo vértice w tal que $w > w'$, garantindo assim que a condição (v) seja atendida. Note que remover os vértices de rótulo maior do w' da pilha que contém o próprio w' é uma tarefa trivial, uma vez que o índice de w' é conhecido. De fato, tal índice acabou de ser escolhido aleatoriamente dentre os índices da pilha correspondente à paridade desejada, apontando w'. Como os candidatos estão empilhados, por construção, em ordem crescente de rótulo, basta apontarmos o fim da pilha para o índice de w', e a remoção está consumada em tempo $O(1)$. Já a remoção em tempo constante dos vértices $w > w'$ da *outra* pilha requer o uso de uma estrutura auxiliar que registra o tamanho de cada pilha no momento em que cada vértice é acrescentado a uma das pilhas[1], como recomendado em [Bento et al. 2014]. Com esse mapa auxiliar (implementado facilmente sobre um vetor com endereçamento direto, identificando-se rótulo de vértice com índice no vetor), podemos recuperar o tamanho que tinha a pilha que não contém w' no momento em que w' foi inserido em sua pilha, isto é, o maior índice contendo um vértice de rótulo menor do que w'. De posse desse índice, basta apontarmos o final da pilha para ele, e teremos assim removido — por meio de uma única operação de tempo constante — todos os vértices com rótulo maior do w'.

Para sortearmos o vértice w dentre os candidatos a destino de um aresta de retorno com origem no vértice corrente v, basta sortearmos um número inteiro aleatório e uniforme entre 1 e o tamanho da pilha de candidatos com a paridade desejada (definida pelo valor do bit corrente) — o que é feito em tempo constante.

O Algoritmo 2 roda claramente em tempo linear na quantidade de vértices da marca d'água, portanto $\Theta(n)$.

[1]Na verdade, é suficiente registrar o tamanho da pilha que *não está recebendo* v, no momento em que v é acrescentado a uma das pilhas — uma vez que a remoção dos vértices da própria pilha que contem v será feita facilmente, como explicado.

© 2017 Sociedade Brasileira de Computação

5. Conclusão

Apresentamos um *codec* que produz marcas d'água que podem aparecer como subgrafos do GFC por meio apenas da adição de código fictício estruturado. Essa propriedade aumenta a furtividade das marcas d'água produzidas, uma vez que o GFC obtido passa se assemelhar a um GFC típico, correspondente a programa estruturado.

Além disso, diferentemente do que foi mostrado para o esquema de marca d'água de Chroni e Nikolopoulos (capaz de se recuperar de ataques de distorção que causem a remoção de, no máximo, 2 arestas [Bento et al. 2016]), o *codec* proposto pode se recuperar de ataques de distorção que produzam um número arbitrário e predefinido b de arestas removidas, por possibilitar o uso de códigos de correção de erro bit-a-bit.

É também digno de nota que o *codec* proposto, ao empregar randomização, ganha pontos no quesito diversidade, pois apresenta diversas codificações possíveis para uma mesma chave.

Os algoritmos de codificação e decodificação propostos foram implementados em linguagem Python 3, e testados com sucesso para todas as chaves inteiras entre 1 e 10^7. Após a codificação da chave, cada marca d'água obtida foi submetida ao algoritmo de decodificação, e a chave decodificada comparada com a chave original. A Figura 2 ilustra um exemplo de execução.

```
========================================

Options:
  an integer positive key --> for encoding/decoding in verbose mode;
  min_key, max_key (comma-separated) --> for testing over a range;
  <enter> --> quit.

Please type the key: 12345
B = 11000000111001 (n = 14)

Watermark:

       1<<            3<<       1<<       8<<        10<<         1<<        14<<
    1--->2--->3--->4--->5--->6--->7--->8--->9--->10--->11    12--->13--->14--->15--->16--->17
                                              >>12
Testing decoding procedure...
Retrieved key: 12345
1 key(s) encoded/decoded successfully.

========================================
```

Figura 2. Exemplo de saída obtida durante os testes do *codec*

O código-fonte completo (em linguagem Python 3) pode ser encontrado em: `https://www.dropbox.com/s/iqv6o5asfl13wlq/watermark.py`.

Um refinamento possível, que em nada altera as características ou a complexidade do *codec*, é o estabelecimento de um grau de entrada máximo para os vértices. Isso tende a evitar um acúmulo de arestas de retorno com destino nos vértices de rótulo mais baixo (mais à esquerda, no caminho hamiltoniano original), e que participam portanto, como candidatos, de um número maior de sorteios. Isso evitaria, por conseguinte, o aparecimento de uma quantidade excessiva de laços aninhados (com início no mesmo vértice do GFC). Alternativamente, seria também possível escolher a aresta de retorno, dentre as admissíveis, com probabilidade não uniforme, favorecendo os vértices de maior

 © 2017 Sociedade Brasileira de Computação

rótulo. A primeira dessas soluções foi de fato implementada em nosso código, onde o grau de entrado máximo do grafo é escolhido aleatória e uniformemente entre 1 e $\sqrt{n}$, adicionando dessa forma um grau extra de aleatoriedade.

Referências

Arboit, G. (2002). A method for watermarking java programs via opaque predicates. In *In Proc. Int. Conf. Electronic Commerce Research (ICECR-5)*.

Bento, L. M., Boccardo, D. R., Machado, R. C., de Sá, V. G. P., and Szwarcfiter, J. L. (2016). On the resilience of canonical reducible permutation graphs. *Discrete Applied Mathematics (aceito para publicação)*.

Bento, L. M. S., Boccardo, D., Machado, R. C. S., Pereira de Sá, V. G., and Szwarcfiter, J. L. (2013). Towards a provably resilient scheme for graph-based watermarking. In Brandstädt, A., Jansen, K., and Reischuk, R., editors, *Graph-Theoretic Concepts in Computer Science: 39th International Workshop, WG 2013, Lübeck, Germany, June 19-21, 2013, Revised Papers*. Springer Berlin Heidelberg.

Bento, L. M. S., Boccardo, D. R., Machado, R., de Sá, V. G. P., and Szwarcfiter, J. L. (2014). A randomized graph-based scheme for software watermarking. In *XIV Simpósio Brasileiro de Seguraça Informação e de Sistemas Computacionais (SBSEG'14). SBC, Belo Horizonte, Brazil*, pages 30–41.

Bento, L. M. S., Boccardo, D. R., Machado, R., de Sá, V. G. P., and Szwarcfiter, J. L. (2015). The graphs of structured programming. In *13th Cologne Twente Workshop on Graphs and Combinatorial Optimization, Istanbul, Turkey, May 26-28, 2015.*, pages 105–108.

Chroni, M. and Nikolopoulos, S. D. (2012a). An efficient graph codec system for software watermarking. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops*, pages 595–600.

Chroni, M. and Nikolopoulos, S. D. (2012b). An embedding graph-based model for software watermarking. In *2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 261–264.

Collberg, C., Carter, E., Debray, S., Huntwork, A., Kececioglu, J., Linn, C., and Stepp, M. (2004). Dynamic path-based software watermarking. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, PLDI '04, pages 107–118, New York, NY, USA. ACM.

Collberg, C., Kobourov, S., Carter, E., and Thomborson, C. (2003). Error-correcting graphs for software watermarking. *Lecture Notes in Computer Science*, 2880:156–167.

Collberg, C. and Nagra, J. (2009). *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional, 1st edition.

Collberg, C. and Thomborson, C. (1999). Software watermarking: Models and dynamic embeddings. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 311–324, New York, NY, USA. ACM.

© 2017 Sociedade Brasileira de Computação

Cousot, P. and Cousot, R. (2004). An abstract interpretation-based framework for software watermarking. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04, pages 173–185, New York, NY, USA. ACM.

Dahl, O. J., Dijkstra, E. W., and Hoare, C. A. R., editors (1972). *Structured Programming*. Academic Press Ltd., London, UK, UK.

Davidson, R. and Myhrvold, N. (1996). Method and system for generating and auditing a signature for a computer program. US Patent 5,559,884.

Dijkstra, E. W. (1968). Go-to statement considered harmful. *Comm. ACM*, 11:174–186.

Guruswami, V. (2005). *List Decoding of Error-Correcting Codes: Winning Thesis of the 2002 ACM Doctoral Dissertation Competition (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Mann, H. B. (1968). *Error correcting codes; proceedings of a symposium. Edited by Henry B. Mann*. Wiley New York.

Monden, A., Iida, H., Matsumoto, K., Torii, K., and Inoue, K. (2000). A practical method for watermarking java programs. In *24th International Computer Software and Applications Conference (COMPSAC 2000), 25-28 October 2000, Taipei, Taiwan*, pages 191–197.

Nagra, J. and Thomborson, C. D. (2004). Threading software watermarks. In *Information Hiding, 6th International Workshop, IH 2004, Toronto, Canada, May 23-25, 2004, Revised Selected Papers*, pages 208–223.

Purser, M. (1995). *Introduction to error-correcting codes*. Artech House.

Qu, G. and Potkonjak, M. (1998). Analysis of watermarking techniques for graph coloring problem. In *Proceedings of the 1998 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '98, pages 190–193, New York, NY, USA. ACM.

Reed, I. S. and Solomon, G. (1960). Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304.

Venkatesan, R., Vazirani, V., and Sinha, S. (2001). *A Graph Theoretic Approach to Software Watermarking*, pages 157–168. Springer Berlin Heidelberg, Berlin, Heidelberg.

Zeng, Y., Liu, F., Luo, X., and Lian, S. (2012). Abstract interpretation-based semantic framework for software birthmark. *Comput. Secur.*, 31(4):377–390.

 © 2017 Sociedade Brasileira de Computação

Um Sistema Adaptativo de Detecção e Reação a Ameaças*

Antonio Gonzalez Pastana Lobato[1], Martin Andreoni Lopez[1,2], Gabriel Antonio F. Rebello[1], e Otto Carlos Muniz Bandeira Duarte[1]

[1]GTA / PEE-COPPE / UFRJ – Brasil
[2]LIP6/CNRS (UPMC Sorbonne Universités) – França

Abstract. *Attackers create new threats and constantly change their behavior to mislead current security systems. Moreover, threats are detected in days or weeks, while a countermeasure must be immediately triggered to avoid or reduce any damages. This paper proposes an adaptive threat detection system with a SDN based schema to perform countermeasures. The contributions of this work are the following: i) threat detection and prevention analyzing only five packets of each flow; ii) adaptive detection algorithms with real time training; iii) instant countermeasure trigger, without waiting the end of the flow; iv) effective threat block, even in scenarios with spoofed IP addresses. A SDN schema effectively monitors the five packets sequence and blocks the threat in its source, avoiding the waste of resources. The results show a high accuracy in threat detection, even with network behavior varying over time.*

Resumo. *Atacantes criam novas ameaças e constantemente mudam seu comportamento para enganar os sistemas de segurança atuais. Aliás, as ameaças são detectadas em dias ou semanas, enquanto uma contramedida deve ser imediatamente efetuada para evitar ou reduzir prejuízos. Este artigo propõe um sistema adaptativo de detecção de ameaças que possui um esquema baseado em Redes Definidas por Software (SDN) para realizar contramedidas. As contribuições do trabalho são: i) a detecção e prevenção de ameaças analisando uma sequência de apenas cinco pacotes de cada fluxo; ii) o desenvolvimento de algoritmos de detecção treinados em tempo real, com comportamento adaptativo; iii) o imediato acionamento de contramedidas sem esperar o fim do fluxo; e iv) o efetivo bloqueio de ameaças mesmo em cenários nos quais o endereço do pacote IP é mascarado. Um esquema baseado na tecnologia SDN efetua o monitoramento da sequência de cinco pacotes e o rápido bloqueio do ataque ainda na origem, evitando que recursos de rede sejam desperdiçados. Os resultados obtidos mostram uma alta acurácia na detecção de ameaças, mesmo com o comportamento da rede variando com o tempo.*

1. Introdução

A grande quantidade de dados transferidos pelos sistemas de comunicação atuais criam um cenário desafiador para a detecção de intrusão [Suthaharan 2014]. A prevenção consiste em uma ação que neutraliza ou mitiga o ataque e impede que maiores prejuízos aconteçam. Os atuais sistemas de segurança, como o *Security Information and Event Management* (SIEM), não possuem um desempenho satisfatório, pois cerca de 85% das intrusões de redes são detectadas semanas depois de terem ocorrido [Ponemon e IBM 2017]. É essencial que o tempo de detecção seja o mínimo possível para que a prevenção da intrusão possa ser efetiva [Wu et al. 2014].

*Este trabalho foi realizado com recursos do INCT INTERNET DO FUTURO, do CNPQ, da CAPES, e da FAPERJ

 © 2017 Sociedade Brasileira de Computação

Além disso, os ataques de segurança vêm se aperfeiçoando e a simples análise e filtragem de pacotes pelo cabeçalho IP e porta TCP deixaram de ser efetivas. O tráfego atacante procura se esconder das ferramentas de segurança falsificando o IP de origem e alterando dinamicamente a porta TCP. Nesse contexto, uma alternativa promissora para classificar tráfego e detectar ameaças, que vem apresentando bons resultados, se baseia no uso de técnicas de aprendizado de máquina.

A prevenção de intrusão deve conseguir bloquear inteiramente as ameaças evitando assim que o tráfego malicioso afete o desempenho da rede. Uma abordagem para realizar esse bloqueio de tráfego é colocar o sistema de prevenção de intrusão diretamente na rota dos pacotes que ele analisa, de maneira que ele possa atuar sobre o tráfego malicioso. Contudo, essa forma apresenta problemas como: a introdução de latência, já que o sistema de detecção analisa o tráfego antes de encaminhar para o destino; e a baixa acurácia, visto que o sistema posicionado dessa maneira não tem acesso aos demais tráfegos da rede e pode classificar errado um ataque sem essas informações. Este trabalho usa uma abordagem alternativa, que não apresenta esses problemas, a detecção fora da rota, na qual os pacotes são espelhados para um elemento sensor. Uma forma de implementar esquemas de prevenção de intrusão fora da rota é através da tecnologia de redes definidas por *software* (*Software Defined Network* - SDN) [McKeown et al. 2008], que fornece um controle logicamente centralizado do tráfego de rede.

Este artigo propõe métodos adaptativos de detecção de ameaças e um esquema para a extração das características dos fluxos com o imediato bloqueio do tráfego malicioso. A abordagem de caracterização dos fluxos é baseada na análise periódica de um reduzido número de pacotes em sequência. O esquema provê a detecção automática de ameaças baseada na análise dessas características por algoritmos adaptativos de aprendizado de máquina. A proposta apresenta as seguintes vantagens: robusta para cenários de ataques por inundação, já que apenas os primeiros pacotes são analisados por período; precisa na identificação do tráfego malicioso, pois possui uma acurácia elevada na classificação de ataques e consegue se adaptar a novas ameaças; e eficaz no bloqueio dos ataques, pois com o uso das redes definidas por *software* é possível bloquear o tráfego atacante na sua origem, mesmo em cenários onde os endereços IP são mascarados.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A Seção 3 apresenta a elaboração do conjunto de dados para avaliar o esquema de detecção apenas com os primeiros pacotes de cada fluxo. Os métodos de detecção adaptativos propostos são apresentados na Seção 4, a Seção 5 detalha o esquema para o desvio de tráfego e bloqueio de ameaças usando redes definidas por *software*. Por fim, a Seção 6 conclui o trabalho.

2. Trabalhos Relacionados

As técnicas de aprendizado de máquina podem ser supervisionadas ou não supervisionadas, dependendo se o conjunto de dados está rotulado ou não. Entre as técnicas de classificação mais conhecidas estão as redes neurais, as árvores de decisão e as máquinas de vetores de suporte (*Support Vector Machines* - SVM) [Buczak e Guven 2015]. Li *et al.* combinam uma técnica de correspondência de padrões, *Dynamic Time Warping* - DTW) e SVM para gerar regras de detecção de intrusão [Ji et al. 2016]. O método é avaliado através do conjunto de dados KDD tradicional para classificar os ataques de negação de serviço (*Denial of Service* - DoS), varredura de porta (*Probe*) e acesso remoto para Local (*Remote to Local* - R2L). No entanto, seu método só pode lidar com ataques conhecidos. Na análise não supervisionada, não há informações sobre a classe a que cada amostra pertence. A detecção de padrões aplica esse tipo de análise. Lakhina *et al.* propõem o uso de entropia de amostra para detecção de anomalia e os

 © 2017 Sociedade Brasileira de Computação

autores mostram que esta métrica combinada para IPs e portas de origem e destino, juntamente com análise de volume pode detectar múltiplas fontes de anomalias [Lakhina et al. 2005].

Braga *et al.* propõem um método de detecção de ataques de negação de serviço distribuído (*Distributed Denial of Service* - DDoS) usando um controlador NOX-OpenFlow [Braga et al. 2010]. A proposta extrai as características dos fluxos das tabelas dos comutadores da rede tais como média de pacotes por fluxo, média de *bytes* por fluxo, entre outras. Logo, esses atributos são enviados a um classificador, baseado em um mapa auto organizável dentro do controlador, que determinam se o tráfego é normal ou se corresponde a um ataque. O processamento é feito pelo controlador da rede que é sobrecarregado pelo processo de detecção utilizado. A detecção e a mitigação de ataques de negação de serviço em ambientes SDN é proposto em [Andreoni Lopez et al. 2016]. Na mitigação proposta, o mecanismo de detecção envia uma mensagem segura ao controlador SDN com as informações do fluxo a ser bloqueado. O esquema não detalha como é feita a escolha dos comutadores que recebem as configurações de bloqueio, podendo ser muito drásticas e acabar bloqueando também fluxos lícitos.

Bernaille *et al.* [Bernaille et al. 2006] propõem o uso dos primeiros pacotes de cada fluxo para determinar a aplicação do tráfego analisado. As aplicações possuem um comportamento bem definido no início, por isso, com a análise dos primeiros pacotes é possível classificá-las. A proposta de Bernaille *et al.* para classificar aplicações é fortemente baseada na característica de tamanho de pacote, e os autores esclarecem que torna a proposta vulnerável a atacantes que podem ajustar esse parâmetro para enganar o sistema. Peng *et. al.* também utilizam os *n* primeiros pacotes aplicados a onze algoritmos supervisados de aprendizado de máquina para a identificação de tráfego na Internet [Peng et al. 2015]. Estes trabalhos comprovam a importância do parâmetro tamanho do pacote na acurácia da classificação das aplicações.

Uma plataforma de código aberto para a classificação de tráfego é proposta por Donato *et. al.*. A plataforma denominada *Traffic Identification Engine* - (TIE) [Donato et al. 2014], utiliza seis algoritmos de aprendizado de máquina que obtêm bom desempenho. Na proposta deste artigo a característica de tamanho do pacote é acrescentada a outras 25 características para detectar ameaças. Espera-se que o mesmo sucesso obtido na acurácia da caracterização de aplicações seja atingido na detecção de ameaças. As outras características ajudam a evitar que os atacantes dissimulem o ataque aumentando apenas o tamanho do pacote e praticamente impossibilita ao atacante ajustar de maneira efetiva todos esses parâmetros para se passar por tráfego legítimo.

3. Conjuntos de Dados de Avaliação

Poucos conjuntos de dados estão disponíveis para avaliar mecanismos de defesa e o principal motivo para não fornecer dados de segurança é a privacidade, uma vez que os dados de tráfego podem conter informações confidenciais. Os dois conjuntos de dados mais conhecidos são o DARPA [Lippmann et al. 2000] e o KDD 99 [Lee et al. 1999]. Tráfego TCP/IP, dados de sistema operacional e dados de ameaças coletados de uma rede de computadores simulada compõem o conjunto de dados DARPA. Já o KDD foi elaborado a partir da extração de características dos logs do DARPA. No entanto, esses conjuntos de dados apresentam diversas limitações [Sharafaldin et al. 2017]. Uma delas é que o tráfego não corresponde a um cenário de rede real, já que foi simulado. Outra questão é que esses conjuntos de dados têm mais de 17 anos e não representam ameaças atuais [Sommer e Paxson 2010].

Visando um conjunto de dados com tráfego real mais recente, neste trabalho são avaliados métodos de detecção adaptativo através de dois conjuntos de dados. Um dos conjuntos

© 2017 Sociedade Brasileira de Computação

de dados é composto por um tráfego normal de uma das principais operadoras de rede no Brasil combinado com ataques de *botnets*, e o outro conjunto de dados foi criado no laboratório GTA/UFRJ. Para ambos os conjuntos de dados, os pacotes são combinados em fluxos. Um fluxo é definido como uma sequência de pacotes do mesmo endereço de IP origem para o mesmo endereço de IP destino durante uma janela de tempo. A cada período de tempo, apenas os cinco primeiros pacotes de um fluxo são capturados para a extração de características. O valor de cinco pacotes foi escolhido através da medida de acurácia de três algoritmos de classificação: árvore de decisão, SVM e redes neurais. Esse valor foi a menor quantidade de pacotes que apresentou acurácia superior a 90%. Cada fluxo tem 26 características, gerados pelos dados de cabeçalho TCP/IP. As principais características são: taxa de pacotes TCP, UDP e ICMP; número de portas de origem e de destino; número de cada *flag* TCP; média e variância do tempo de chegada entre pacotes; média e variância do comprimento do pacote de fluxo; entre outros.

O conjunto de dados do operador de rede, doravante denominado NetOp, é composto por informações de acesso real de 373 usuários residenciais de banda larga da cidade do Rio de Janeiro por um período de uma semana. O tráfego de rede é anonimizado por questões de privacidade. Como a operadora filtrou os dados usando um sistema de detecção de intrusão (IDS) e o tráfego depois da filtragem é assumido como tráfego benigno. Foram analisados a quantidade de registros desse IDS e os ataques filtrados correspondem a 15% do tráfego total. Para avaliar os algoritmos de detecção de ameaças foi adicionado o tráfego malicioso real capturado no trabalho de García *et. al* [Garcia et al. 2014]. Os dados de ataque são de uma *botnet* e têm 13 cenários diferentes de infecção por *malware*. Foi então inserido no conjunto de dados o tráfego malicioso de *botnet* correspondendo a 15% de proporção de tráfego de ameaças.

Esse trabalho utiliza um segundo conjunto de dados criado no laboratório com tráfego de rede real para avaliar a proposta. O conjunto de dados foi elaborado através da captura de tráfego contendo tanto o tráfego normal quanto as ameaças de rede reais. O conjunto de dados possui diferentes tipos de ataques. Ao todo, o conjunto de dados contém sete tipos de negação de serviço (*Denial of Service* - DoS) e nove tipos de varredura de portas. Os ataques de DoS são *ICMP flood*, *land*, *nestea*, *smurf*, *SYN flood*, *teardrop*, e *UDP flood*. Os diferentes tipos de varredura no conjunto de dados são *TCP SYN scan*, *TCP connect scan*, *SCTP INIT scan*, *Null scan*, *FIN scan*, *Xmas scan*, *TCP ACK scan*, *TCP Window scan*, e *TCP Maimon scan*. As ferramentas da distribuição *Kali Linux*, que visa testar a segurança de redes de computadores, foram utilizadas para a criação dos ataques. Esses ataques foram rotulados com base no destino, já que os ataques foram enviados para *honeypots*. O conjunto de dados contém cerca de 95 GB de dados de captura de pacotes.

4. Métodos Adaptativos Propostos de Detecção de Ameaças

Os atacantes criam novos ataques e alteram os ataques convencionais de forma a ludibriarem e passarem despercebidos por sistemas de detecção de intrusão baseados em assinatura ou métodos de classificação *off-line*. Para aumentar a robustez da detecção de ameaças, este trabalho propõe um novo esquema adaptativo de coleta de dados capaz de aprender ataques novos e detectar ameaças que se modificam com o tempo. Este esquema de coleta de dados proposto garante adaptabilidade tanto para o comportamento legítimo como para o malicioso. Os métodos se adaptam a mudanças aceitáveis no uso da rede e aprendem novos ataques que são realizados nos *honeypots*. Assim, são propostos três métodos que adaptam seus parâmetros em tempo real à medida que os fluxos de dados chegam. Os dois primeiros algoritmos de detecção são algoritmos de classificação *on-line* que visam classificar os ataques com base na captura de comportamento de ataque conhecida pelo *honeypots*, enquanto o outro é um algoritmo de

 © 2017 Sociedade Brasileira de Computação

detecção de anomalia. Os três algoritmos implementados possuem a capacidade de detectar novos ataques.

Os métodos de classificação *on-line* requerem dados rotulados. Assim, no método de detecção proposto, todos os dados provenientes de *honeypots* são rotulados como ameaças, uma vez que não há nenhum serviço real sendo oferecido nos *honeypots* e todo o acesso é destinado a explorar uma vulnerabilidade intencionalmente instalada. Portanto, todos os fluxos que chegam nos *honeypots* serão usados pelos algoritmos para atualizar os parâmetros. Esta realimentação de dados em tempo real garante comportamento adaptativo para a detecção de ameaças. Então, sempre que um atacante executa um novo ataque ou muda seu comportamento, os modelos de classificação são atualizados. Os métodos de classificação propostos assumem como tráfego benigno os fluxos recebidos nos sensores de tráfego na rede durante o tempo de configuração, que é um período em que o uso da rede é monitorado para garantir que todos os fluxos sejam legítimos. Após esse tempo, caso um fluxo proveniente dos sensores de rede for classificado como ameaça, os parâmetros no algoritmo não são atualizados. Se o fluxo for considerado normal os parâmetros do algoritmo são atualizadosadaptando-se às mudanças de comportamento normal da rede.

O Algoritmo Gradiente Estocástico Descendente com Momento é uma aproximação do algoritmo do gradiente estocástico descendente (*Stochastic Gradient Descent* - SGD), na qual o gradiente é aproximado por uma única amostra. Na aplicação de detecção de ameaças deste artigo são consideradas duas classes: normal e ameaça. Portanto, a função Sigmoid 1

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\intercal x}} \tag{1}$$

é utilizada para executar a regressão logística. Na função Sigmoid, baixos valores de produto dos parâmetros $\theta^\intercal$ vezes o vetor da característica da amostra x retornam zero, enquanto valores altos retornam um. Quando uma nova amostra $x_{(i)}$ chega, o método avalia a função Sigmoid e retorna um para $h_\theta(x_{(i)})$ maiores que 0.5 e zero caso contrário. Esta decisão apresenta um custo associado, com base na classe real da amostra $y_{(i)}$. A função de custo é definida na Equação 2. Esta função é convexa e o objetivo do algoritmo SGD é encontrar o seu mínimo, expresso por

$$J_{(i)}(\theta) = y_{(i)} log(h_\theta(x_{(i)})) + (1 - y_{(i)}) log(1 - h_\theta(x_{(i)})). \tag{2}$$

Quando uma nova amostra chega, o algoritmo dá um passo em direção ao mínimo custo com base no gradiente desta função.

O Algoritmo 1 mostra a implementação do algoritmo do SGD. A cada amostra recebida do vetor $x_{(i)}$ o método determina a classe $y_{(i)}$ baseado no tipo da origem. Se a amostra vier de um *honeypot* a etiqueta é 1, enquanto se vier de uma sonda de trafego a etiqueta é 0. Se a amostra vem de um analisador de tráfego e o SGD prevê como uma ameaça, o algoritmo envia um alerta. Caso contrário, atualiza os parâmetros com base no gradiente da função custo. O termo $\Delta\theta$[1] é o momento e tem o valor da atualização anterior do parâmetro. No contexto do SGD, este termo considera o movimento passado ao atualizar os parâmetros θ. Os parâmetros α e β são periodicamente atualizados de forma *offline* com base no custo histórico de cada amostra.

A Figura 1 mostra o comportamento de acurácia ao longo do tempo para cada amostra

[1]Na física clássica, o momento estimado indica a dificuldade de mudar o movimento de um objeto em movimento circular.

© 2017 Sociedade Brasileira de Computação

Entrada: Características de fluxo de entrada x, Classe y
Saida : Classe Prevista $predict$, Parâmetros do treinamento θ

Inicializar θ, $\Delta\theta$, α, β;
para $i \leftarrow 1$ **até** m **faça**
  $h_\theta(x_{(i)}) = \frac{1}{1+e^{-\theta^\intercal x_{(i)}}}$;
  $predict = round(h_\theta(x_{(i)}))$;
  se $predict == 1$ *and* $y_{(i)} == 0$ **então**
    Envia Alerta;
  senão
    $\theta = \theta - \alpha\nabla J_{(i)}(\theta) + \beta\Delta\theta$;
    $\Delta\theta = \alpha\nabla J_{(i)}(\theta)$;
  fim
fim

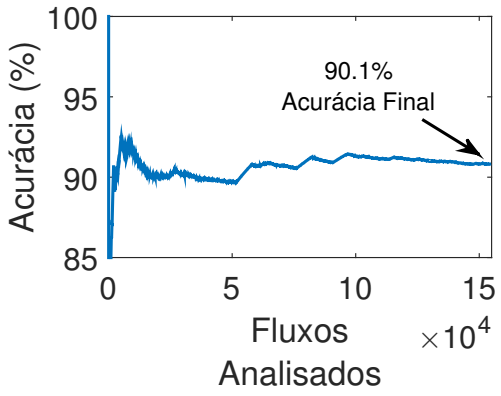
Algoritmo 1: Gradiente Estocástico Descendente com Momento.

Tabela 1: Matriz de Confusão SGD para o conjunto de dados do Laboratório.

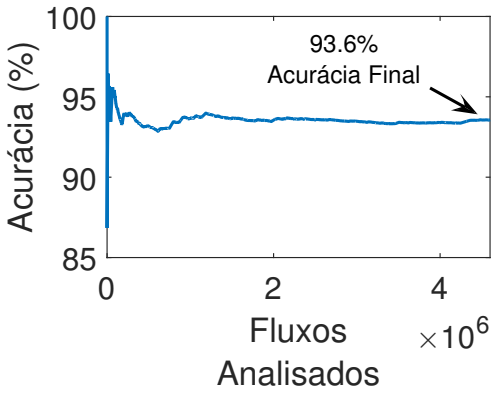
	Normal	**Ameaça**
Normal	104028	2927
Ameaça	11245	35987

Tabela 2: Matriz de Confusão SGD para o conjunto de dados NetOp.

	Normal	**Ameaça**
Normal	4172989	21431
Ameaça	287441	341722



(a) Conjunto de dados do Laboratório.



(b) Conjunto de dados do Operador.

Figura 1: Acurácia do Gradiente Estocástico Descendente. Em ambos os conjuntos de dados, a acurácia permanece estável mesmo com novos ataques e mudanças de comportamento de uso legítimo.

recebida, e as Tabelas 1 e 2 mostram a matriz de confusão final para ambos conjuntos de dados. No início da análise, o *o*verfit ocorre porque o algoritmo tem poucas amostras e se adapta especificamente a elas, resultando em uma precisão muito alta. No entanto, à medida que as amostras chegam, o SGD estabiliza e adquire uma ótima capacidade para generalizar e adaptar a novas amostras de tráfego, terminando com precisão de 90,1% para o conjunto de dados do laboratório e 93,6% para o conjunto de dados NetOp. Apesar das taxas de detecção de ameaças de 76,2% e 54,3%, respectivamente, esse algoritmo possui taxas falso positivas muito baixas de 2,7% e 0,5%. A baixa taxa de falsos positivos é uma característica desejada e um resultado importante da proposta, pois assegura confiança nos alertas de segurança da plataforma.

© 2017 Sociedade Brasileira de Computação

Entrada: Características de fluxo de entrada x, Classe y
Saída : Classe prevista $predict$, parâmetros do treinamento θ

Inicializar θ, α, λ;
para $i \leftarrow 1$ **até** m **faça**
  $predict = sign(\theta^{\intercal} x_{(i)})$;
  se $predict == 1$ *and* $y_{(i)} == -1$ **então**
    Envia Alerta;
  senão
    se $y_{(i)} \theta^{\intercal} x_{(i)} > 1$ **então**
      $\nabla_{(i)} = \theta$;
    senão
      $\nabla_{(i)} = -\lambda y_{(i)} x_{(i)}$;
    fim
    $\theta = \theta - \alpha \nabla_{(i)}$
  fim
fim

Algoritmo 2: Máquina de Vetores de Suporte *Online*.

Tabela 3: Matriz de Confusão SVM para o conjunto de dados do Laboratório.

	Normal	**Ameaça**
Normal	100658	6297
Ameaça	5900	41332

Tabela 4: Matriz de Confusão SVM para o conjunto de dados NetOp.

	Normal	**Ameaça**
Normal	4099766	94654
Ameaça	117051	512112

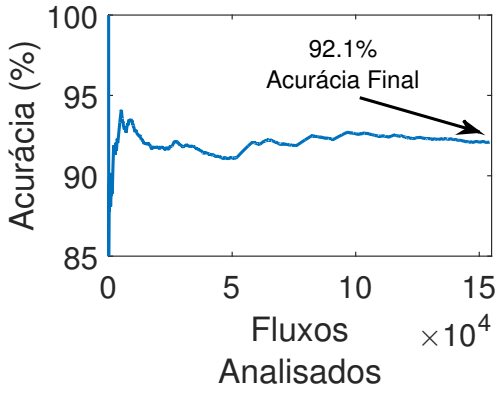
A máquina de vetores de suporte (*Support Vector Machine* - SVM) *online* é um classificador binário com base no conceito de plano de decisão que define os limites das classes. Um hiperplano construído em um espaço multidimensional divide os dados. O algoritmo SVM *online* usa uma aproximação de margem suave com a função convexa de perda de articulação (*hinge loss*), dada por

$$\max\{0, 1 - y\theta^{\intercal} x\}. \tag{3}$$

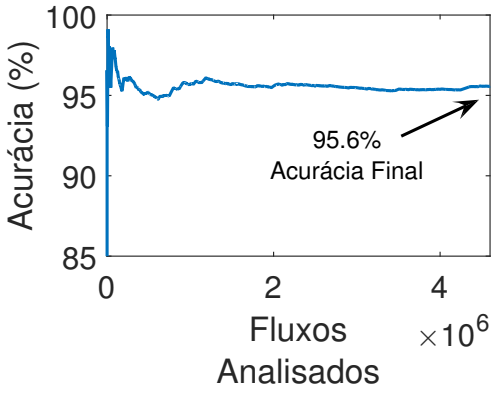
O objetivo deste algoritmo é minimizar a função de perda. Assim como o algoritmo anterior, o método determina a classe $y_{(i)}$ com base na origem. Se vier de um *honeypot*, o rótulo é 1, enquanto que se ele vem de um sensor de rede, o rótulo é -1. Novamente, quando um tráfego de uma sonda é classificado como ameaça, o método envia um alerta e não atualiza o modelo. O Algoritmo 2 mostra a implementação do SVM *online*. Os parâmetros α, λ são periodicamente atualizados com base na avaliação da função de perda sobre as amostras históricas.

A Figura 2 mostra as mudanças da acurácia ao longo do tempo em cada amostra recebida, e as Tabelas 3 e 4 mostram a matriz de confusão final para ambos conjuntos de dados. A acurácia final é melhor do que o algoritmo SGD, porque o SVM é um classificador robusto que maximiza a margem para os limites de decisão do hiperplano. A Figura 2 também mostra que o algoritmo se adapta muito bem às mudanças de uso e novos ataques, mantendo a alta precisão. As acurácias finais são 92,1% para o conjunto de dados do laboratório e 95.6% para o NetOp. As matrizes de confusão mostram que a taxa de detecção de ataque é maior do que a SGD, com valores de 87,5% e 81,4%, respectivamente. Além disso, o SVM também obtém uma boa taxa de falso positivo, com valores de 5,9% e 2,3%.

© 2017 Sociedade Brasileira de Computação



(a) Conjunto de dados do Laboratório.



(b) Conjunto de dados do Operador.

Figura 2: Máquinas de vetores de suporte *online*. Novamente, mesmo com o comportamento mudando, a acurácia permanece estável.

A detecção de anomalia é o terceiro esquema de detecção proposto, que tem a capacidade de descobrir ataques de dia zero que são difíceis de se detectar, já que ainda não existem dados sobre ele. Os parâmetros são atualizados durante o tempo de configuração com base apenas nos dados capturados pelos sensores de tráfego. Após esse período, se uma amostra for considerada normal, os parâmetros serão atualizados. Por outro lado, se for considerada uma ameaça, um alerta é emitido e os parâmetros de algoritmos não são atualizados. Para avaliar os algoritmos, utilizamos 70% dos dados normais para ambos os conjuntos de dados no tempo de configuração e os outros 30% para avaliar os falsos positivos. A taxa de detecção de ataque é obtida com base em todas as ameaças no conjunto de dados.

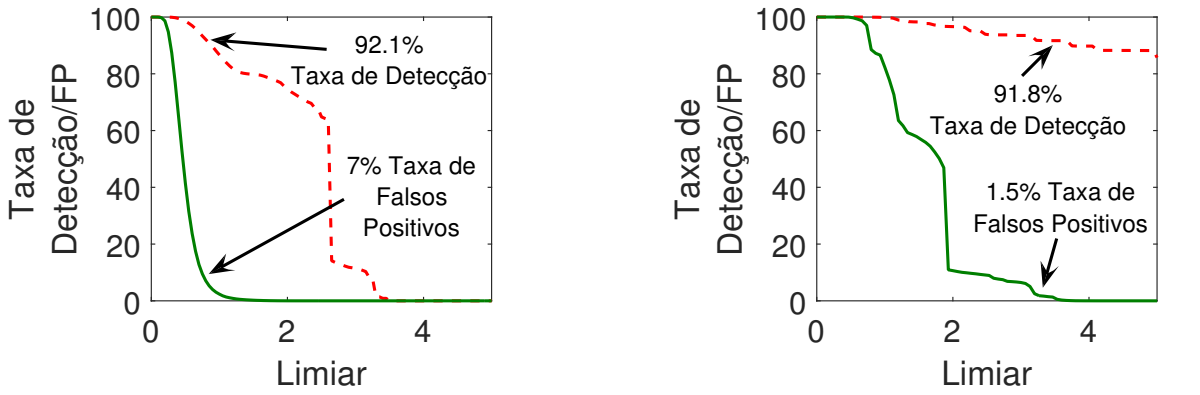
A identificação de uma anomalia é obtida pela análise do valor de entropia de uma janela deslizante de fluxos. A entropia de amostra, expressa por

$$H(X) = -\sum_{i=1}^{N}(\frac{n_i}{S})\log_2(\frac{n_i}{S}), \tag{4}$$

indica o grau de concentração ou dispersão de uma característica, onde S é o número total de observações, n_i é o número de observações dentro do intervalo i de valores e N é o número de intervalos. Quando todos os valores estão concentrados em um intervalo, $H(X)$ é igual a zero e quando cada valor está em uma faixa diferente i, o valor de $H(X)$ é $\log_2(N)$. Então, dada uma série de observações X, a entropia de amostra resume o nível de concentração em um único valor. Foi definida uma janela deslizante de 40 fluxos e calculado o valor de $H(X)$ para cada uma dessas janelas, gerando as séries temporais. Outro parâmetro determinado na fase de configuração é o intervalo contendo a maioria das amostras. Estes parâmetros são atualizados conforme novas amostras chegam. Os valores normais de entropia do tráfego tendem a ser concentrados e o valor mais frequente tende a estar no centro. Assim, o esquema de detecção de ameaça por anomalia proposto define um limiar que determina a distância aceita da entropia $H(X)$ para o intervalo mais frequente.

A Figura 3 mostra os resultados para diferentes valores de limiar. Para valores de limiar pequenos, a taxa de detecção é muito alta, resultando também em uma alta taxa de falsos positivos. Para valores baixos, ocorre o oposto, uma baixa taxa de falsos positivos ao custo de uma baixa taxa de detecção. No entanto, existem valores de limiar que apresentam um *trade-off* excelente entre essas taxas. Para o conjunto de dados do laboratório, o limite 1,3 resulta em

 © 2017 Sociedade Brasileira de Computação



(a) Conjunto de dados do Laboratório. **(b) Conjunto de dados do Operador de Rede.**

Figura 3: Série Temporal de Entropia.

92,1% de taxa de detecção de ataque com 7% de taxa de Falso Positivo, enquanto que para o conjunto de dados NetOp, o limite de 3,4 obtém uma taxa muito baixa de Falsos positivos, de 1,5%, com uma taxa de detecção de ataque alta, de 91,8%.

5. Desenvolvimento e Avaliação de um Protótipo do Sistema Proposto

No protótipo desenvolvido a detecção de intrusão é realizada fora da rota através do espelhamento do tráfego real para não gerar latência no tráfego de produção e para poder reunir informações de outras rotas e correlacionar dados importantes para a detecção correta dos ataques. Com isso, a prevenção do tráfego malicioso é feita de maneira eficiente utilizando a programabilidade das redes definidas por *software* (*Software Defined Networks* - SDN) para realizar o espelhamento de apenas alguns pacotes de um fluxo por período.

A detecção dos ataques é feita com base na análise periódica dos primeiros pacotes de cada fluxo. Para fazer isso, sempre que um fluxo indefinido chega a um comutador SDN, ele o encaminha ao controlador, que então instala uma regra com duas ações, a primeira encaminhando o fluxo para o seu destino e a segunda replicando o tráfego e o encaminhando para a máquina de análise. O início de cada fluxo sempre é encaminhado à máquina de análise, já que fluxos novos não estão definidos. Isso é importante, pois aplicações possuem um comportamento bem definido no início. Por sua vez, a máquina de análise mantém um controle do número de pacotes de cada fluxo que ela está analisando e quando esse número chega a cinco pacotes ela envia as informações do fluxo para o sistema de detecção de intrusão e também uma mensagem avisando ao controlador que a análise dos pacotes daquele fluxo específico foi concluída. Um fluxo é definido como todos os pacotes de um mesmo IP origem e mesmo IP destino e a máquina analisadora extrai as 26 características do fluxo. Após o término da análise de fluxo, o controlador recebe esta informação, lista todos os fluxos que possuem os IPs de origem e destino do fluxo e retira a ação de desvio. O controlador mantém ainda a ação de encaminhamento para o destino, o que preserva o funcionamento da rede. É importante ressaltar que o sistema proposto é robusto contra ataques de negação de serviço e possui uma grande capacidade de analisar fluxos devido ao fato do fluxo não ser mais encaminhado o sistema depois de realizada a análise. É igualmente importante ressaltar que quando o controlador retira a ação de desvio, a ação voltará a ser ativa após o término do temporizador do fluxo definido no controlador. Este procedimento faz com que os fluxos sejam checados periodicamente, aumentando a segurança da rede e evitando que atacantes burlem o sistema proposto ao começar com um fluxo normal e depois mudar para o ataque. Como o fluxo será analisado novamente após o

© 2017 Sociedade Brasileira de Computação

término do temporizador, o ataque será detectado, mesmo com o início sem ameaças.

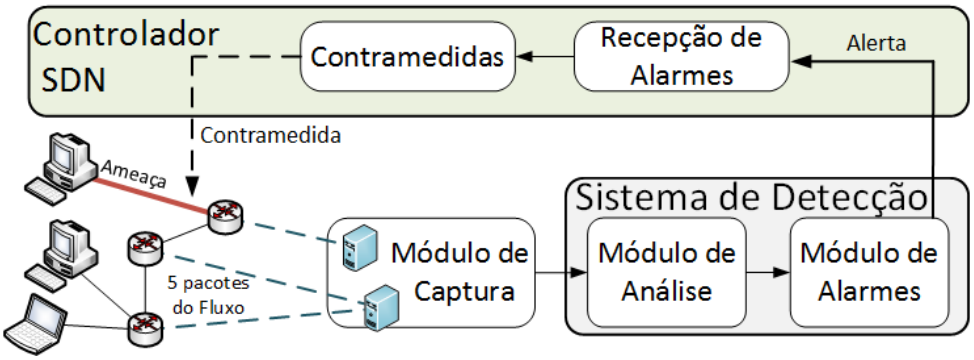


Figura 4: Esquema de prevenção de ameaças proposto: i) o controlador instala regras de desvio para o módulo de captura, ii) o tráfego desviado é analisado pelo sistema de detecção que gera alertas e iii) o controlador bloqueia as ameaças.

A Figura 4 mostra uma rede exemplo sendo analisada e protegida pelo sistema de detecção e prevenção de intrusão proposto. Todos os comutadores SDN possuem uma interface para qual o tráfego é duplicado e enviado até a máquina de análise. O controlador é o responsável por instalar essa regra e também por retirá-la quando a máquina já recebeu os cinco pacotes necessários. O módulo de captura, composto pelas máquinas analisadoras, encaminha então as informações dos fluxos para o sistema de detecção, que utiliza algoritmos adaptativos de aprendizado de máquina em tempo real para fazer sua análise. Caso alguma ameaça seja detectada, o sistema de detecção envia um alerta para o controlador, que então bloqueia o IP de origem do atacante em todos os comutadores.

5.1. Estratégia de Defesa contra Ataques com IP Mascarado

A regra de bloqueio do IP origem nos comutadores não é efetiva contra ataques que alternam periodicamente o IP de origem para enganar os sistemas de defesa. A situação é ainda mais crítica quando o encaminhamento do tráfego é feito usando as redes definidas por *software*, já que além dos danos do ataque, outras duas consequências são a sobrecarga no controlador para definir novos fluxos a cada mudança de IP e a ocupação excessiva das tabelas de encaminhamento dos comutadores pelos fluxos criados com os IPs falsos. Para resolver esse problema, uma estratégia de defesa contra ataques com IP mascarado foi proposta e desenvolvida, baseando-se em uma sequência de alertas e na marcação de caminho dos fluxo. A intuição por trás desse conceito é a seguinte: se o controlador instalou uma regra de bloqueio contra um ataque e, mesmo assim, um alarme chegou logo depois, pode significar que a regra de bloqueio não foi eficaz. Portanto, o controlador mantém um controle de tempo entre o recebimento de alertas e quando dois alertas chegam em um intervalo de tempo curto, o controlador ”suspeita”de que o IP do ataque está sendo mascarado. A partir dessa suspeita, o controlador passa a mapear o caminho de todos os fluxos da rede por um determinado período de tempo. Isto é possível graças à visão global do controlador, que conhece toda a topologia da rede. Agora, se um terceiro alerta chega ao controlador, além de fazer a regra tradicional de bloqueio do IP de origem, o controlador também consegue descobrir por qual porta do comutador o atacante entra na rede e pode instanciar uma regra de bloqueio diretamente na porta em questão. Portanto, se um atacante estiver mascarando o IP do ataque, após três alertas, seu tráfego de ataque será bloqueado. Um aspecto importante a ser destacado é que o processamento adicional para o controlador monitorar o caminho de todos os fluxos só ocorre quando dois alertas ocorrem em um curto período de tempo e é desfeito após um tempo sem receber nenhum novo alerta.

 © 2017 Sociedade Brasileira de Computação

Além disso, a regra de bloqueio do IP que está atacando é mantida, pois pode não se tratar de um ataque com IP mascarado, mas sim um ataque distribuído. Contra um ataque distribuído, é essencial que se tente bloquear todas as origens e, ao impedir o tráfego tanto da interface de entrada na rede quanto do IP de origem, a regra de bloqueio será mais efetiva.

5.2. Monitoramento e Bloqueio de Ameaças

Foi desenvolvido em um ambiente real uma rede de teste sobre uma plataforma de experimentação para avaliar o protótipo do sistema proposto, o esquema de monitoramento de apenas cinco pacotes e o bloqueio do tráfego usando redes definidas por *software*. A virtualização é realizada através do hipervisor Xen e o encaminhamento do tráfego é desempenhado pelo OpenFlow. A Figura 5 mostra a topologia montada para os experimentos, que é formada por três máquinas clientes que se comunicam com uma máquina servidora. A comunicação é feita através de comutadores Open vSwitch que são controlados por uma aplicação programada no controlador POX. Além disso, também há a presença de uma máquina de análise que caracteriza os fluxos usando a ferramenta Bro. Na ligação entre os comutadores e a máquina de análise é necessário um túnel GRE (*Generic Routing Encapsulation*), que encapsula os pacotes e coloca o endereço da máquina de análise como destino. Na máquina de análise, os pacotes são então desencapsulados e analisados. Os experimentos foram realizados em um servidor Intel Xeon X5690 com 24 núcleos de processamento, cada um deles com frequência de 3.47GHz de relógio e com 48 GB de memória RAM.

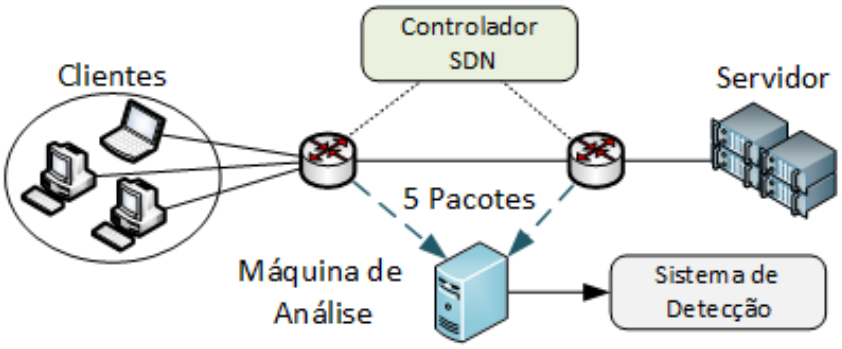
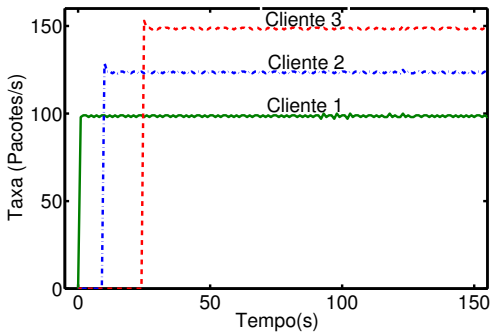


Figura 5: Topologia de rede implementada para os testes do esquema de análise de apenas cinco pacotes e do bloqueio do tráfego malicioso.

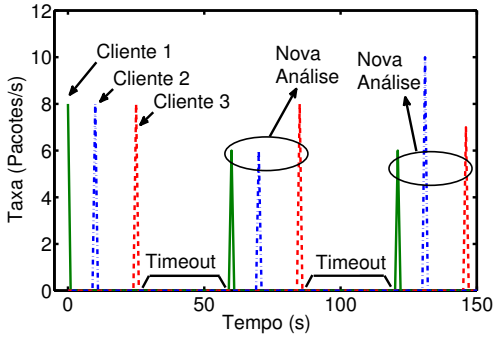
A Figura 6 mostra o funcionamento da regra de desvio de tráfego e seu posterior fim após a análise de cinco pacotes. Nesse experimento, as três máquinas clientes enviam tráfego a uma taxa constante para a máquina servidora. A Figura 6a mostra que o esquema proposto funciona muito bem, sem influir no tráfego de produção recebido pela máquina servidora. Já a Figura 6b mostra a taxa de pacotes recebida pela máquina de análise, que envia uma mensagem ao controlador depois de conseguir capturar os cinco pacotes necessários para caracterizar o fluxo. Por isso, apesar da taxa de envio ser bem mais elevada, a máquina de análise recebe poucos pacotes. A razão pela qual ela recebe um pouco mais que cinco pacotes é o tempo de comunicação entre a máquina de análise e o controlador. Outro comportamento que pode ser observado nessa figura é que os fluxos são analisados periodicamente e, sempre que o tempo de duração do fluxo definido no controlador se esgota, o fluxo é analisado novamente. Nesse experimento, foi definido o tempo de 60 segundos e é possível perceber que a máquina de análise recebe os pacotes a serem analisados de cada cliente periodicamente. Dois aspectos importantes de serem ressaltados nessa proposta são o tempo necessário para caracterizar o fluxo e a capacidade de se aumentar a quantidade de fluxos a serem analisados pela máquina

© 2017 Sociedade Brasileira de Computação

de análise. Como a máquina só requer cinco pacotes para caracterizar o fluxo, ela não precisa esperar até o fim do fluxo ou da conexão para enviar as informações para o sistema de detecção de intrusão, o que tem como consequência um tempo menor na detecção e bloqueio de ameaças. Além disso, a máquina de análise fica imune à ataques de inundação, pois não recebe todos os pacotes de um fluxo. Pela mesma razão, também se pode analisar uma quantidade muito maior de fluxos, garantindo robustez e um potencial para escalabilidade.



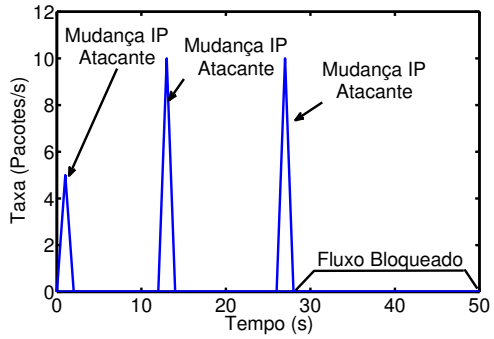
(a) Taxa de pacotes recebidos pela máquina servidora.



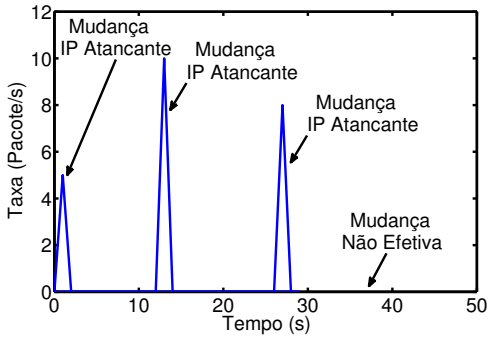
(b) Taxa de pacotes recebidos pela máquina de análise.

Figura 6: Funcionamento da rede quando há a comunicação de três máquinas com o servidor. O servidor recebe o tráfego sem ser afetado pelo esquema, enquanto a máquina de análise recebe poucos pacotes de cada fluxo.

O segundo experimento mostra uma situação de ameaça na qual uma das máquinas cliente ataca o servidor. A máquina atacante, além de realizar a ameaça, mascara o IP de origem para dificultar ainda mais a detecção. De maneira semelhante ao primeiro experimento, a Figura 7 mostra o tráfego recebido pela máquina servidora e pela máquina de análise. Após o ataque começar e a máquina análise enviar as informações para o sistema de detecção de intrusão, uma regra bloqueando o IP de origem imediatamente é instalada em todos os comutadores, o que faz com que o tráfego recebido pela servidora fique com taxa zero após o tempo de detecção, como ilustrado na Figura 7a.



(a) Taxa de pacotes do atacante recebidos pela máquina servidora.



(b) Taxa de pacotes do atacante recebido pela máquina de análise.

Figura 7: Funcionamento do sistema proposto sob um ataque que mascara o endereço IP de origem. As regras de bloqueio deixam de ser efetivas quando a máquina modifica o IP de origem e o bloqueio volta a ser efetivo após a identificação e localização da interface pela qual o ataque entra na rede.

Assim, quando a máquina altera o IP de origem pela primeira vez, o fluxo volta a chegar na máquina servidora e é novamente desviado e analisado, como mostrado na Figura 7b.

© 2017 Sociedade Brasileira de Computação

Porém, desta vez quando o controlador recebe o alerta, ele passa a marcar o caminho dos fluxos para poder mapear a entrada na rede de cada fluxo. Mais uma vez o fluxo do IP de origem é bloqueado e a máquina servidora para de receber tráfego. Novamente, em torno de 28 segundos a máquina atacante muda seu IP de origem e o tráfego é novamente analisado e um novo alerta é gerado. Contudo, dessa vez, quando o controlador recebe o alerta, ele procura por qual interface do comutador o tráfego desse IP está vindo e cria uma regra para bloquear o tráfego dessa interface. Agora, quando o atacante altera novamente o IP, o tráfego dele não chega mais na máquina servidora, nem é analisado novamente. Isso ocorre pois o tráfego malicioso é bloqueado o mais perto possível da origem. Uma observação importante é que apesar das Figuras 7a e 7b serem muito parecidas, a máquina de análise só recebe essa quantidade pequena de pacotes por causa da regra de fim de desvio que o controlador instala, enquanto no resto da rede o bloqueio acontece em um tempo curto após a geração do alerta.

6. Conclusão

Este artigo apresentou um sistema adaptativo e eficaz de detecção e bloqueio de tráfegos maliciosos. A detecção do sistema proposto é feita com base na análise das características de cinco pacotes de cada fluxo. Essa abordagem se mostra eficiente em diversos aspectos. Primeiro, os resultados mostraram um excelente desempenho, classificando as amostras de fluxo com acurácias maiores que 90%. Além disso, ao se analisar apenas cinco pacotes, a detecção de ameaças é efetuada praticamente de maneira instantânea, sem ter de esperar o fim das conexões para conseguir classificar os fluxos como legítimos ou maliciosos. Por fim, a máquina que analisa o tráfego da rede é colocada fora da rota do pacote e, como só recebe poucos pacotes de cada fluxo, ela fica imune a ataques de inundação que poderiam vir a impedir que ela extraísse as características de todos os fluxos.

Três algoritmos de detecção de ameaças foram propostos: dois por classificação e um por anomalia. Os três utilizam os dados provenientes de *honeypots* como assinaturas de ataques para adaptar seus modelos em tempo real e aprender a detectar novas ameaças. O algoritmo de detecção de anomalias apresenta uma boa relação entre detecção e falsos positivos e os de classificação se mostraram capazes de se adaptar a novas ameaças e mudança de comportamento legítimo. Por fim, todo o esquema para a prevenção de intrusão foi realizado com o auxílio das redes definidas por *software*. Como prova de conceito, foi implementado um protótipo em uma plataforma híbrida de testes que combina o hypervisor Xen com o OpenFlow. Além disso, há uma estratégia de defesa contra os tipos de ataques que mudam dinamicamente de IP. Isso é essencial, pois, além de ser uma ameaça à máquina que estiver sendo atacada, o mascaramento do IP também tem como consequência uma negação de serviço do controlador. Os resultados mostraram o efetivo bloqueio de uma ameaça ao se mapear o caminho do fluxo de acordo com o intervalo de tempo dos alertas, que podem indicar a presença de um ataque com IP mascarado.

Referências

Andreoni Lopez, M., Mattos, D. M. F. e Duarte, O. C. M. B. (2016). An elastic intrusion detection system for software networks. *Annales des Telecommunications/Annals of Telecommunications*, 71(11-12):595–605.

Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A. e Salamatian, K. (2006). Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26.

Braga, R., Mota, E. e Passito, A. (2010). Lightweight DDoS flooding attack detection using NOX/OpenFlow. Em *IEEE 35th Conference on Local Computer Networks*, páginas 408–415.

© 2017 Sociedade Brasileira de Computação

Buczak, A. e Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, (99):1–26.

Donato, W. D., Pescape, A. e Dainotti, A. (2014). Traffic identification engine: an open platform for traffic classification. *IEEE Network*, 28(2):56–64.

Garcia, S., Grill, M., Stiborek, J. e Zunino, A. (2014). An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123.

Ji, S.-Y., Jeong, B.-K., Choi, S. e Jeong, D. H. (2016). A multi-level intrusion detection method for abnormal network behaviors. *Journal of Network and Computer Applications*, 62:9–17.

Lakhina, A., Crovella, M. e Diot, C. (2005). Mining anomalies using traffic feature distributions. Em *ACM SIGCOMM Computer Communication Review*, volume 35, páginas 217–228. ACM.

Lee, W., Stolfo, S. J. e Mok, K. W. (1999). Mining in a data-flow environment: Experience in network intrusion detection. Em *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, páginas 114–124. ACM.

Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K. et al. (2000). Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. Em *Proceedings of DARPA Information Survivability Conference and Exposition. DISCEX'00.*, volume 2, páginas 12–26. IEEE.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. e Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev., 2008*.

Peng, L., Yang, B. e Chen, Y. (2015). Effective packet number for early stage internet traffic identification. *Neurocomputing*, 156:252 – 267.

Ponemon, I. e IBM (2017). 2017 cost of data breach study: Global analysis. `www.ibm.com/security/data-breach/`. Acessado: 15/07/2017.

Sharafaldin, I., Gharib, A., Lashkari, A. H. e Ghorbani, A. A. (2017). Towards a reliable intrusion detection benchmark dataset. *Software Networking*, 2017(1):177–200.

Sommer, R. e Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. Em *IEEE Symposium on Security and Privacy (SP)*, páginas 305–316. IEEE.

Suthaharan, S. (2014). Big data classification: Problems and challenges in network intrusion prediction with machine learning. *ACM SIGMETRICS Performance Evaluation Review*, 41(4):70–73.

Wu, K., Zhang, K., Fan, W., Edwards, A. e Yu, P. S. (2014). RS-Forest: A rapid density estimator for streaming anomaly detection. Em *IEEE International Conference on Data Mining (ICDM)*, páginas 600–609.

 © 2017 Sociedade Brasileira de Computação

Uso de Controle Chaveado para Mitigação de Ataque Ativo de Identificação de Sistemas com Malha Fechada

Alan Oliveira de Sá[1,2], **Luiz F. Rust da Costa Carmo**[1,3], **Raphael C. S. Machado**[3,4]

[1]Programa de Pós-Graduação em Informática - Instituto Tércio Pacitti / IM,
Universidade Federal do Rio de Janeiro, 21.941-901, RJ – Brasil

[2]Centro de Instrução Almirante Wandenkolk – Marinha do Brasil,
Ilha das Enxadas, Baía de Guanabara – Rio de Janeiro – RJ – Brasil

[3]Instituto Nacional de Metrologia, Qualidade e Tecnologia (Inmetro)
Av. Nossa Senhora das Graças, 50, Xerém, Duque de Caxias, 25.250-020, RJ – Brasil

[4]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Av. Maracanã, 229, Maracanã, 20.271-110 - Rio de Janeiro – RJ – Brasil

alan.oliveira.sa@gmail.com, {lfrust,rcmachado}@inmetro.gov.br

Abstract. *The literature regarding to cyber-physical attacks in Networked Control Systems (NCS) indicates that covert and accurate attacks must be planned based on an accurate knowledge about the model of the attacked system. In this sense, the literature on NCS recognizes the Active System Identification attack as a tool to provide the attacker with the required system models. However, there is still a lack of discussion about countermeasures for this specific attack. In this sense, this work proposes the use of a randomly switching controller as a countermeasure for the Active System Identification attack. The simulation results indicate that this countermeasure is capable to mitigate the mentioned attack at the same time that it performs a satisfactory plant control.*

Resumo. *A literatura referente aos ataques físico-cibernéticos em Sistemas de Controle em Rede indica que ataques furtivos e acurados devem ser planejados com base em um conhecimento igualmente acurado sobre o modelo do sistema atacado. Neste contexto, a literatura sobre Sistemas de Controle em Rede reconhece o ataque Ativo de Identificação de Sistemas como uma ferramenta capaz de prover tais modelos ao atacante. No entanto, há uma carência de discussão sobre contramedidas para este ataque específico. Nesse sentido, este trabalho propõe o uso de controladores aleatoriamente chaveados como uma contramedida para o ataque Ativo de Identificação de Sistemas. Os resultados de simulação indicam que esta contramedida é capaz de mitigar o referido ataque e executar, simultaneamente, um controle satisfatório da planta.*

1. Introdução

Um Sistema de Controle em Rede, ou *Networked Control System* (NCS), consiste em uma planta física controlada por um controlador digital – *i.e.* um sistema computacional – por meio de uma rede de comunicação que, deste modo, integra o ciberespaço ao domínio físico. Considerando o crescente uso de NCSs em plantas industriais e infraestruturas críticas, bem como as ameaças cibernéticas que podem afetar estes sistemas, estudos têm sido realizados para caracterizar vulnerabilidades e propor soluções de segurança para tais sistemas. Neste contexto, a literatura [Smith 2011, Amin et al. 2013, Teixeira et al. 2015,

 © 2017 Sociedade Brasileira de Computação

Smith 2015, de Sá et al. 2017b] demonstra que uma série de ataques sofisticados a NCSs requerem um conhecimento prévio sobre o modelo do sistema atacado. Considerando esta necessidade, recentes trabalhos [de Sá et al. 2017b, de Sá et al. 2017a] apresentam um conjunto de ataques de Identificação de Sistemas que podem ser lançados contra NCSs para fornecer ao atacante os requeridos modelos e, portanto, subsidiar a elaboração de outros ataques.

A literatura [de Sá et al. 2017b, Smith 2011, Smith 2015, Teixeira et al. 2015] indica que, para ser eficaz, ataques furtivos e acurados devem contar com um conhecimento igualmente acurado sobre o modelo do sistema atacado. Em [de Sá et al. 2017b], por exemplo, a operação conjunta de um ataque de Identificação de Sistemas e um ataque de Injeção de Dados é utilizada para degradar, de forma fisicamente furtiva, o serviço desempenhado por uma planta. No referido trabalho, é demonstrado que a performance deste ataque furtivo de injeção de dados é diretamente afetado pela acurácia da informação obtida pelo ataque de Identificação de Sistemas.

Conforme mostrado na Figura 1, o conhecimento sobre o sistema é um dos requisitos para o lançamento de uma série de ataques dependentes de modelo, tais como ataques controlados de *Denial of Service* (DoS-Controlled) [de Sá et al. 2017b] e ataques controlados de *Service Degradation* (SD-Controlled) [de Sá et al. 2017b]. Nesta figura, é possível ver que o Ataque Ativo de Identificação de Sistemas, ou *Active System Identification*, proposto em [de Sá et al. 2017a] constitui um caminho para a implementação dos referidos ataques, uma vez que ele é capaz de proporcionar ao atacante o conhecimento necessário sobre o sistema. Embora os autores de [de Sá et al. 2017a] encorajem o desenvolvimento de contramedidas para o Ataque Ativo de Identificação de Sistemas, há carência de discussão sobre contramedidas para este ataque específico. Neste sentido, este trabalho tem por objetivo discutir e propor uma contramedida para o referido ataque.

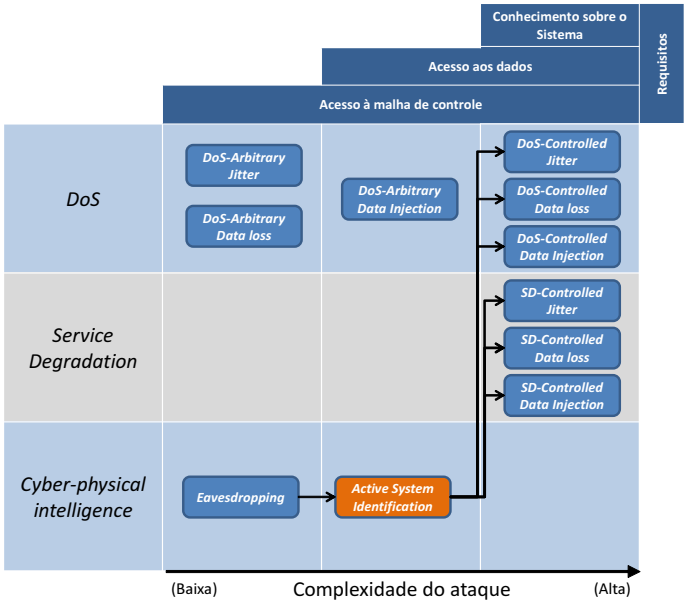


Figura 1. Categorias e requisitos de ataques em NCSs.

A contramedida mais evidente para prevenir o sucesso de um ataque de Identificação de Sistemas em um NCS é prevenir o acesso não autorizado à malha de controle utilizando, por exemplo, segmentação de rede, *demilitarized zones* (DMZ), políticas de *firewall* e implementando arquiteturas de rede específicas, tal como estabe-

 © 2017 Sociedade Brasileira de Computação

lecido em [Stouffer et al. 2015]. Uma contramedida complementar – para o caso de o atacante conseguir acessar a malha de controle – é prevenir o acesso aos dados que fluem no NCS utilizando, por exemplo, algoritmos de criptografia de chaves simétricas, algoritmos de *hash* e uma estratégia de *timestamp* para formar um mecanismo seguro de comunicação entre o controlador e a planta, tal como proposto em [Pang and Liu 2012]. Entretanto, quando as referidas contramedidas falham e o atacante obtém acesso aos dados que fluem no NCS, a alternativa para prevenir que o atacante obtenha o modelo do sistema é dificultar a análise do dado capturado – *i.e.* tornar o algoritmo de Identificação de Sistemas impreciso/ineficaz.

Uma possível estratégia para causar dificuldades ao algoritmo de Identificação de Sistemas é ter no NCS funções de controle específicas que sejam, ao mesmo tempo, mais difíceis de serem identificadas e capazes de controlar a planta. Considerando esta estratégia, é proposto neste trabalho o uso de controladores aleatoriamente chaveados como uma contramedida factível para o Ataque Ativo de Identificação de Sistemas, proposto em [de Sá et al. 2017a].

O restante deste artigo está organizado da seguinte forma. Na Seção 2, são apresentados trabalhos relacionados à esta pesquisa. Na Seção 3, o ataque Ativo de Identificação de Sistemas proposto em [de Sá et al. 2017a] é brevemente descrito. Na Seção 4, o controlador chaveado é apresentado e discutido como uma contramedida para o Ataque Ativo de Identificação de Sistemas. A Seção 5 apresenta resultados de simulações, onde o desempenho do controlador chaveado é analisado do ponto de vista da contramedida e do controle. Finalmente, na Seção 6, são apresentadas as conclusões e possíveis trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção é apresentada uma revisão sobre trabalhos relacionados aos ataques físico-cibernéticos em NCSs. O foco está em estudos recentes que englobam ataques furtivos e dependentes de modelo, bem como ataques de Identificação de Sistemas.

Em [Teixeira et al. 2015], os autores analisam diversos ataques em NCSs e estabelecem que a implementação de ataques furtivos requer um elevado conhecimento sobre o modelo do sistema atacado. Em [Smith 2011, Amin et al. 2013, Smith 2015], são propostos e analisados ataques furtivos que corroboram o requisito estabelecido por [Teixeira et al. 2015]. Em [Smith 2011, Smith 2015], o atacante, atuando como um *Man-in-the-Middle* (MitM), injeta dados falsos nos sinais de controle do NCS para assumir o controle da planta. O atacante, então, utiliza o modelo da planta atacada para calcular os dados que são injetados no sinal de realimentação do NCS, a fim de tornar o ataque furtivo. A furtividade do ataque proposto em [Smith 2011] é analisada do ponto de vista dos sinais que chegam ao controlador e, conforme demonstrado em [Smith 2015], depende da diferença entre o modelo real da planta e o modelo conhecido pelo atacante. Em [Amin et al. 2013] o atacante, conhecendo o modelo do sistema atacado, injeta dados falsos no NCS para furtivamente roubar água do canal Gignac, localizado no sul da França.

Em [Amin et al. 2013, Smith 2011, Smith 2015, Teixeira et al. 2015], onde os ataques são planejados e implementados com base nos modelos dos sistemas atacados, não é descrito como estes modelos são obtidos pelo atacante. Considera-se apenas que os modelos são previamente conhecidos e, então, utilizados para subsidiar o planejamento destes ataques furtivos/dependentes de modelo.

© 2017 Sociedade Brasileira de Computação

Para preencher este hiato, em [de Sá et al. 2017a, de Sá et al. 2017b], os autores propõem dois novos tipos de ataque: o Ataque Passivo de Identificação de Sistemas, ou *Passive System Identification Attack* [de Sá et al. 2017b]; e o Ataque Ativo de Identificação de Sistemas, ou *Active System Identification Attack* [de Sá et al. 2017a]. Estes ataques, que pertencem à categoria de Ataques de Inteligência Físico-cibernética, ou *Cyber-physical Intelligence Attacks* [de Sá et al. 2017b], visam estimar o modelo dos sistemas atacados. O Ataque Passivo de Identificação de Sistemas [de Sá et al. 2017b] não precisa injetar sinais no NCS para estimar seu modelo. Entretanto, o mesmo depende da ocorrência de eventos, que não são controlados pelo atacante, para produzir sinais que carreguem informações significativas para o algoritmo de identificação. Tal ataque estima a função de transferência de cada dispositivo do NCS – *i.e.* controlador e planta – capturando passivamente os sinais de controle e de realimentação do sistema. O Ataque Ativo de Identificação de Sistemas [de Sá et al. 2017a], por sua vez, constitui uma alternativa aos Ataques Passivos de Identificação de Sistemas em situações em que o atacante não pode aguardar muito tempo pela ocorrência de sinais que contenham informações significativas para o processo de identificação. Para isso, conforme descrito na Seção 3, o atacante estima a função de transferência de malha aberta do NCS injetando um sinal de ataque e capturando a sua consequente resposta. Ambas as ações – *i.e.* injeção e captura de dados – são realizadas em um único ponto de interceptação do sistema. Neste trabalho é proposta uma contramedida para mitigar o Ataque Ativo de Identificação de Sistemas, mesmo que o atacante obtenha acesso aos dados transmitidos no NCS.

3. O Ataque Ativo de Identificação de Sistemas

Nesta Seção, o ataque Ativo de Identificação de Sistemas proposto em [de Sá et al. 2017a] é brevemente descrito, a fim de prover as informações basilares e necessárias à compreensão da contramedida proposta no presente trabalho. O referido ataque visa estimar os coeficientes da função de transferência de malha aberta $G(z) = C(z)P(z)$ de um NCS, apresentado na Figura 2, onde $C(z)$ é a função de controle executada pelo controlador e $P(z)$ é o modelo da planta. Para tal, o ataque é realizado em três etapas:

- ETAPA-I: Primeiramente, agindo como um *Man-in-the-Middle* (MitM), o atacante injeta um sinal de ataque $a(k)$ no NCS, conforme apresentado na Figura 2.
- ETAPA-II: Após injetar $a(k)$, o atacante captura a saída $y(k)$ da planta, durante um período de monitoração T, a fim de obter a resposta $y_a(k)$ causada por $a(k)$.
- ETAPA-III: Conhecendo o sinal de ataque $a(k)$ e a consequente resposta $y_a(k)$ do NCS, o atacante estima a função de transferência de malha aberta do sistema $G(z)$ aplicando $a(k)$ em um modelo estimado $G_e(z)$, o qual é ajustado até que a saída estimada $\hat{y}_a(k)$ coincida com $y_a(k)$. Em [de Sá et al. 2017a], este ajuste é realizado por algoritmos de otimização bio-inspirados.

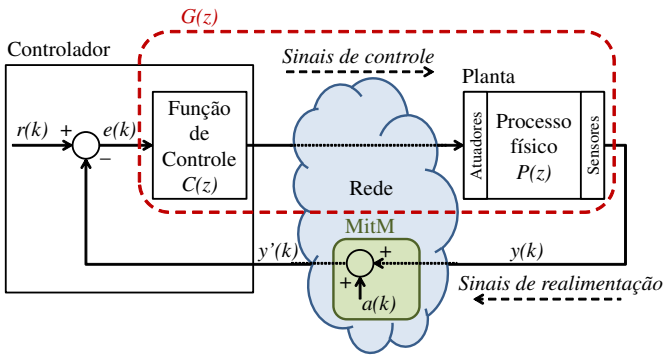


Figura 2. Ataque Ativo de Identificação de Sistema em NCS. [de Sá et al. 2017a]

© 2017 Sociedade Brasileira de Computação

Note que $y_a(k)$, obtido na ETAPA-II, é apenas uma parcela de $y(k)$. A resposta completa do sistema é $y(k) = y_r(k) + y_a(k)$, onde $y_r(k)$ é a resposta do sistema causada por $r(k)$. Considerando que o sistema é estável, a parcela $y_r(k)$ converge e estabiliza em um valor constante q após um determinado número de amostras k_s. Assim, na ETAPA-II, para obter $y_a(k)$, $\forall k > k_s$, o atacante deve calcular $y_a(k) = y(k) - q$. Isto elimina a parcela de $y(k)$ causada por $r(k)$, tornando o problema de identificação dependente de apenas o sinal de ataque $A(z) = \mathcal{Z}[a(k)]$, conforme descrito por (1):

$$y_a(k) = y(k) - q = -\mathcal{Z}^{-1}\left[\frac{G(z)}{1+G(z)}A(z)\right], \forall k > k_s. \quad (1)$$

onde $\mathcal{Z}$ representa a operação da transformada Z. O valor de q é obtido pelo atacante através da captura de $y(k)$ após a estabilização do NCS, antes da injeção de $a(k)$.

Na ETAPA-III, o atacante estima $G(z)$ aplicando $a(k)$ em um modelo estimado, definido por (2):

$$\hat{y}_a(k) = -\mathcal{Z}^{-1}\left[\frac{G_e(z)}{1+G_e(z)}\right] * a(k), \quad (2)$$

onde $G_e(z)$ é a estimativa de $G(z)$ e $\hat{y}_a(k)$ é a saída do sistema estimado. A função de transferência genérica $G_e(z)$, é definida como (3):

$$G_e(z) = \frac{\alpha_n z^n + \alpha_{n-1}z^{n-1} + ... + \alpha_1 z^1 + \alpha_0}{z^m + \beta_{m-1}z^{m-1} + ... + \beta_1 z^1 + \beta_0}, \quad (3)$$

onde $[\alpha_n, \alpha_{n-1}, ...\alpha_1, \alpha_0]$ e $[\beta_{m-1}, \beta_{m-2}, ...\beta_1, \beta_0]$ são os coeficientes do numerador e do denominador, respectivamente, os quais o Ataque Ativo de Identificação de Sistemas visa descobrir. A ordem do numerador e do denominador são expressas por n e m, respectivamente. Deste modo, para encontrar $G(z)$, os coeficientes de $G_e(z)$ são ajustados até que a saída estimada $\hat{y}_a(k)$ convirja para a resposta $y_a(k)$, conhecida.

Em [de Sá et al. 2017a], o *Backtracking Search Optimization Algorithm* (BSA) [Civicioglu 2013] e o *Particle Swarm Optimization* (PSO) [Kennedy 1995] são utilizados para ajustar iterativamente os parâmetros de $G_e(z)$, minimizando uma função de aptidão específica, até que $G_e(z)$ convirja para $G(z)$. Os coeficientes de $G_e(z)$ são as coordenadas $x_j = [\alpha_{n,j}, \alpha_{n-1,j}, ...\alpha_{1,j}, \alpha_{0,j}, \beta_{m-1,j}, \beta_{m-2,j}, ...\beta_{1,j}, \beta_{0,j}]$ de um indivíduo j do BSA/PSO. A aptidão f_j de cada indivíduo j do BSA/PSO é calculada de acordo com (4):

$$f_j = \frac{\sum_{k=0}^{N}(y_a(k) - \hat{y}_{aj}(k))^2}{N}, \quad (4)$$

onde N é o número de amostras que existem durante o período de monitoração T da ETAPA-II; e $\hat{y}_{aj}(k)$ é a resposta do modelo estimado (2) causada por $a(k)$, quando os coeficientes de $G_e(z)$ são x_j. Se nenhuma outra entrada – perturbação ou ruído – afetar o NCS durante T, então $\min f_j = 0$ quando $[\alpha_{n,j}, \alpha_{n-1,j}, ...\alpha_{1,j}, \alpha_{0,j}, \beta_{m-1,j}, \beta_{m-2,j}, ...\beta_{1,j}, \beta_{0,j}] = [\alpha_n, \alpha_{n-1}, ...\alpha_1, \alpha_0, \beta_{m-1}, \beta_{m-2}, ...\beta_1, \beta_0]$, *i.e.* quando $G_e(z)$ converge para $G(z)$.

4. Controladores Chaveados: uma Contramedida

Conforme discutido na Seção 1, uma possível forma de causar dificuldades ao algoritmo de identificação de sistemas é ter, no NCS, funções de transferências específicas que sejam mais difíceis de ser identificadas. Sendo assim, é necessário se debruçar sobre as

duas funções de transferência $C(z)$ e $P(z)$, apresentadas na Figura 2, para verificar o que pode ser feito para dificultar a identificação do NCS. No caso da planta, não é desejável ou mesmo viável modificar a sua função de transferência $P(z)$ apenas para torná-la mais difícil de identificar. Isto decorre do simples fato de que a função de transferência da planta é uma consequência da estrutura física da mesma. Portanto, modificar $P(z)$ significa modificar o processo físico que está sendo controlado, o que não é conveniente. Entretanto, é possível projetar uma função de controle de forma que esta atenda, simultaneamente, dois objetivos:

Objetivo I - Cumprir os requisitos de controle da planta considerando, em primeiro lugar, a estabilidade do sistema e, em segundo lugar, outros requisitos, tais como: tempo de acomodação, *overshoot*, etc.

Objetivo II - Dificultar o processo de identificação, fazendo com que o modelo obtido pelo atacante seja impreciso ou ambíguo, de tal forma que o atacante hesite em lançar contra o NCS ataques furtivos ou dependentes de modelo.

Note que, no caso do Ataque Ativo de Identificação de Sistemas proposto em [de Sá et al. 2017a], o atacante não identifica $C(z)$ e $P(z)$ separadamente, analisando suas respectivas entradas e saídas. O atacante intercepta a malha de controle em apenas um ponto e, a partir deste ponto de interceptação, estima a função de transferência de malha aberta do sistema $G(z) = C(z)P(z)$, conforme mostra a Figura 2. Assumindo que não é conveniente modificar $P(z)$, conforme anteriormente discutido, $C(z)$ deve ser projetado para dificultar a identificação da função de transferência de malha aberta $G(z) = C(z)P(z)$ do NCS. Neste sentido, considerando os dois objetivos apresentados, é proposto neste artigo o uso de controladores chaveados como uma contramedida para o Ataque Ativo de Identificação de Sistemas.

Um controlador chaveado consiste em um conjunto de funções de controle $C_i(z)$, $i \in \mathcal{I} = \{1, ..., N\}$, comutadas entre N estados por meio de uma regra de comutação S, para controlar uma planta $P(z)$, conforme apresentado na Figura 3. Quando as funções de controle $C_i(z)$ e a função de transferência da planta $P(z)$ são lineares, como no caso do presente artigo, o sistema é classificado como um sistema linear comutado, ou *switched linear system* (SLS). Por uma questão de simplicidade, mas sem prejuízo à generalidade, neste trabalho, o controlador chaveado é representado e discutido com apenas duas funções de controle $C_1(z)$ e $C_2(z)$ – *i.e.* $i \in \mathcal{I} = \{1, 2\}$.

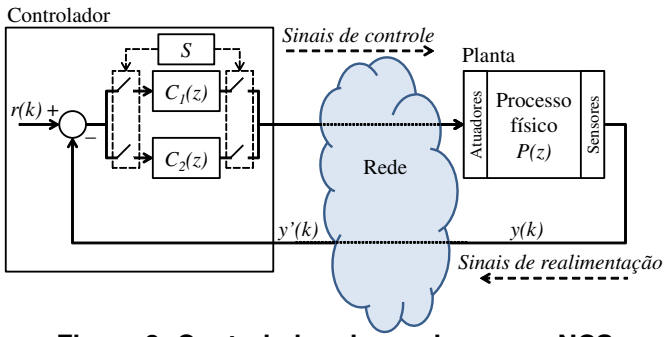


Figura 3. Controlador chaveado em um NCS.

Em geral, a regra de comutação S considera o comportamento da planta para chavear entre as funções de controle, tal como descrito em [Skafidas et al. 1999]. Entretanto, na solução proposta neste trabalho, a regra de comutação não observa o comportamento da planta para comandar os chaveamentos. Para tornar a identificação mais difícil, a regra de

 © 2017 Sociedade Brasileira de Computação

comutação proposta ocorre segundo a cadeia de Markov da Figura 4, onde as funções de controle são chaveadas em intervalos aleatórios seguindo as probabilidades $p_{11}(l)$, $p_{12}(l)$. $p_{21}(l)$ e $p_{12}(l)$, onde l é o número de intervalos de amostragem ocorridos desde o último chaveamento. A razão para chavear as funções de controle em intervalos aleatórios é que, conforme [Wang 2013], se o tempo de cada chaveamento é conhecido, a identificação de um SLS se torna simples. No entanto, quando o tempo de cada chaveamento não é conhecido, a identificação de um SLS se torna uma tarefa não trivial. As probabilidades, $p_{12}(l)$ e $p_{21}(l)$ são obtidas da função de densidade de probabilidade (FDP) da Figura 5, onde a é o número mínimo de intervalos de amostragem que o sistema deve permanecer no mesmo estado e b é o número máximo de intervalos de amostragem que o sistema pode permanecer no mesmo estado. Note que $p_{11}(l) = 1 - p_{12}(l)$ e $p_{22}(l) = 1 - p_{21}(l)$.

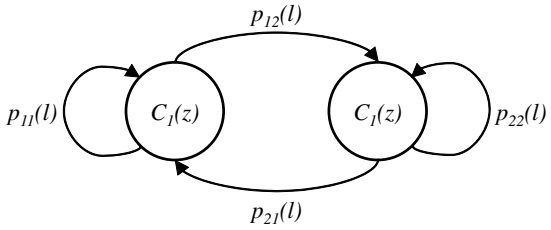


Figura 4. Regra de comutação pela cadeia de Markov.

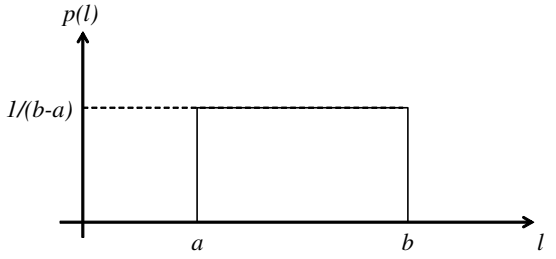


Figura 5. FDP de p_{12} e p_{21}.

Controladores com chaveamento estocástico, modelados por cadeias de Markov, já existem na literatura [Yang et al. 2011] com o objetivo de melhorar a estabilização e a resposta dinâmica do NCS em face de longos atrasos aleatórios na rede. No entanto, no presente trabalho – diferente de [Yang et al. 2011] –, as probabilidades de transição e os estados da cadeia são dimensionados para mitigar um ataque de identificação de sistemas mantendo a qualidade do controle.

Para ser estável sob comutações arbitrárias e sem restrições, o SLS deve atender a duas condições [Lin and Antsaklis 2009, Dasgupta et al. 2013]:

1. Todos os subsistemas – onde cada subsistema corresponde à função de transferência da planta $P(z)$ disposta em malha fechada com uma das funções de controle $C_i(z)$ – devem ser assintoticamente estáveis; e
2. Deve existir uma função de Lyapunov que seja comum para todos os subsistemas.

Assim, para tornar o SLS estável sob comutações arbitrárias e sem restrições, todas as funções de controle $C_i(z)$ devem ser projetadas de forma a atender às duas condições acima descritas.

Outra estratégia válida para alcançar a estabilidade em um SLS é restringir os eventos de chaveamento estabelecendo, por exemplo, um tempo mínimo entre dois chaveamentos consecutivos – *i.e.* um mínimo *dwell time*. Em um SLS, a instabilidade gerada ao chavear entre dois subsistemas estáveis é causada pela incapacidade de absorver o aumento de energia gerado pelas sucessivas comutações [Lin and Antsaklis 2009]. Intuitivamente, é razoável pensar que se o sistema permanecer nos subsistemas estáveis por um tempo suficiente – usando uma regra de comutação lenta – é possível prevenir o aumento de energia causado pelos chaveamentos. Com base neste conceito, em [Morse 1996], é provado que a estabilidade global de um SLS é garantida quando todos os seus subsistemas são estáveis e o *dwell time* é suficientemente grande. No entanto, é factível que o SLS ocasionalmente tenha um *dwell time* menor, desde que isso não ocorra com

© 2017 Sociedade Brasileira de Computação

frequência. Conforme apresentado em [Hespanha and Morse 1999], se todos os subsistemas forem exponencialmente estáveis, então o SLS será estável desde que o tempo médio de comutação seja suficientemente grande. Este conceito, referido como *average dwell-time*, é estendido em [Zhai et al. 2002] para SLSs discretos no tempo – que é o caso de NCSs com controle chaveado.

Neste trabalho, ao invés de projetar $C_1(z)$ e $C_2(z)$ para que o SLS seja estável sob comutações arbitrárias e sem restrições – *i.e.* atendendo às condições 1 e 2, previamente descritas – utiliza-se a estratégia de restringir as comutações do SLS. Neste sentido, primeiramente, $C_1(z)$ e $C_2(z)$ são dimensionadas de forma independente com base na análise do lugar das raízes, a fim de que cada subsistema seja estável. A estabilidade global do SLS é, então, alcançada ajustando os parâmetros a e b da FDP apresentada na Figura 5, visando um *average dwell-time* que torne o NCS estável. Os parâmetros a e b, além de serem ajustados visando a estabilidade, também devem ser ajustados para dificultar o ataque de identificação de sistemas. Sob este aspecto, a literatura [Baştuğ 2012] indica que pequenos *dwell times* dificultam o processo de identificação do sistema.

Assim, no que diz respeito ao Objetivo I, especificamente por questão de estabilidade, a e b são aumentados, tanto quanto possível, para garantir o *average dwell-time* necessário para a estabilidade. Por outro lado, no que diz respeito ao Objetivo II – *i.e.* dificultar o ataque de identificação de sistemas –, a e b devem ser diminuídos, tanto quanto possível, para reduzir os *dwell times* do SLS. Considerando estes aspectos, no presente trabalho, a e b são empiricamente ajustados para satisfazer os dois objetivos conflitantes.

É digno de nota que, além de dificultar o Ataque Ativo de Identificação de Sistemas proposto em [de Sá et al. 2017a], a regra de chaveamento aleatório adotada também dificulta o lançamento de ataques furtivos/dependentes de modelo – mesmo que o atacante conheça as funções de controle $C_i(z)$. Isto decorre do simples fato de que é mais difícil sincronizar o ataque furtivo/dependente de modelo com as funções de controle $C_i(z)$ que alternam em intervalos aleatórios.

5. Resultados

Nesta seção, a performance da contramedida proposta é analisada em face do Ataque Ativo de Identificação de Sistemas de [de Sá et al. 2017a]. Nas simulações são feitas comparações entre dois NCSs: um com a contramedida proposta – *i.e.* utilizando um controlador chaveado; e outro sem a contramedida proposta – *i.e.* utilizando um controlador não chaveado. A performance do ataque a cada um dos NCSs é avaliada por meio de um conjunto de simulações realizadas no MATLAB. Os modelos dos NCSs, bem como os parâmetros do Ataque Ativo de Identificação de Sistemas, são descritos na Seção 5.1.

Cabe lembrar que, conforme definido na Seção 4, o projeto do controlador chaveado deve atender simultaneamente dois objetivos: cumprir os requisitos de controle da planta; e dificultar o processo de identificação. Neste sentido, a Seção 5.2 analisa os resultados do controlador chaveado sob o ponto de vista de uma contramedida para o Ataque Ativo de Identificação de Sistemas. A Seção 5.3, por sua vez, analisa o desempenho da contramedida proposta sob o ponto de vista do controle, a fim de identificar possíveis relações de compromisso que possam existir entre os dois objetivos.

5.1. NCSs Atacados e Parâmetros do Ataque

O NCS sem a contramedida proposta – também referido neste artigo como *sistema com modelo vulnerável* – é o mesmo NCS atacado em [de Sá et al. 2017a]. Consiste em

 © 2017 Sociedade Brasileira de Computação

um motor DC cuja velocidade de rotação é controlada por um controlador Proporcional-Integral (PI), não chaveado. A função de transferência do motor DC $P(z)$ e a função de controle PI $C_1(z)$ são representadas por (5) e (6), respectivamente:

$$P(z) = \frac{0.3379z + 0.2793}{z^2 - 1.5462z + 0.5646}, \quad (5) \qquad C_1(z) = \frac{0.1701z - 0.1673}{z - 1}. \quad (6)$$

Assim, a função de transferência em malha fechada do *sistema com modelo vulnerável* $G_1(z)$, a ser identificada pelo atacante, é definida por (7):

$$G_1(z) = C_1(z)P(z) = \frac{g_{1,1}z^2 + g_{2,1}z + g_{3,1}}{z^3 + g_{4,1}z^2 + g_{5,1}z + g_{6,1}}, \quad (7)$$

onde $g_{1,1} = 0.0575$, $g_{2,1} = -0.0090$, $g_{3,1} = -0.0467$, $g_{4,1} = -2.5462$, $g_{5,1} = 2.1108$ e $g_{6,1} = -0.5646$.

O NCS dotado da contramedida proposta – *i.e.* com controlador chaveado – também controla um motor DC definido pela função de transferência (5). O controlador chaveado alterna entre duas funções de controle: $C_1(z)$, que é a mesma função de controle (6) do *sistema com modelo vulnerável*; e $C_2(z)$ que é definida por (8):

$$C_2(z) = \frac{0.1208z - 0,1167}{z - 1}. \quad (8)$$

Portanto, o NCS com o controlador chaveado é um SLS composto por dois subsistemas, cada um tendo uma função de transferência em malha aberta. As duas funções de transferência em malha aberta são, respectivamente: $G_1(z)$, que é a mesma função (7) do *sistema com modelo vulnerável*; e $G_2(z)$, definida por (9):

$$G_2(z) = C_2(z)P(z) = \frac{g_{1,2}z^2 + g_{2,2}z + g_{3,2}}{z^3 + g_{4,2}z^2 + g_{5,2}z + g_{6,2}}, \quad (9)$$

onde $g_{1,2} = 0.0408$, $g_{2,2} = -0.0057$, $g_{3,2} = -0.0326$, $g_{4,2} = -2.5462$, $g_{5,2} = 2.1108$ e $g_{6,2} = -0.5646$. Note que os denominadores de $G_1(z)$ e $G_2(z)$ são iguais, visto que apenas os numeradores de $C_1(z)$ e $C_2(z)$ são diferentes. Portanto, $g_{4,1} = g_{4,2}$, $g_{5,1} = g_{5,2}$ e $g_{6,1} = g_{6,2}$.

As funções de controle $C_1(z)$ e $C_2(z)$ são projetadas de forma que os dois subsistemas deste SLS sejam estáveis. Conforme descrito na Seção 4, as funções de controle são chaveadas aleatoriamente com base na cadeia de Markov apresentada na Figura 4, sob uma regra de comutação restrita, cujas restrições são estabelecidas pela FDP apresentada na Figura 5. Os parâmetros a e b da FDP foram empiricamente ajustados para $a = 20$ e $b = 40$, a fim de atender aos Objetivos I e II, discutidos na Seção 5. Cabe ressaltar que, em relação ao Objetivo I, os parâmetros a e b foram empiricamente ajustados visando, em primeiro lugar, a estabilidade global do sistema. No entanto, o tempo de acomodação e o *overshoot* também são avaliados nestas simulações.

O ataque é implementado utilizando o BSA, tendo em vista que esta metaheurística obteve o melhor desempenho nas simulações de ataque de [de Sá et al. 2017a]. Os parâmetros do BSA são os mesmos utilizados em [de Sá et al. 2017a]: a população possui 100 indivíduos; os limites de cada dimensão do espaço de busca são $[-10, 10]$; e η – que estabelece a amplitude do deslocamento dos indivíduos do BSA – é ajustado para 1. Em cada simulação, o BSA é executado por 4500 iterações.

© 2017 Sociedade Brasileira de Computação

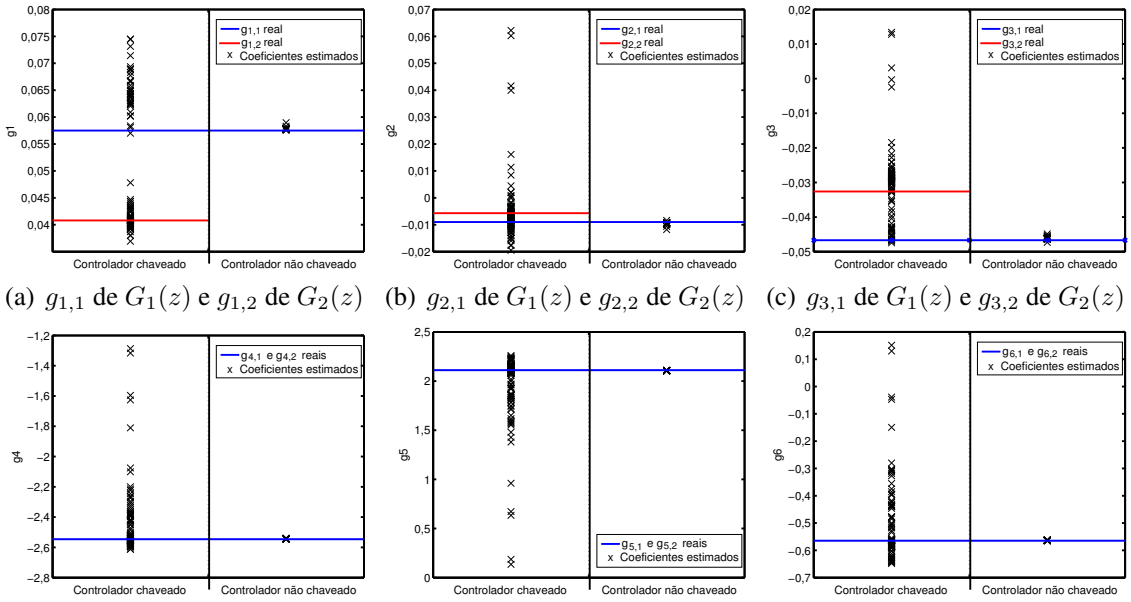
Assim como em [de Sá et al. 2017a], o sinal de ataque $a(k)$ mostrado na Figura 2 é um impulso unitário (10):

$$a(k) = \begin{cases} 1 & \text{se } k = k_a; \\ 0 & \text{se } k \neq k_a, \end{cases} \tag{10}$$

onde k_a é a única amostra em que o o atacante interfere no sistema, adicionando 1 ao sinal de realimentação. Em cada simulação, o sinal de realimentação é capturado pelo atacante durante um período $T = 2s$ (100 amostras), iniciando na amostra $k_a + 1$. Nos dois NCSs, a taxa de amostragem é de 50 amostras/s e a entrada $r(k)$ é uma degrau unitário. As simulações deste artigo não consideram perda de pacotes nem atrasos na rede.

5.2. Desempenho como Contramedida

Nesta seção, são apresentados os resultados obtidos pelo ataque Ativo de Identificação de Sistemas, quando lançado sobre os NCSs descritos na Seção 5.1 – um NCS utilizando o controlador chaveado e o outro utilizando o controlador não chaveado. Em cada NCS foram executadas 100 simulações de ataque.



(a) $g_{1,1}$ de $G_1(z)$ e $g_{1,2}$ de $G_2(z)$ (b) $g_{2,1}$ de $G_1(z)$ e $g_{2,2}$ de $G_2(z)$ (c) $g_{3,1}$ de $G_1(z)$ e $g_{3,2}$ de $G_2(z)$

(d) $g_{4,1}$ de $G_1(z)$ e $g_{4,2}$ de $G_2(z)$ (e) $g_{5,1}$ de $G_1(z)$ e $g_{5,2}$ de $G_2(z)$ (f) $g_{6,1}$ de $G_1(z)$ e $g_{6,2}$ de $G_2(z)$

Figura 6. Coeficientes estimados pelo ataque no NCS com a contramedida proposta (*i.e.* com controlador chaveado) e no NCS sem a contramedida proposta (*i.e.* com controlador não chaveado).

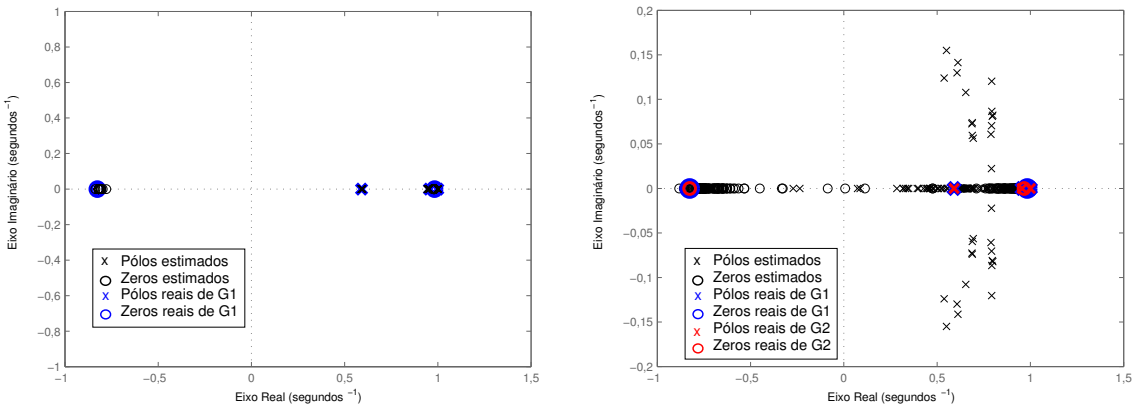
Todos os coeficientes estimados nas 100 simulações de ataque em cada NCS são apresentados na Figura 6. Cabe lembrar que o NCS que utiliza o controlador não chaveado só possui uma função de transferência em malha aberta $G_1(z)$, enquanto que o NCS com o controlador chaveado possui duas funções $G_1(z)$ and $G_2(z)$. Note que os valores reais dos coeficientes $[g_{1,1}, g_{2,1}, g_{3,1}, g_{4,1}, g_{5,1}, g_{6,1}]$ e $[g_{1,2}, g_{2,2}, g_{3,2}, g_{4,2}, g_{5,2}, g_{6,2}]$ das funções $G_1(z)$ e $G_2(z)$, respectivamente, também são representados na Figura 6. Observando as Figuras 6(a) a 6(f), é possível verificar que os coeficientes estimados no NCS com o controlador não chaveado são precisos e exatos. Neste NCS, o ataque Ativo de Identificação de Sistemas fornece a informação e a confiança que o atacante necessita para projetar ataques furtivos/dependentes de modelo. Por outro lado, no NCS dotado da contramedida proposta, o uso do controlador chaveado causa a dispersão dos valores

 © 2017 Sociedade Brasileira de Computação

estimados, reduzindo a precisão e a exatidão dos coeficientes obtidos pelo atacante. Conforme apresentado na Figura 6, os valores estimados neste SLS são difusos e não indicam, de forma acurada, nenhum dos coeficientes de $G_1(z)$ e $G_2(z)$.

O impacto do controlador chaveado na performance do ataque também pode ser verificado ao comparar os valores mínimos globais encontrados, pelo BSA, para a função de aptidão (4). No NCS dotado com o controlador chaveado, os valores mínimos globais de todas as simulações de ataque estão entre $1,81 \times 10^{-06}$ e $1,96 \times 10^{-04}$ (a média é $2,50 \times 10^{-05}$ e o desvio padrão é $3,97 \times 10^{-05}$). Por outro lado, no NCS com o controlador não chaveado, todos os valores mínimos globais estão entre $7,82 \times 10^{-09}$ e $4,46 \times 10^{-08}$ (a média é $8,75 \times 10^{-09}$ e o desvio padrão é $4,80 \times 10^{-09}$). Lembre-se que, conforme discutido na Seção 3, sem perturbações ou ruído, o valor mínimo de (4) é $\min f_j = 0$ quando o sistema atacado é identificado de forma perfeita. Dessa forma, a maior ordem dos valores mínimo globais, causada pelo uso do controlador chaveado, também demonstra a efetividade da contramedida proposta. Do ponto de vista do atacante, estes valores mínimos globais majorados podem ser um indicativo de que o Ataque Ativo de Identificação de Sistemas não foi efetivo em obter o modelo do sistema atacado. Neste caso, o atacante deve hesitar em lançar um ataque furtivo/dependente de modelo com base nas informações obtidas pelo Ataque Ativo de Identificação de Sistemas.

O impacto da contramedida no referido ataque também pode ser verificado nos diagramas de pólos e zeros da Figura 7. Na Figura 7(a), são apresentados os pólos e zeros das funções de transferência em malha aberta estimadas nas 100 simulações com o controlador não chaveado. A Figura 7(b), por sua vez, apresenta os pólos e os zeros das funções de transferência em malha aberta estimadas nas simulações utilizando o controlador chaveado. Note que, nas simulações com o controlador não chaveado, os pólos e zeros estimados convergem, de forma acurada, para os pólos e zeros da função de transferência em malha aberta $G_1(z)$. Por outro lado, a Figura 7(b) mostra que, quando a contramedida proposta é utilizada, os pólos e zeros estimados se espalham e não concorrem para os pólos e zeros reais de $G_1(z)$ e $G_2(z)$ – as funções de transferência em malha aberta dos dois subsistemas do SLS.



(a) NCS sem a contramedida proposta (b) NCS com a contramedida proposta

Figura 7. Pólos e zeros obtidos com o Ataque Ativo de Identificação de Sistemas.

O espalhamento dos pólos e zeros da Figura 7(b), a imprecisão dos coeficientes apresentados na Figura 6, e os elevados mínimos globais encontrados pelo BSA demonstram a efetividade do uso de controladores chaveados como uma contramedida para o Ataque Ativo de Identificação de Sistemas de [de Sá et al. 2017a]. Com a contramedida

© 2017 Sociedade Brasileira de Computação

proposta, é possível afirmar que o modelo obtido é impreciso/ambíguo, de tal forma que o atacante hesite em lançar contra o sistema ataques que sejam furtivos/dependentes de modelo. Assim, o Objetivo II definido na Seção 4 é atendido.

5.3. Desempenho como Controlador

Nesta seção, o desempenho da contramedida proposta é analisado do ponto de vista do controle, a fim de identificar possíveis impactos da mesma no controle da planta. Para isso, os seguintes aspectos são avaliados: estabilidade; *overshoot*; e tempo de acomodação. Com base nestes aspectos, o desempenho do controlador chaveado é comparado com o desempenho do controlador não chaveado. Tendo em vista a natureza estocástica do controlador chaveado descrito na Seção 5.1, o qual chaveia aleatoriamente entre duas funções de controle, os aspectos mencionados são avaliados por meio de um conjunto de 100.000 simulações.

Na Figura 8, são apresentadas as respostas de ambos os NCSs no domínio do tempo. As respostas do NCS dotado da contramedida proposta são representadas pela área destacada. Os limites desta área são desenhados com base nos valores máximos e mínimos da saída da planta, considerando todas as 100.000 simulações. Em outras palavras, com o controlador chaveado, todos os sinais de saída $y(k)$ gerados nas simulações estão contidos na área destacada. A resposta, não estocástica, do NCS com o controlador não chaveado é representada pela linha vermelha da Figura 8. Note que, até $t = 0.4s$ as respostas obtidas com o controlador chaveado são iguais à resposta obtida com o controlador não chaveado. Isto é causado pelo mínimo *dwell time* de $0.4s$, que corresponde à quantidade mínima de intervalos de amostragem que o sistema deve permanecer no mesmo estado (estabelecida na Seção 5.1 como $a = 20$ amostras). Com base na Figura 8, considerando todas as 100.000 simulações, é possível verificar que o NCS com a contramedida proposta é estável, a saída da planta converge para o *set point* ($r(k) = 1rad/s$) sem erro estacionário, e a planta não apresenta *overshoots*. Considerando estes aspectos, do ponto de vista do controle, a contramedida proposta apresenta o mesmo desempenho que o controlador não chaveado.

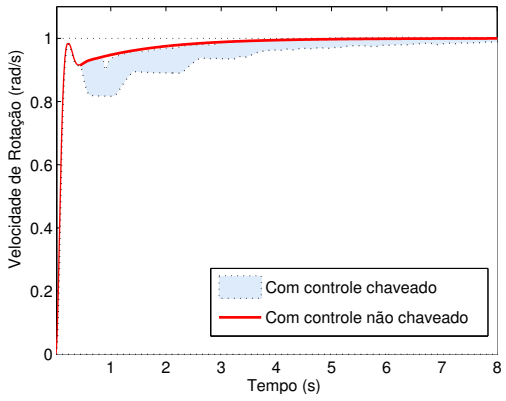


Figura 8. Resposta do motor no domínio do tempo.

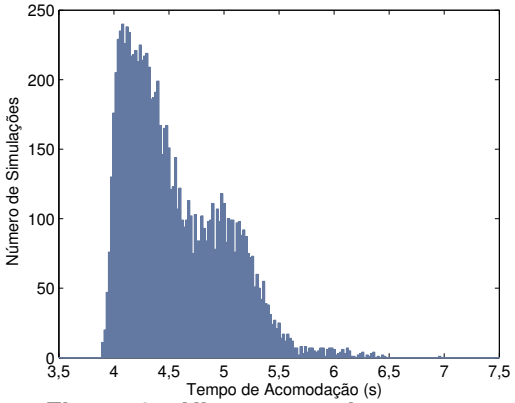


Figura 9. Histograma dos tempos de acomodação quando a contramedida proposta é utilizada.

Por outro lado, devido aos chaveamentos sucessivos, é possível verificar na Figura 8 que os tempos de acomodação obtidos com a contramedida proposta são maiores do que o tempo de acomodação obtido com o controlador não chaveado. O tempo de acomodação, determinístico, do NCS com o controlador não chaveado é $2,4s$. Já o tempo de acomodação t_s obtido com o controlador chaveado é estocástico e depende da

© 2017 Sociedade Brasileira de Computação

sequência de *dwell times* ocorridos antes de alcançar t_s, o que é aleatório. Os tempos de acomodação de todas as 100.000 simulações com o controlador chaveado são apresentados no histograma da Figura 9. Os tempos de acomodação máximo e mínimo são $3,90s$ e $6,96s$, respectivamente. A média é $4,555 \pm 0,0088s$, com intervalo de confiança de 95%.

O desempenho da contramedida proposta, na perspectiva do controle, é satisfatório e indica a viabilidade de atender aos Objetivos I e II, simultaneamente. Nestas simulações, o controle realizado pelo controlador chaveado apresenta uma performance similar à do controlador não chaveado. O requisito primário do Objetivo I – *i.e.* a estabilidade – é satisfeito, bem como o requisito de não causar *overshoot* na planta. No entanto, as simulações indicam um aumento no tempo de acomodação do sistema, o que pode não ser crítico, mas deve ser analisado dependendo do processo específico que está sendo controlado. Sendo assim, ao decidir pelo uso desta contramedida, deve-se levar em conta a relação de compromisso que existe entre mitigar o ataque de identificação e aumentar o tempo de acomodação do sistema.

6. Conclusão

Neste artigo, é proposto o uso de um controlador aleatoriamente chaveado como contramedida para o Ataque Ativo de Identificação de Sistemas, em caso de falha das contramedidas convencionais – tais como o uso de criptografia e arquiteturas de segurança de rede. É demonstrado, por meio de simulações, que esta contramedida é capaz de mitigar o ataque mencionado, tornando o modelo obtido pelo atacante impreciso/ambíguo. Ao mesmo tempo, as simulações demonstram que o desempenho da contramedida proposta é satisfatório do ponto de vista do controle. Considerando os aspectos de controle, em geral, a contramedida proposta apresenta uma performance similar à performance do controlador não chaveado, com um aumento do tempo de acomodação – o que, em muitas aplicações não representa um óbice.

Como trabalhos futuros, planejamos avaliar o desempenho desta contramedida em face de outros ataques/algoritmos de identificação de sistemas. Encorajamos também o desenvolvimento de uma heurística ou um método analítico capaz de prover funções de controle e regras de chaveamento que maximizem o desempenho da contramedida, no que concerne aos dois objetivos apresentados – *i.e.* cumprir os requisitos de controle da planta; e dificultar o processo de identificação.

Agradecimentos

Esta pesquisa foi parcialmente apoiada pelos órgãos de fomento CNPq e FAPERJ.

Referências

Amin, S., Litrico, X., Sastry, S., and Bayen, A. M. (2013). Cyber security of water scada systems part i: analysis and experimentation of stealthy deception attacks. *IEEE Transactions on Control Systems Technology*, 21(5):1963–1970.

Başţuğ, M. (2012). Recursive modeling of switched linear systems: a behavioral approach. Master's thesis, Istambul Technical University.

Civicioglu, P. (2013). Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation*, 219(15):8121–8144.

Dasgupta, S., Routh, A., Banerjee, S., Agilageswari, K., Balasubramanian, R., Bhandarkar, S., Chattopadhyay, S., Kumar, M., and Gupta, A. (2013). Networked control

© 2017 Sociedade Brasileira de Computação

of a large pressurized heavy water reactor (phwr) with discrete proportional-integral-derivative (pid) controllers. *IEEE Transactions on Nuclear Science*, 60(5):3879–3888.

de Sá, A. O., da Costa Carmo, L. F. R., and Machado, R. C. S. (2017a). Bio-inspired active attack for identification of networked control systems. In *10th EAI International Conference on Bio-inspired Information and Communications Technologies (BICT)*, pages 1–8. ACM.

de Sá, A. O., da Costa Carmo, L. F. R., and Machado, R. C. S. (2017b). Covert attacks in cyber-physical control systems. *IEEE Transactions on Industrial Informatics*, 13(4):1641–1651.

Hespanha, J. P. and Morse, A. S. (1999). Stability of switched systems with average dwell-time. In *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, volume 3, pages 2655–2660. IEEE.

Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of 1995 IEEE International Conference on Neural Networks*, pages 1942–1948.

Lin, H. and Antsaklis, P. J. (2009). Stability and stabilizability of switched linear systems: a survey of recent results. *IEEE Transactions on Automatic control*, 54(2):308–322.

Morse, A. S. (1996). Supervisory control of families of linear set-point controllers-part i. exact matching. *IEEE Transactions on Automatic Control*, 41(10):1413–1431.

Pang, Z.-H. and Liu, G.-P. (2012). Design and implementation of secure networked predictive control systems under deception attacks. *IEEE Transactions on Control Systems Technology*, 20(5):1334–1342.

Skafidas, E., Evans, R. J., Savkin, A. V., and Petersen, I. R. (1999). Stability results for switched controller systems. *Automatica*, 35(4):553–564.

Smith, R. (2011). A decoupled feedback structure for covertly appropriating networked control systems. In *Proceedings of the 18th IFAC World Congress 2011*, volume 18. IFAC-PapersOnLine.

Smith, R. S. (2015). Covert misappropriation of networked control systems: Presenting a feedback structure. *Control Systems, IEEE*, 35(1):82–92.

Stouffer, K., Pillitteri, V., Lightman, S., Abrams, M., and Hahn, A. (2015). Nist special publication 800-82, revision 2: Guide to industrial control systems (ics) security. *Gaithersburg, MD, USA: National Institute of Standards and Technology*.

Teixeira, A., Shames, I., Sandberg, H., and Johansson, K. H. (2015). A secure control framework for resource-limited adversaries. *Automatica*, 51:135–148.

Wang, J. (2013). *Identification of Switched Linear Systems*. PhD thesis, University of Alberta.

Yang, C., Guan, Z.-H., and Huang, J. (2011). Stochastic switched controller design of networked control systems with a random long delay. *Asian Journal of Control*, 13(2):255–264.

Zhai, G., Hu, B., Yasuda, K., and Michel, A. N. (2002). Qualitative analysis of discrete-time switched systems. In *American Control Conference, 2002. Proceedings of the 2002*, volume 3, pages 1880–1885. IEEE.

 © 2017 Sociedade Brasileira de Computação

Eigenface vs *Random Forest*: Um Estudo Comparativo em Reconhecimento Facial

Jardel Ribeiro de Lima[1] **e Rosalvo Fereira Oliveira Neto**[1]

[1]Colegiado de Engenharia da Computação
Universidade Federal do Vale do São Francisco (Univasf)
Caixa Postal 15.064 – 91.501-970 – Juazeiro – BA – Brasil

jardelribeiro.lima@gmail.com, rosalvo.oliveira@univasf.edu.br

Abstract. *Face recognition is a biometric technique that has some advantages when compared to other existing biometrics techniques, for example, it is not an invasive method. This technique has several specific methods used for recognition that were developed in years of research, such as Eigenface. On the other hand, Artificial Intelligence techniques have been used in face recognition and have obtained good results. This paper presents a performance comparison between these two approaches, applied to two databases from two known benchmarks. The selected techniques were Eigenface and Random Forest. The comparison was performed by measuring the error rate, training time, classification time and memory consumption on the training fase.*

Resumo. *Reconhecimento facial é uma técnica de biometria que apresenta algumas vantagens quando comparada com outras técnicas de biometria existentes. Esta técnica possui vários métodos específicos usados para reconhecimento que foram desenvolvidos em anos de pesquisa, tal como o Eigenface. Por outro lado, técnicas de Inteligência Artificial vêm sendo utilizadas para reconhecimento facial e apresentam bons resultados. Este artigo apresenta uma comparação de performance entre essas duas abordagens, aplicadas em duas base de dados de dois conhecidos benchmarks. As técnicas selecionadas foram Eigenface e Random Forest, sendo comparado os valores de taxa de erro, tempo de treinamento, tempo de classificação e memória consumida no treinamento.*

1. Introdução

Diversos sistemas fornecem aos seus usuários mecanismos como senhas e cartões magnéticos com o objetivo de identificá-los e permitir-lhes acesso a determinados recursos. No entanto, estes mecanismos não garantem que as pessoas que os possuem são realmente as pessoas autorizadas a acessar o sistema, pois esses recursos de segurança podem ser extraviados. Uma alternativa para solucionar esse problema é a utilização de técnicas de biometria para identificação e autenticação de usuários [Unar et al. 2014].

Técnicas biométricas são métodos automáticos capazes de identificar uma pessoa com base em suas características fisiológicas ou comportamentais [Wayman et al. 2005]. Esses métodos proporcionam uma maior confiabilidade na questão de autenticação de usuário, uma vez que as características físicas de uma pessoa são únicas e intransferíveis [Jain and Nandakumar 2012].

 © 2017 Sociedade Brasileira de Computação

Existem muitas aplicações que utilizam biometria como mecanismo de autenticação. Caixas eletrônicos, celulares, controle de fronteiras, casas inteligentes e eleições eletrônicas são alguns exemplos de aplicações que utilizam autenticação biométrica. O corpo humano possui várias medidas biométricas que podem ser usadas para reconhecimento e autenticação de indivíduos, entre elas podemos citar: digital, íris, mão, retina, odor do corpo, forma da orelha, face, entre outros [Bhattacharyya et al. 2009].

Reconhecimento Facial é uma técnica biométrica que tem como objetivo verificar e identificar se um indivíduo faz parte de uma base de dados de imagens de faces conhecidas [Tolba et al. 2008]. Ela possui algumas vantagens em relação às outras técnicas biométricas, dentre elas podemos destacar: não é um método invasivo; é um método que pode ser validado por uma pessoa; não está associado ao crime; os dados requeridos são obtidos facilmente [Tolba et al. 2008]. Tais vantagens e o fato de ser um método que possui muitas aplicações comerciais e de segurança fez com que Reconhecimento Facial recebesse atenção de várias pesquisas nos últimos anos [Zhao et al. 2003] [Jain and Li 2011].

O estudo na área de Reconhecimento Facial tem se intensificado nas últimas décadas, o que resultou na criação de métodos específicos para esta tarefa. Alguns métodos usam características geométricas faciais, como a distância entre olhos e a posição da boca e do nariz. Outros métodos utilizam a imagem da face como um todo para realizar o reconhecimento, que é o caso do *Eigenface* [Jain and Li 2011]. Esses métodos específicos serão chamados neste trabalho de métodos tradicionais de reconhecimento facial. Por outro lado, métodos gerais de Inteligência Artificial (IA) têm sido usados para reconhecimento facial por imagem e têm obtido bons resultados [Reddy et al. 2011]. Uma característica esperada de um sistema de reconhecimento facial é que ele possua uma baixa taxa de erro e um baixo custo computacional. Nesta perspectiva, este artigo realizou um estudo comparativo entre um método tradicional, o *Eigenface*, e um método de IA, o *Random Forest*. O Eigenface foi escolhido por ser um método tradicional simples e eficiente [Gupta et al. 2010], enquanto que o Random Forest foi selecionado pelo seu bom desempenho em relação a outros métodos de IA [Breiman 2001]. Para validar o estudo, foram utilizadas duas bases de dados de dois conhecidos *benchmarks*. A comparação foi realizada através do processo de validação cruzada repetido 3 vezes para definir os intervalos de confiança para a avaliação de desempenho, medido pela taxa de erro, tempo de classificação, tempo e memória consumida para treinamento dos modelos.

O restante do artigo está dividido como segue. A seção 2 apresenta os trabalhos relacionados. A seção 3 descreve a técnicas selecionadas. A seção 4 apresenta os materiais e métodos adotados no estudo. A seção 5 apresenta os resultados obtidos. Por fim, a seção 6 conclui o trabalho e propõe trabalhos futuros.

2. Trabalhos Relacionados

Diversos trabalhos encontrados na literatura realizaram comparação de performance entre métodos de reconhecimento facial. No entanto, de acordo com o levantamento bibliográfico realizado nesta pesquisa, nenhum comparou o *Eigenface* e o *Random Forest*. Dentre os trabalhos mais próximos podemos destacar: [Kremic and Subasi 2016], [Diniz et al. 2013] e [Delac et al. 2005].

[Kremic and Subasi 2016] realizaram um estudo comparativo de desempenho en-

 © 2017 Sociedade Brasileira de Computação

tre os classificadores *Random Forest* e *Support Vector Machine* (SVM) em reconhecimento facial. O estudo utilizou o conjunto de imagens de faces da Universidade Internacional de Burch. Esse conjunto é composto por imagens de indivíduos com diferentes poses, expressões faciais, orientações de rosto e qualidade de imagem, algumas imagens apresentam elementos externos, como óculos. Na metodologia usada por [Kremic and Subasi 2016] em cada imagem era realizada a detecção da cor da pele, com o objetivo de remover ruído. A ferramenta *Weka* foi utilizada para a execução dos métodos. A métrica de avaliação de desempenho utilizada no estudo foi a taxa de erro. De acordo com o estudo, o classificador que obteve melhor resultado foi o *Random Forest* com uma taxa de acerto de 97.10%.

[Diniz et al. 2013] compararam a performance de três métodos de IA em Reconhecimento Facial. Os métodos selecionados foram: o K-NN, *Random Forest* e o *K-Star*. O banco de imagens utilizado foi obtido pelos próprios pesquisadores e era composto por 1280 imagens de 64 pessoas em ângulos diferentes. As imagens foram redimensionadas para diferente tamanhos e transformadas para a escala de cinza. Para o K-NN, foram utilizados 3 diferentes tipos de medidas de distância para a realização da identificação: distância Euclidiana, Euclidiana Normalizada e a distância de *Manhattan*. Os parâmetros dos classificadores foram empiricamente modificados com o objetivo de encontrar os melhores resultados. Para o treinamento do classificador K-NN foram utilizadas 5 imagens para o treinamento e 15 para testes por pessoa. Nos outros classificadores, foram utilizados o software *Weka* utilizando como método de avaliação *10-fold-cross-validation*. Foi analisada a taxa de acerto apresentada por cada configuração de cada classificador a fim de determinar a melhor configuração. A melhor configuração de todos os classificadores utilizados obteve uma taxa de acerto em torno de 90%. De acordo com o estudo, o melhor classificador foi o K-NN com valor de k = 1 e usando a distância Euclidiana Normalizada, obtendo uma taxa de 91,98%.

Motivados pela falta de material em que algoritmos de reconhecimento facial fossem comparados de maneira direta e bem detalhada, [Delac et al. 2005] realizou um estudo com quatro métodos de projeção utilizadas para reconhecimento facial. Os métodos foram comparados sobre as mesmas condições de pré-processamento e implementação dos algoritmos. Foram comparados os métodos *Eigenface*, Análise Discriminante Linear (LDA), e duas variações de Análise de Componente Independente (ICA). O conjunto de imagens de faces utilizados foi FERET. Esse conjunto é dividido em galeria, utilizada para treinamento, e quatro conjuntos de imagens de teste com diferentes características. Após a construção dos subespaços pelos métodos citados, todas as imagens do conjunto foram projetadas nesses subespaços. O reconhecimento foi realizado por meio do classificador k-NN, utilizando as medidas de distância quarteirão (L1), Euclidiana (L2), distância cosseno (COS) e distância *Mahalanobis* (MAH). O *Eigenface* obteve os melhores resultados em todos conjuntos de treinamento quando utilizando a medida L1, atingindo uma taxa de acerto de 82.26 % para um dos conjuntos.

Trabalhos em que os autores propuseram uma nova abordagem, seja quanto ao método de extração de características ou ao método de reconhecimento, também realizaram comparações entre métodos de reconhecimento facial existentes e a abordagem proposta. Existem centenas de novas abordagens encontradas na literatura e descrever todas seria inviável devido a limitação de espaço do artigo, no entanto, para ilustrar este

© 2017 Sociedade Brasileira de Computação

tipo de trabalho podemos citar [Gonçalves et al. 2010].

[Gonçalves et al. 2010] propuseram um método baseado em grafos que utiliza a metodologia de Redes Complexas para a extração de características da imagem. Redes Complexas é um campo de pesquisa que combina análise de grafos e física estatística. Para comprovar a eficiência do método proposto, foi realizado um estudo comparativo entre a nova abordagem e quatro métodos tradicionais: *Eigenface*, *Fisherface*, *Laplancianface* e o *Neighborhood Preserving Embedding* (NPM). A base de dados utilizada foi o conjunto público de imagens de face ORL do Laboratório AT&T de Cambridge, que é composta por 400 imagens de 40 indivíduos. Os resultados mostram que o método proposto obteve um taxa de acerto superior aos demais métodos, 93,64%.

Algumas limitações foram observadas nos trabalhos apresentados, dentre elas podemos destacar: os trabalhos utilizaram apenas uma base de imagens nos experimentos, o que não permite fazer uma maior generalização dos resultados obtidos; o trabalho de [Diniz et al. 2013] não deixa clara a metodologia utilizada e apenas o trabalho de [Delac et al. 2005] realizou uma análise estatística para validar os resultados.

3. Técnicas Selecionadas

A seguir, serão descritos os dois métodos que foram comparados nesse estudo: o *Eigenfaces*, e o *Random Forest*.

3.1. Eigenfaces

Eigenface é um método de reconhecimento facial proposto por [Turk and Pentland 1991]. Este método usa a Análise de Componentes Principais (do Inglês Principal Component Analysis, PCA) para extrair características faciais que possuem um alto grau de variância e construir um espaço dimensional reduzido com tais características. Ele é um dos métodos mais simples e eficientes na área de reconhecimento facial [Gupta et al. 2010] e é usado como base de comparação em vários trabalhos [Heseltine et al. 2003][He et al. 2005a][He et al. 2005b] [Cai et al. 2006].

A Figura 1 mostra o fluxograma do *Eigenface*. Dado um conjunto de imagens de faces, primeiro, cada imagem que é uma matriz de pixel é transformada em um vetor de pixel, no qual cada posição representa um pixel da imagem. Em seguida, é calculada uma imagem média para que se possa construir um vetor com a variância existente entre o vetor que representa a face original e o vetor que representa a face média para cada face do conjunto [Georgescu 2011]. A partir dos vetores de variância de cada imagem de face é construída a matriz de covariância. O próximo passo é encontrar os autovetores para a matriz de covariância. No *Eigenface*, esses autovetores são chamados de eigenfaces, pois está associado ao fato de que quando os autovetores são montados na forma matricial é formado algo parecido com uma face [Gupta et al. 2010]. A Figura 2 mostra cinco eigenfaces de um conjunto de imagens de face. A matriz de covariância tem dimensões D^2xD^2, sendo D a largura ou altura da imagem em pixel. Para um conjunto de imagens de tamanho 256x256 teremos uma matriz de covariância com 65.536 autovetores, que não é computacionalmente eficiente, pois a maioria desses autovetores não são relevantes [Gupta et al. 2010]. São realizadas algumas manipulações matemáticas com matrizes e é montada uma outra matriz de covariância MxM, sendo M o número de imagens da base de imagens. São escolhidos N eigenfaces que possuem os autovalores mais elevados,

 © 2017 Sociedade Brasileira de Computação

essas N eigenfaces formarão o espaço de face [Heseltine 2005]. Uma imagem de face será representada por uma combinação linear dessas N eigenfaces. O reconhecimento no *Eigenface* será realizado através de dois passos: 1) a projeção da imagem no espaço de faces [Georgescu 2011] e 2) calculando a distância euclidiana entre essa projeção da imagem e as projeções das classes das faces conhecidas.

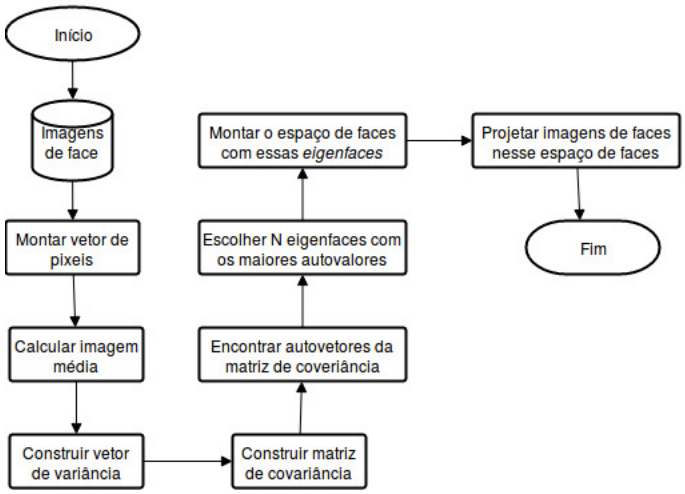


Figura 1. Fluxograma da construção do *Eigenfaces*.

Figura 2. *Eigenfaces* de um conjunto de imagens de faces [Heseltine 2005].

3.2. Random Forest

O *Random Forest* é um classificador de aprendizagem combinada (*ensemble learning*) do tipo *bagging*[1] proposto por [Breiman 2001]. Esse tipo de classificador é uma junção de vários classificadores responsáveis por gerar uma saída de forma individual que será combinada com o objetivo de apresentar uma classificação final [Kouzani et al. 2007]. O *ensemble learning* combina classificadores que isoladamente não apresentam um bom desempenho, mas quando agrupados são capazes de obter um desempenho melhor.

No contexto do *ensemble learning*, *Random Forest* é um conjunto de árvores de decisão que compõe uma floresta. Cada árvore da floresta será construída independentemente de acordo com o fluxograma da Figura 3.

Para cada árvore é obtido uma amostra *bootstrap*[2] do conjunto de treinamento, para cada nó da árvore m variáveis são aleatóriamente selecionadas. Em seguinda, é

[1]Combinação de **B**ootstrap com **agg**regat**ing** proposto por [Breiman 1996].
[2]Amostras com repetição e de mesmo tamanho [Efron 1979].

© 2017 Sociedade Brasileira de Computação

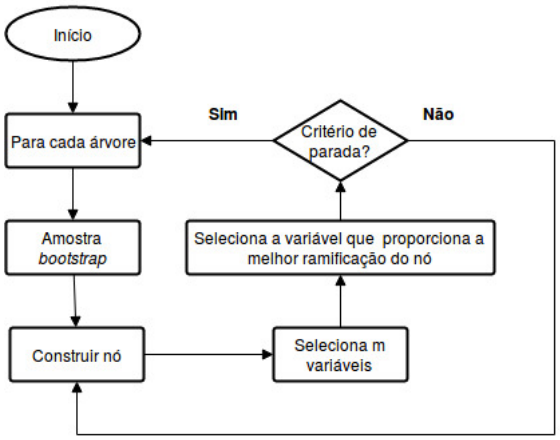


Figura 3. Fluxograma da construção do *Random Forest*.

usada, entre essas m variávies, a que proporciona a melhor ramificação do nó, isso será feito até que a árvore seja completamente construída. Esse processo se repete até que todas as árvores da floresta sejam construídas. Na fase de classificação, cada árvore receberá o dado a ser classificado e votará em qual classe esse dado pertence, como cada árvore poderá gerar uma saída diferente, a floresta escolherá a classe com maior quantidade de votos (*aggregating*) [Khan et al. 2010].

De acordo com [Breiman 2001], o *Random Forest* tem um bom desempenho quando comparado com outros classificadores tradicionais, como o SVM e redes neurais. Uma das principais vantagens do *Random Forest* é sua facilidade de uso, uma vez que só possui dois parâmetros: número de árvores da floresta e número de atributos escolhidos aleatoriamente para determinar a melhor ramificação dos nós das árvores [Liaw and Wiener 2002]. [Belle 2008] pontua mais algumas vantagens do *Random Forest*: aprendizado direto; representação local; classificação com oclusão; paralelismo e rápido tempo de treinamento.

4. Materiais e Métodos

Essa seção descreve os materiais e métodos que foram utilizados paro o desenvolvimento do presente trabalho. Como o principal objetivo desta pesquisa foi realizar um estudo comparativo entre as técnicias selecionadas de reconhecimento facial, o formalismo apresentado nesta seção é crucial para a relevância do artigo.

4.1. Bases de Imagens

Foram utilizadas duas bases de dados. A primeira base de imagens de face escolhida foi a ORL do Laboratório AT&T de Cambridge [Cambridge 2002]. Ela é uma base simples onde as imagens estão em escala de cinza e o rosto dos indivíduos estão bem enquadrados. Essa base é composta por 400 imagens de 40 indivíduos, tendo 10 fotos por indivíduo. Algumas imagens de indivíduos possuem pequenas variações de iluminação, diferentes expressões faciais (olhos fechados/abertos, sorrindo/sério), detalhes faciais (usando/não usando óculos) e em alguns casos as imagens de um indivíduo foram obtidas em diferentes datas. A Figura 4 é uma pequena amostras do base ORL.

 © 2017 Sociedade Brasileira de Computação

Figura 4. Amostra de imagens da base ORL [Cambridge 2002].

A segunda base de imagens de face escolhida foi da Universidade de Yale [Yale 1997]. Ela é composta por 165 imagens em escala de cinza de 15 indivíduos. São 11 imagens por indivíduo com variações de iluminação, expressão facial e usando ou não óculos. A Figura 5 é uma amostra da base de Yale. Os criadores da base disponibilizaram alguns tratamentos das imagens, por exemplo, existe um tratamento em que todas as imagens tiveram as faces centralizadas em imagens de 195x231 píxeis, essa foi a variação usada neste experimento.

Figura 5. Amostra de imagens da base de Yale [El Aroussi 2009].

4.2. Métricas de Avaliação de Desempenho

A comparação entre os métodos utilizou as seguintes métricas de avaliação de desempenho: 1) a taxa de erro [3] para medir a acurácia, 2) tempo para treinamento do modelo, 3) tempo para classificar uma nova face e 4) consumo de memória durante o treinamento do modelo. Essas três últimas foram selecionadas para comparar a eficiência computacional entre os dois métodos.

4.3. Método de Avaliação de Desempenho

O método de Validação Cruzada *k-fold* é uma forma amplamente aceita para dividir uma única amostra [Jain and Mao 2000] em k conjuntos de testes estatisticamente independentes, permitindo a construção de intervalos de confiança para a medida de desempenho utilizada como critério de avaliação. Para o estudo foi utilizado o método de Validação Cruzada com k = 5. Cada base de dados foi divida em 5 conjuntos (*folds*) de mesmo tamanho, sendo 4 *folds* usados para treinamento e um *fold* usado para teste. O treinamento foi repetido até que todos os *folds* fossem usados para teste. O método Validação Cruzada *k-fold* foi repetido 3 vezes para obtenção de uma amostra maior para construção do teste de hipóteses. A Figura 6 representa a divisão de uma base imagens em 5 *folds*.

[3] Razão entre a quantidade de faces incorretamente classificadas e a quantidade total de faces classificadas.

© 2017 Sociedade Brasileira de Computação

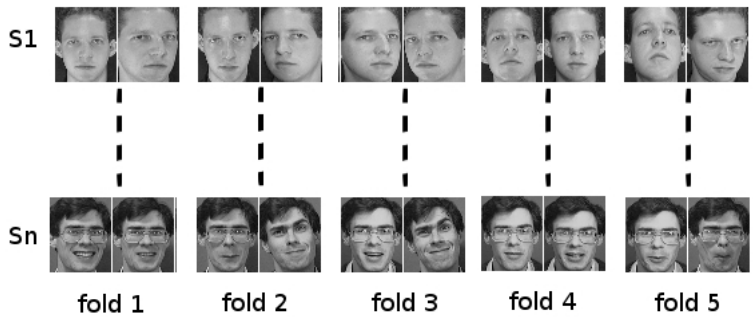


Figura 6. Representação da separação de uma base de imagens usado *5-fold-cross-validation*. Para cada indivíduo da base de imagens (S1 a Sn) 2 imagens serão escolhidas para formar cada fold

4.4. Teste *t-Student* emparelhado

O teste *t-Student* emparelhado é um caso especial de teste de hipóteses que se aplica quando as observações nas duas populações de interesse são coletadas em pares, tendo cada par de observações tomado sob condições homogêneas [Montgomery and Runger 2010]. Para este estudo, as métrica de desempenho de interesse foram a diferença na média da taxa de erro, consumo de memória no treinamento, tempo de treinamento e tempo de renconhecimento obtido por cada uma das técnicas quando avaliadas para cada conjunto de teste. A configuração do teste utilizada neste estudo está detalhada abaixo.

- **Hipótese nula**: $\mu_1 - \mu_2 = 0$
- **Hipótese alternativa**: $\mu_1 < \mu_2$

Onde

- μ_1 representa a média da métrica que obteve o menor resultado;
- μ_2 representa a média da métrica que obteve o maior resultado.

4.5. Configuração Experimental

Os métodos comparados neste estudo foram executados a partir de bibliotecas gratuitas e de código aberto. O *Eigenface* foi utilizado por meio da biblioteca *OpenCV*[4] versão 3.0. O *Random Forest* foi executado por meio da biblioteca *Python scikit-learn*[5] versão 0.18.1. As medidas de consumo de memória dos treinamentos dos modelos foram obtidas pela biblioteca *Python memory_profiler*[6] versão 0.47, as medidas de memória são dadas em MiB (1 MiB $\cong$ 1,05 MB). Para evitar problemas causados por incompatibilidade de *software* e diferentes configurações de *hardware*, todos os experimentos foram executados na mesma máquina. Um computador Intel Core i3 CPU M 350 2.27 GHz, com 2,9 GiB de memória RAM, com o sistema operacional Ubuntu 16.04 de 32 bits e HD de 500 GB e 5400 rpm.

Os parâmetros dos métodos estudados influenciam em sua performance, por isso foram escolhidos parâmetros baseados em recomendações encontradas na literatura para

[4]http://opencv.org/
[5]http://scikit-learn.org
[6]https://pypi.python.org/pypi/memory_profiler

 © 2017 Sociedade Brasileira de Computação

cada método nas duas bases de dados. As recomentações utilizadas foram: [Yang 2002] para o *Eigenface* na base de ORL e na base de Yale. [Kouzani et al. 2007] para o *Random Forest* na base ORL. Para o *Random Forest* na base de Yale não foram encontradas recomendações, por isso foi realizado um estudo preliminar para definição destes parâmetros. A Tabela 1 mostra os parâmetros utilizado em cada método para as duas bases de dados. O código fonte necessário para execução dos experimentos realizados neste estudo está diponível em *https://github.com/jardel-lima/FaceRecognitionExperiment*. A Figura 7 exibe um resumo da metodologia experimental adotada no estudo.

Tabela 1. Valores dos parâmetros dos métodos utilizados.

	Base	
Método	ORL	Yale
Eigenface	40 eigenfaces	30 eigenfaces
Random Forest	255 árvores 9 variáveis aleatórias	100 árvores 212 variáveis aleatórias

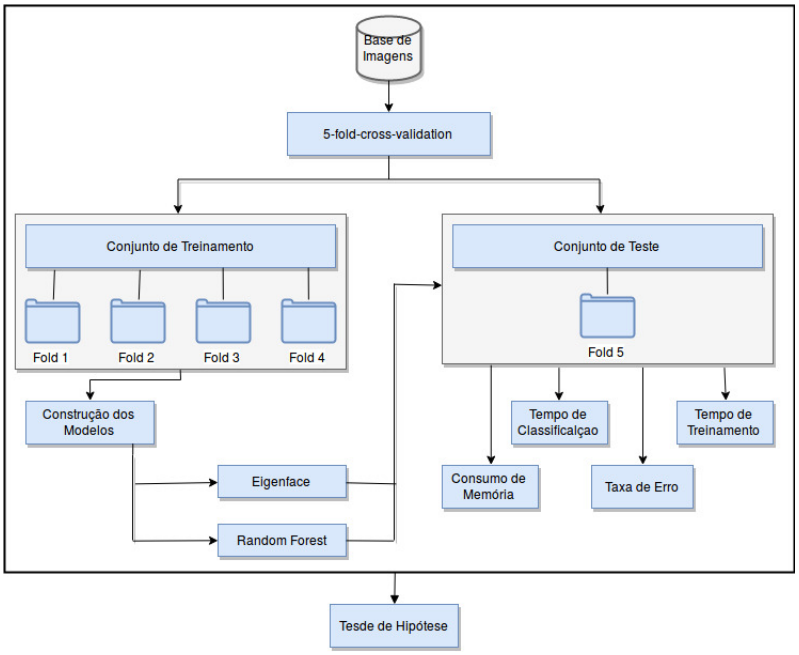


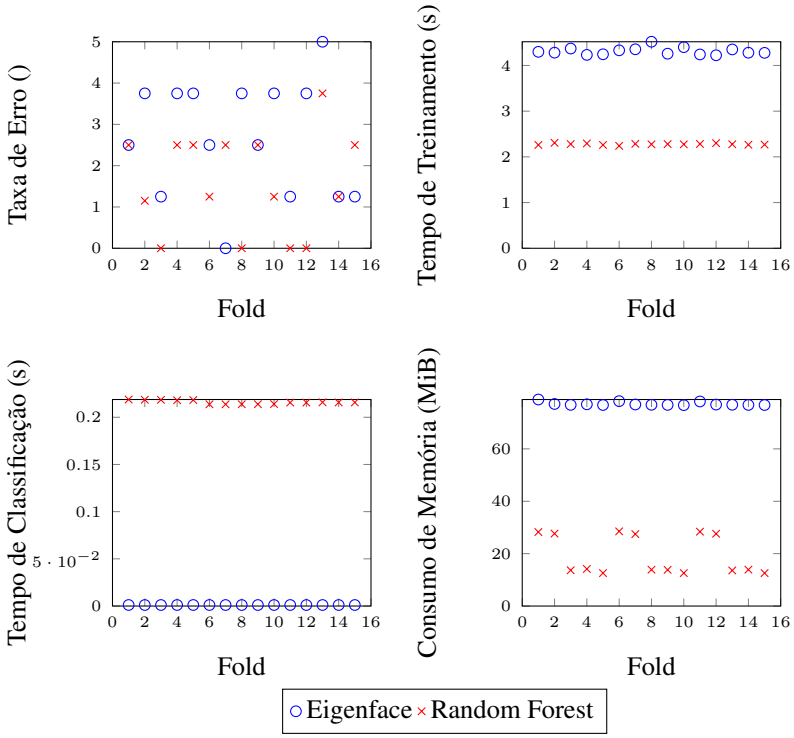
Figura 7. Procedimento para construção e treinamento dos modelos usando o *Eigenface* e o *Random Forest*.

5. Resultados

As simulações foram realizadas de acordo com a configuração experimental descrita anteriormente para cada base de dados. A Figura 8 ilustra os gráficos com as métricas de avaliação de desempenho para a base de dados ORL. Como pode ser observado, o *Random Forest* apresentou um maior poder preditivo, pois sua taxa de erro foi sistematicamente inferior a taxa de erro apresentada pelo *Eigenface*. Os gráficos mostram também que o *Random Forest* possui um custo computacional menor do que o *Eigenface* para construção da solução, pois apresentou um menor tempo para o treinamento do modelo

© 2017 Sociedade Brasileira de Computação

Figura 8. Resultados do experimento *Eigenface* vs *Random Forest* com a base ORL.



e um menor consumo de memória no treinamento. Por outro lado, o *Eigenface* foi mais rápido para classificar uma imagem após o modelo ser construído.

A Tabela 2 exibe o resumo dos resultados obtidos no teste *t-Student* emparelhado com a base ORL. O *Random Forest* apresentou uma taxa de erro 40,06% menor, foi 47,19% mais rápido para construção do modelo e consumiu 75,06% menos memória. No entanto, o *Eigenface* foi 21400% mais rápido para classificar uma imagem após a construção da solução. Uma vez que o *p-value* é menor que 0.05 para todas as métricas, os dados indicam que o *Random Forest* possui um maior poder preditivo, um menor custo computacional para construção da solução, e um maior custo computacional para classificação de uma nova imagem do que o *Eigenface* com um nível de confiança de 95%.

Tabela 2. Resultado do teste *t-Student* emparelhado com os dados do experimento com a base ORL.

Métrica	μ_1	μ_2	μ_1 - μ_2	***p-value***
Taxa de erro (%)	1,583	2,666	-1,083	0.01334
Tempo de treinamento (s)	2,276	4,310	-2,034	2.2e-16
Consumo de memória para treinamento (MiB)	19,239	77,155	-57,916	7.7e-15
Tempo de classificação (s)	0,001	0,215	-0,214	2.2e-16

A Figura 9 ilustra os gráficos com as métricas de avaliação de desempenho para a base de dados Yale. Como pode ser observado, os gráficos corroboram com os resultados

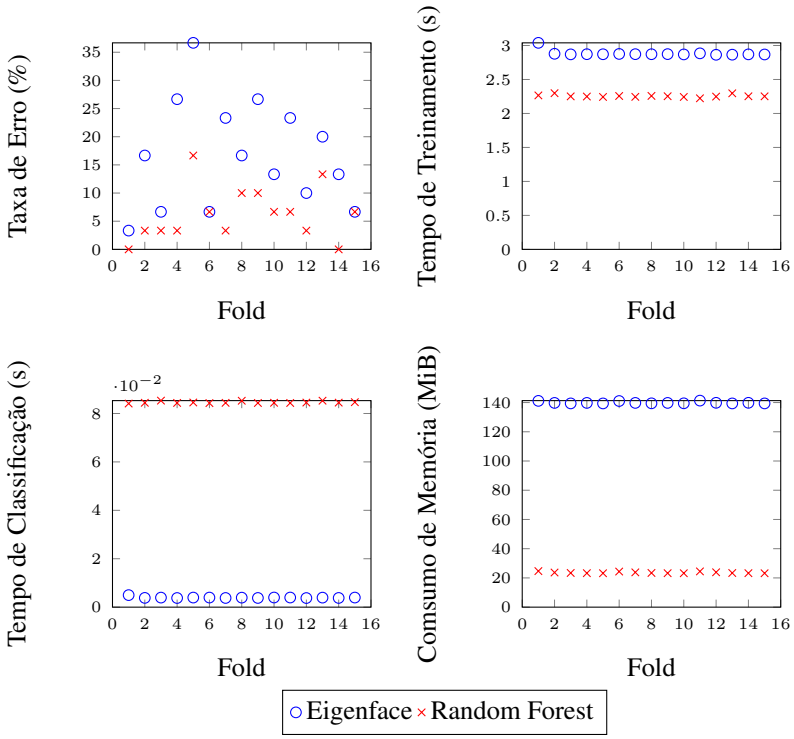
 © 2017 Sociedade Brasileira de Computação

Tabela 3. Resultado do teste *t-Student* emparelhado com os dados do experimento com a base de Yale.

Métrica	μ_1	μ_2	μ_1 - $_2$	***p-value***
Taxa de erro (%)	6,222	16,666	-10,444	5.733e-05
Tempo de treinamento (s)	2,225	2,882	-0,657	2.2e-16
Consumo de memória para treinamento (MiB)	23,623	139,91	-116,287	2.2e-16
Tempo de classificação (s)	0,003	0,084	-0,081	2.2e-16

obtidos com a base de dados ORL. A Tabela 3 exibe o resumo dos resultados obtidos no teste *t-Student* emparelhado com a base Yale. O *Random Forest* apresentou uma taxa de erro 62,67% menor, foi 22,8% mais rápido para construção do modelo e consumiu 83,12% menos memória. No entanto, o *Eigenface* foi 2700% mais rápido para classificar uma imagem após a construção da solução. Uma vez que o *p-value* é menor que 0.05 para todas as métricas, os dados indicam que o *Random Forest* possui um maior poder preditivo, um menor custo computacional para construção da solução, e um maior custo computacional para classificação de uma nova imagem do que o *Eigenface* com um nível de confiança de 95%.

Figura 9. Resultados do experimento *Eigenface* vs *Random Forest* com a base YALE.



6. Conclusões

Este trabalho apresentou uma comparação entre duas abordagens para Reconhecimento Facial, *Eigenface* e *Random Forest*. A comparação foi realizada utilizando dois bancos de dados considerados *benchmarks* para a área. Como metodologia experimental, foi

© 2017 Sociedade Brasileira de Computação

aplicado o teste *t-Student* emparelhado unicaudal sobre o desempenho medido por quatro métricas de avaliação de desempenho nos conjuntos de teste gerados pelo processo de validação cruzada 5 *fold* repetido 3 vezes. O estudo mostrou que o *Random Forest* supera de forma estatisticamente significativa o *Eigenface* em poder preditivo e custo de treinamento com um nível de confiança de 95%. No entanto, apresentou um maior tempo para classificação após a construção do modelo.

Os resultados obtidos corroboram com resultados recentes que mostram que o *Random Forest* é bem adequado para problemas com alta-dimensionalidade e com amostras pequenas, que foi a configuração das bases de dados utilizadas neste estudo. O maior poder preditivo do *Random Forest* pode ser justificado por sua estratégia de criar *ensembles* de árvores de decisão. Outro resultado importante deste estudo, é que ele mostra que soluções de Reconhecimento Facial com o *Random Forest* apresentam um custo computacional competitivo e que podem ser utilizadas em equipamentos com um menor poder computacional.

Como trabalhos futuros, este estudo será ampliado para que o equipamento de execução dos experimentos seja um *smartphone* de preço acessível, que em geral, possui uma configuração menor. Desta forma, será possível verificar se os resultados serão mantidos. Além disso, considerar mais dois métodos de Inteligência Artificial e mais uma base de dados.

Referências

Belle, V. (2008). Detection and recognition of human faces using random forests for a mobile robot. *Master of Science Thesis, Academic Knowledge-based Systems Group*.

Bhattacharyya, D., Ranjan, R., Alisherov, F., Choi, M., et al. (2009). Biometric authentication: A review. *International Journal of u-and e-Service, Science and Technology*, 2(3):13–28.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Cai, D., He, X., Han, J., and Zhang, H.-J. (2006). Orthogonal laplacianfaces for face recognition. *IEEE transactions on image processing*, 15(11):3608–3614.

Cambridge, A. L. (2002). The database of faces. Acesso em: 02 mar. 2017.

Delac, K., Grgic, M., and Grgic, S. (2005). Independent comparative study of pca, ica, and lda on the feret data set. *International Journal of Imaging Systems and Technology*, 15(5):252–260.

Diniz, F. A., Neto, F. M. M., Júnior, F. d. C. L., and Fontes, L. M. O. (2013). Redface: um sistema de reconhecimento facial baseado em técnicas de análise de componentes principais e autofaces. *Revista Brasileira de Computação Aplicada*, 5(1):42–54.

Efron, B. (1979). Bootstrap methods: another look at the jackknife. *The annals of Statistics*, pages 1–26.

El Aroussi, M. (2009). Information fusion towards a robust face recognition system.

Georgescu, D. (2011). A real-time face recognition system using eigenfaces. *Journal of Mobile, Embedded and Distributed Systems*, 3(4):193–204.

 © 2017 Sociedade Brasileira de Computação

Gonçalves, W. N., de Andrade Silva, J., and Bruno, O. M. (2010). A rotation invariant face recognition method based on complex network. In *Iberoamerican Congress on Pattern Recognition*, pages 426–433. Springer.

Gupta, S., Sahoo, O., Goel, A., and Gupta, R. (2010). A new optimized approach to face recognition using eigenfaces. *Global Journal of Computer Science and Technology*, 10(1).

He, X., Cai, D., Yan, S., and Zhang, H.-J. (2005a). Neighborhood preserving embedding. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1208–1213. IEEE.

He, X., Yan, S., Hu, Y., Niyogi, P., and Zhang, H.-J. (2005b). Face recognition using laplacianfaces. *IEEE transactions on pattern analysis and machine intelligence*, 27(3):328–340.

Heseltine, T., Pears, N., Austin, J., and Chen, Z. (2003). Face recognition: A comparison of appearance-based approaches. In *Proc. VIIth Digital image computing: Techniques and applications*, volume 1.

Heseltine, T. D. (2005). *Face recognition: Two-dimensional and three-dimensional techniques*. PhD thesis, University of York.

Jain, A. K. and Li, S. Z. (2011). *Handbook of face recognition*. Springer.

Jain, A. K. and Nandakumar, K. (2012). Biometric authentication: System security and user privacy. *IEEE Computer*, 45(11):87–92.

Khan, R., Hanbury, A., and Stoettinger, J. (2010). Skin detection: A random forest approach. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 4613–4616. IEEE.

Kouzani, A., Nahavandi, S., and Khoshmanesh, K. (2007). Face classification by a random forest. In *TENCON 2007-2007 IEEE Region 10 Conference*, pages 1–4. IEEE.

Kremic, E. and Subasi, A. (2016). Performance of random forest and svm in face recognition. *Int. Arab J. Inf. Technol.*, 13(2):287–293.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

Montgomery, D. C. and Runger, G. C. (2010). *Applied statistics and probability for engineers*. John Wiley & Sons.

Reddy, N. V., Krishna, D. A., Reddy, P. S., and Shirisha, R. (2011). Neural network based intelligent local fac e recognition using local pattern averaging. In *2011 3rd International Conference on Electronics Computer Technology*, pages 363–367.

Tolba, A., El-Baz, A., and El-Harby, A. (2008). Face recognition: A literature review.

Turk, M. A. and Pentland, A. P. (1991). Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE.

Unar, J., Seng, W. C., and Abbasi, A. (2014). A review of biometric technology along with trends and prospects. *Pattern recognition*, 47(8):2673–2688.

© 2017 Sociedade Brasileira de Computação

Wayman, J., Jain, A., Maltoni, D., and Maio, D. (2005). *An introduction to biometric authentication systems*. Springer.

Yale, U. (1997). The yale face database. Acesso em: 02 jun. 2017.

Yang, M.-H. (2002). Kernel eigenfaces vs. kernel fisherfaces: Face recognition using kernel methods. In *Fgr*, volume 2, page 215.

Zhao, W., Chellappa, R., Phillips, P. J., and Rosenfeld, A. (2003). Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458.

 © 2017 Sociedade Brasileira de Computação

The performance of error correcting codes in discrete and continuous fuzzy pairing protocols

Henrique de Rocha Deus[1], Vladimir Portela Parente[2], Jeroen van de Graaf[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

[2]Graduate Program in Electrical Engineering
Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

{henriquerdd,jvdg}@dcc.ufmg.br, vladimir@cpdee.ufmg.br

Abstract. *In fuzzy pairing, two parties compare two bit strings which are supposed to be similar, but almost never identical. So A and B engage in a protocol to verify whether $d(s_A, s_B)$ is less than some threshold T; if not, the parties abort and authentication failed. Here $d()$ is to be taken as the Hamming distance. One standard protocol is the code-offset method: A computes a random vector x such that $s_A - x$ is a code word of some pre-agreed error-correcting code and sends x to B, who decodes $s_B - x$. Together they verify whether the two decoded codewords are the same. A common secret key can be obtained subsequently,*
We cast this problem in a different context, in which A and B want to compare continuous *signals, instead of discrete bit strings. We test the code offset method for four classes of error-correcting codes: Reed-Solomon (RS) Codes, Low Density Parity Check (LDPC) Codes, Repeat-Accumulate Codes (RAC) and Low Density Lattice Codes (LDLC).*
For similar error correction capability our results show that RS codes perform slowly, while LDPC and RAC which very similar are both really fast. LDLC has the best correction capabilities, but are slower because of their mathematical complexity. Our results can be generalized to fuzzy extractors.

1. Introduction

1.1. Fuzzy pairing with bit strings

Consider the following setting: Alice has a string s_A while Bob has a string s_B. Alice and Bob know that their strings are similar, but probably not identical. In addition, they know or suspect that the adversary Eve has some information about their bit string, but up to some known limit. In addition we assume that Alice and Bob can send each other authenticated messages, to which Eve also has access. Now consider the following cryptographic functionalities:

Fuzzy mutual authentication: A and B want to verify whether $d(s_A, s_B)$ is less than some threshold T, where $d()$ is to be taken as the Hamming distance. If true, authentication succeeds; if not, the parties abort and this authentication attempt failed.

 © 2017 Sociedade Brasileira de Computação

Fuzzy key establishment: Alice and Bob want to obtain two smaller strings s'_A and s'_B such that $s'_A = s'_B$ with overwhelming probability.
This problem is otherwise known as key reconciliation or information reconciliation, and seems to have occurred first in the context of quantum cryptography: the bit strings are the result of those positions in which Alice and Bob used the same basis when transmitting resp. measuring the quantum state.
Note also that after reconciliation succeeded, Eve still might have information about the common string $s' = s'_A = s'_B$, so it is typically followed by another protocol known as privacy amplification. See [Bennett et al. 1995] for details.

Fuzzy pairing: In fact, pairing is the combination of the two preceding functionalities: the two parties decide to trust each other, and agree on some symmetric key to establish a secure communication channel.

Note that there exist a simple protocol to verify equality at the sacrifice of one bit per round: Alice choose random $r \in \mathbb{Z}_2^m$ and computes the parity (xor sum) $p_A = par(s'_A \wedge r)$, and sends the pair (r, p_A) to Bob. Bob computes $p_B = par(s'_B \wedge r)$. It is easy to see that if $s'_A = s'_B$ then $p_A = p_B$ always, whereas if $s'_A \neq s'_B$ then $p_A \neq p_B$ with probability $\frac{1}{2}$. So this protocol, which we call EQUAL must be repeated a sufficient number of times.

1.2. The code-offset method

The more famous reconciliation protocols for bitstrings, such as [Brassard and Salvail 1993], rely heavily on interaction. But a simpler approach can be based on error correcting codes. Later known as the *code-offset method*, $\mathcal{A}$ computes a random vector $x \in \mathbb{Z}_2^n$ such that $s'_A - x$ belongs to $C \subseteq \mathbb{Z}_2^n$, some pre-agreed error-correcting code, and sends x to B, who decodes $s'_B - x$. Then the parties verify whether the decoded codewords are equal, using the EQUAL protocol outlined above. If C is a linear code, an equivalent result is obtained if A sends the syndrome of s'_A to B.

1.3. Fuzzy pairing with signals

Traditionally, the information to be reconciled is represented as bits. In this paper we discuss a different setting: the information shared between A and B are signals that are similar, but not identical. In principle these signals can be of any kind: audio, wifi signal strength, accelerometer data, biometric data etc.

To model this setting adequately we need a continuous communication model. The natural choice is the AWGN (Additive White Gaussian Noise) channel, which is essentially as follows: imagine two Gaussian probability distributions: $N_0 = N(-1, \sigma)$ with center 0 and $N_1 = N(-1, \sigma)$ with center 1. To send a bit v, A produces a signal according to N_v; this signal is received by B as w. B's task is to guess v given w. Obviously, B's error depends on σ. We also need to substitute the Hamming distance for the Euclidean distance in $\mathbb{R}^n$.

In order to reconcile information in this continuous context, the logical step is to apply the code-offset method for an error-correcting code suitable for this context. In fact, no other approach seems possible since all the interactive protocols do not seem to carry over to this new setting.

 © 2017 Sociedade Brasileira de Computação

1.4. Our research

It turns out that in all realistic scenarios for fuzzy pairing, the main performance bottleneck is decoding time, since it translates directly in the time the users have to wait to know if the pairing process succeeded or not.

In this paper we analyse the decoding performance in the signal (i.e. continuous) setting with AWGN channel for four classes of error correcting codes: Reed-Solomon (RS) Codes, Low Density Parity Check (LDPC) Codes, Repeat-Accumulate Codes (RAC) and Low Density Lattice Codes (LDLC).

This choice of these codes is natural: RS are very popular and well-known. Later LDPC codes were developed with superior performance; RAC were included because they are a dramatic simplification of LDPC with respect to encoding, while using essentially the same decoding algorithm. Finally, in theory LDLC will correct more than the others, however due to its complexity it is expected to be much slower. In this work we want to study how big that difference in both correction and speed is in practice.

1.5. Comparison with other work

Many existing pairing protocols, including Bluetooth, are based on the Diffie-Hellman Key Exchange. This has two disadvantages. First, it is based on computational assumptions. Second, to exclude a man-in-the-middle attack some user intervention is necessary. By using the newest features of smart phones, this comparison task can be performed by one of the devices. The hash value is no longer communicated or compared by a human, but through some alternative, reliable channel with the help of a human. Possibilities are:

- One display shows a code, user enters it in the keyboard of the other device.
- Printer prints or displays a bar code; a camera takes a photo.
- One smart phone plays a random melody; the other records the sound.

See [Kumar et al. 2009] for an overview. Observe that all these options require user action.

In our fuzzy pairing protocol no human intervention is necessary. Also it is not based on a computational assumption but on physical proximity: we assume that $\mathcal{A}$ and $\mathcal{B}$ have access to some similar signal, whereas the signal that Eve can obtain is not sufficiently similar.

1.6. Paper outline

The next section present a broad overview of fuzzy cryptography, providing many additional references to reconciliation and fuzzy crypto protocols. In section 3 we outline our protocol for the audio setting, and discuss the literature on other approaches. Section 4 discusses the classes of codes we tested. In section 5 we present our simulation and results, while Section 6 contains the conclusion.

2. Fuzzy Mutual Authentication

Fuzzy cryptography can be described as the area in which legitimate parties have similar but not identical information. Historically this problem has been approached in two different ways: one is based on information reconciliation, the other is called fuzzy cryptography.

© 2017 Sociedade Brasileira de Computação

2.1. Reconciliation protocols

A great variety of quantum key distribution protocols make the legitimate parties to have keys with little difference because of eavesdroppers and imperfections in the transmission line and detectors. Therefore, we can see this problem as Alice transmitting the key through a binary symmetric channel for Bob. In order to Alice and Bob have the same key, an error correction has to be done.

[Brassard and Salvail 1994] describes the Cascade protocol. It takes two correlated strings from Alice and Bob, divide them in blocks and, in each round, sends the parity of a block. Essentially, a binary search is done in order to find a different bit. However, because the parity of some blocks are revealed, privacy amplification has to be executed after the protocol to reduce the adversary's knowledge. This is one of the first works to address the key exchange problem when the legitimate parties have some correlated knowledge.

[Linnartz and Tuyls 2003] proposed an authentication scheme that can also be slightly modified to create a key conciliation protocol. Additional references are [Buhan et al. 2007a][Chang et al. 2004][Zheng et al. 2006][Verbitskiy et al. 2010].

2.2. Fuzzy cryptography constructions

[Juels and Sudan 2006] proposes a new construction called fuzzy vault. If Alice has a secret k and a set A, she does a commitment with the secret k. Bob can only perform a decommitment if he has a set B close enough to A. A practical protocol using Reed-Solomon codes (described in [Reed and Solomon 1960]) is showed.

In [Dodis et al. 2008] two important constructions are showed. The secure sketch construction can be seen as an authentication scheme. One side is the user and the other is the database with the user information. Authentication is only possible if the user has a feature that is close enough by some metric to the information that the database has. The other construction is the fuzzy extractor. It allows that Alice and Bob distill a secret and common key if they both have information with enough correlation.

[Buhan et al. 2007b] extends the definition of fuzzy extractor to continuous sources. It shows that it is possible to see the schemes that work with continuous source as authentication systems with some False Acceptance Rate (FAR) and some False Rejection Rate (FRR) with a public distribution of the feature of the users (global distribution) and private distributions of the users (each user has his own).

In [Verbitskiy et al. 2010] the results regarding the cs-fuzzy extractor (fuzzy extractor for continuous source) are extended and some limits of security are calculated. It is important to highlight that has been discussed that no universal fuzzy extractor for continuous source can exist. It is necessary to make assumptions about the distribution used.

3. Fuzzy pairing based on audio

We illustrate the context with a specific scenario in which we use audio as the common signal between two devices (smartphones) A and B. This was indeed the original context of our research, after we discarded accelerometer data for having insufficient entropy. We make the following assumptions for our protocol:

 © 2017 Sociedade Brasileira de Computação

(a) Each device has a microphone.
(b) Each device has processing capacity.
(c) The devices dispose of a wireless, unauthenticated communication channel. An adversary can listen to this channel, but cannot modify messages sent by A or B.
(d) The two microphones are physically close to each other during the recording phase in order to have very similar sound recordings. We call this the *private sound zone*.
(e) An adversary cannot enter the private sound zone of the two legitimate parties without being detected, and if he tries, A and B will simply not execute the protocol. Therefore the adversary cannot obtain information about the audio recording.

Fuzzy pairing protocol:

1. A and B share the same sound environment and are synchronized in some way not discussed here. They record the sound.
2. Each party submits the collected audio data to an identical set of transformations resulting in two singular vectors s_A and s_B.
3. $\mathcal{A}$ computes a random vector $x \in \mathbb{Z}_2^n$ such that $s_A - x$ belongs to $C \subseteq \mathbb{Z}_2^n$, some pre-agreed error-correcting code, and sends x to B.
4. A decodes $s_A - x$ to obtain s'_A while B decodes $s_B - x$ to obtain s'_B. This is the most time-consuming protocol step. Observe that if $d(s_A, s_B) < d_{min}(C)$ then $s'_A = s'_B$.
5. A and B both perform privacy amplification, as follows. A compute $s''_A = h(s'_A)$, where $h()$ is a universal hash function. B applies h to his value.
6. A and B use the EQUAL protocol to test whether $s''_A = s''_B$. If true, authentication succeeded and s'' is there common shared secret key. If not, authentication has failed and they abort

3.1. Other authentication protocols using audio

[Schurmann and Sigg 2013] proposed and built an application using a fuzzy extractor scheme. Two computers, each one with it own microphone, agree on a common time to record. The audio data was processed in order to generate a fingerprint of 512 bits. A secret random string of 204 bits was generated and encoded to 512 bits using a Reed-Solomon code, specifically $RS(2^{10}, 204, 512)$. This value was added to the audio fingerprint and then transmitted to the other legitimate part.

One positive aspect of this work is the amount of tests that had been done. With the analysis of four environments and varying the distance from the microphones, 7500 tests were done. This work has the same goal and uses the same source to extract features. Furthermore, the audio processing uses STFT to calculate the energy difference between frames. Although this approach works with the discrete Hamming distance, the STFT processing is an interesting choice to use with audio.

[Goodrich et al. 2006] creates a system called Loud and Clear (L&C). It users that fact that every device, such as a smartphone, has at least one user. A sound is produced at a device and the user needs to verify if the sentence recognized is the same. From this, one-way authentication is possible (two-way is just another execution of the protocol). Although this protocol uses sound in order to authenticate the devices, it ignores the sound context of the surrounding of both devices. Furthermore, the improvement suggested in

© 2017 Sociedade Brasileira de Computação

the paper uses the Diffie-Hellman key exchange protocol. Summarizing, it returns to the computational security in an unnecessary way. Albeit the protocol is interesting, it is not suitable for actual use.

4. Fuzzy authentication using other phenomena

4.1. Authentication with Acceleration

[Kirovski et al. 2007a] and [Kirovski et al. 2007b] proposed a key conciliation protocol based on acceleration. Before doing the readings, it is necessary to pair the device and synchronize the clocks in order to have correlated information. Next, a shaking of the devices is done. The name of this work comes from the fact that shaking the two devices is similar to shake a Martini. The readings are separated in quantization zones and each zone is divided between an correct zone and an error zone. The preliminary information is if the measurement is in the error zone or not. Alice and Bob reveal their preliminary information. If they agree on a particular point, then the quantization zone where the measurement belongs is kept as private information. If they do not agree, then they discard the information.

With this approach, is possible that Alice and Bob agree when the measurements belong to different zones. Therefore, a correction might be necessary after the distillation of the private strings. Like [Linnartz and Tuyls 2003], we can see that the space is divided in exactly two interval, however the secret information is the quantization zone that the measurement belongs and the helper data tells us if the measurement is in the part odd or even of the quantization zone.

In the first version, the correction is done using BCH codes (described in [Bose and Ray-Chaudhuri 1960]) and transmitting the checksums bits. The leaked information is derived from the amount of communication needed to correct all the errors. In the second version, there is no direct correction, but the quantization steps are calculated in order to assure a low false negative rate.

One important point is that an adversary could try a man-in-the-middle attack. Every time that one party sends a bit b from the preliminary information, he sends $\neg b$ back. By doing this, the adversary determines how many positions he has to guess in order to discover the secret information. The solution proposed is to divided the information in two blocks and each party reveals one of them. Moreover, a minimum of samples with the same preliminary information (odd or even) in each partition is created. With this approach, impersonation becomes harder.

This work was an important advance in key conciliation with noisy information. However, the functions used discard too much information. The rate used for sampling is 220Hz and the rate of bits distilled for seconds is at maximum 20 bits, thus there is a great waste of information. A better result could be achieved if a better cryptography scheme were used.

[Mayrhofer and Gellersen 2007] and [Mayrhofer and Gellersen 2009] is another work that has created a key conciliation protocol based on acceleration. Differently from [Kirovski et al. 2007a] and [Kirovski et al. 2007b] that do the conciliation for each dimension, this work uses the magnitude over all normalized dimensions. After acquiring and processing the information, the protocol has two approaches to distill a key. The first

 © 2017 Sociedade Brasileira de Computação

one uses Diffie-Hellman and Interlock to exchange the times series generated from each part. The security of this scheme is based on computational assumptions, which is undesirable. The second uses the Candidate Key Protocol described in [Mayrhofer 2007] to exchange the time series and extract a key. This approach is very dependent of how the processing of the feature was done, possibly a large amount of information is discarded and, also, high entropy is necessary to avoid off-line lookup table attacks. The lower bound of seven bits extracted per second is achieved in this work.

4.2. Authentication with Wireless Signal Power

[Premnath et al. 2014] created a protocol that measures the wireless signal power of a Bluetooth connection and, from it, derives a key between close devices. Some processing is necessary because of the low entropy of the measured power. Because of this, the bit rate distillation is considerably small, making this protocol less preferable in an real scenario.

4.3. Authentication with Biometry

In [Tuyls et al. 2005], another authentication system is built. The main focus is to use this in fingerprint recognition. A user has his feature measured multiple times and, for every measurement, if the value is greater than the global mean, then the output is 1, otherwise is 0. A feature is reliable if all the outputs are equal. The great disadvantages are the short keys and that there are few features that are really reliable.

5. Classes of codes compared

5.1. Reed-Solomon Codes

Reed-Solomon codes are the most well-known class of codes, appearing in CDs, NASA space missions, QR Codes, and many scientific papers, including [Schurmann and Sigg 2013]. For this reason they serve as a base line, and were therefore included in our comparison.

Reed Solomon codes are a subset of BCH codes and are linear block codes. A Reed-Solomon code is specified as $RS(n, k)$ with s-bit symbols. This means that the encoder takes k data symbols of s bits each and adds parity symbols to make an n symbol codeword. There are $n - k$ parity symbols of s bits each. A Reed-Solomon decoder can correct up to t symbols that contain errors in a codeword, where $2t = n - k$.

Reed-Solomon algebraic decoding procedures can correct errors and erasures. An erasure occurs when the position of an erred symbol is known. A decoder can correct up to t errors or up to $2t$ erasures. Erasure information can often be supplied by the demodulator in a digital communication system, i.e. the demodulator 'flags' received symbols that are likely to contain errors.

When a codeword is decoded, there are three possible outcomes:

1. If $2s + r < 2t$, where s is the number of erasures and r the number of erros, then the original transmitted code word will always be recovered.
2. The decoder will detect that it cannot recover the original code word and indicate this fact.

© 2017 Sociedade Brasileira de Computação

3. The decoder will mis-decode and recover an incorrect code word without any indication.

The probability of each of the three possibilities depends on the particular Reed-Solomon code and on the number and distribution of errors. For detailed information about encoding and decoding RS codes please refer to [Lin and Costello 2004].

5.2. Low-Density Parity Check Codes

Low-Density Parity Check Codes (LDPC) [Gallager 1962] are capacity achieving codes that works using the sum-product algorithm in the decoding process. The graph that this algorithm uses is known as the Tanner graph. It is a bipartite undirective graph. In one side we have the check nodes connected only to the variable nodes, that are on the other side. The check nodes take the value that is believed to be true and check if the constraint of the code is correct. Figure 1 from [Johnson 2009] shows an example of this graph.

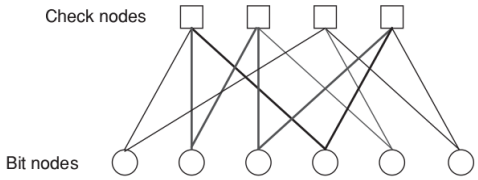


Figure 1. Tanner Graph of (1)

This graph represents the parity matrix:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \quad (1)$$

Each row correspond to a check node and each column to a variable node. The soft and hard decoding approaches can be used with the Tanner graph.

We have that the generator matrix G and parity matrix H have the property that $GH^t = 0$. The encoding is done simply by a matrix multiplication which can be optimized. Words c that belong to the code have the property that $Hc^t = 0$.

5.3. Repeat-Accumulate Codes

Repeat-Accumulate Codes are a simplification of LDPC codes. The encoding process is even simpler than the bipartite graphs of LDPC codes. The following example is taken from [MacKay 2003] (page 582): let the source bits be $s_1 \dots s_K$. Repeat each bit 3 (say) times: $s_1 s_1 s_1 \dots s_K s_K s_K$ and apply some arbitrary but fixed permutation on these bits, to obtain $u_1 u_2 u_3 \dots u_N$ with $N = 3K$. Transmit the accumulated sum: $t_1 = u_1$; $t_i = t_{i-1} \oplus u_i$ for $1 < i \leq N$.

RACs can also be represented through a Tanner graph and therefore decoding is also performed through the sum-product algorithm. RACs have a spectacular performance given their simplicity.

 © 2017 Sociedade Brasileira de Computação

5.4. Low-Density Lattice Codes

Low-Density Lattice Codes (LDLC) [Sommer et al. 2008] are capacity achieving codes of the Unconstrained Power Channel (UPC) [Poltyrev 1994] and are based on the LDPC codes.

The UPC is simply an AWGN channel without the power restriction. So, because of this, another definition of capacity is used: instead of Signal Power, the volume of the lattice is used. This volume is defined as the generator matrix G of the code. In LDLC we have the restriction that $|det(G)| = 1$ and $G = H^{-1}$. In LDPC codes messages are encoded as bits and the distance notion is the Hamming distance. In LDLCs messages are represented as integer vectors with Euclidean distance; they can be interpreted as indexes; the encoded messages are lattice points, and decoding is performed using a sum-product algorithm adapted to the constraints.

One important aspect is that the UPC is merely a theoretical channel; it does not model a true communication scenario. In order to be applicable, the infinite region of the lattice has to be shaped into a finite one. See for example the approaches presented in [Sommer et al. 2009] and [Kurkoski et al. 2009].

Code-Offset Method with LDLC

Although in order to use the UPC in a real communication channel the shaping has to be done, in the code-offset method there is no actual communication, so there is no power restriction. Therefore, LDLC are suitable to be used directly. Compared to a channel like the BSC, we see that it can be used directly because in this specific case it is assumed that the quantization has already been done. By analyzing this, we can see that the shaping is done implicitly by the space of the feature itself.

6. Our simulations

To compare these four classes of codes, we need a good implementation of each one. LDPC codes have a good library in python named pyldpc which we decided to use. We also found a good implementation of the Reed-Solomon codes in the wikiversity site. However we couldn't find implementations for the LDLC and Repeat-Accumulate codes, so we coded those ourselves.

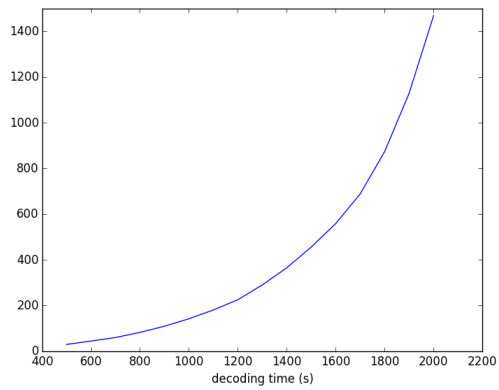
Each Repeat-Accumulate Code is defined by some parameters, as explained before, so after we had a fully functional implementation it was also necessary to search for a good set of them. We did so by generating several random combinations and testing the resulting codes for decoding speed and correction capacity.

6.1. Comparing speed

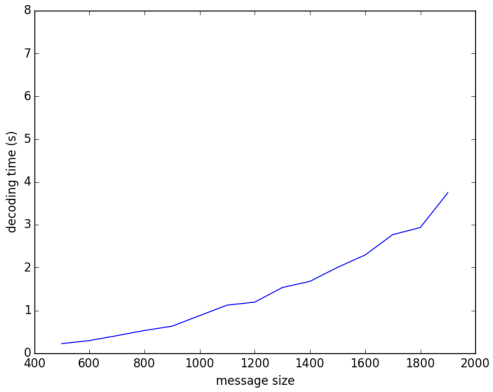
To compare the decoding speed, we generated 5000 random messages with size varying from 500 to 2000 values (these values where bits in the LDPC and Repeat-Accumulate codes, and integers in LDLC and Reed-Solomon codes), we then added noise to them and proceeded with decoding.

This process gave us the following results:

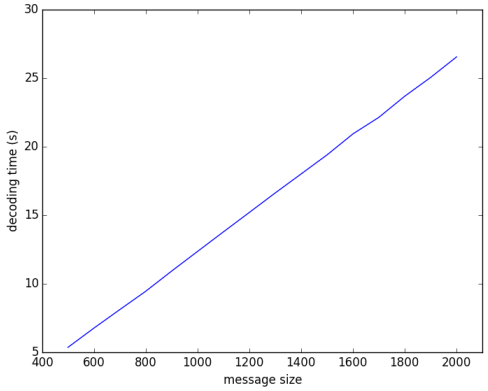
© 2017 Sociedade Brasileira de Computação



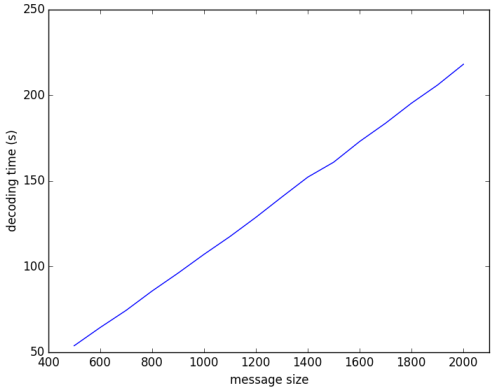
(a) Reed-Solomon Codes



(b) Low-Density Parity-Check Codes



(c) Repeat-Accumulate Codes



(d) Low-Density Lattice Codes

Figure 2. Message Size by Decoding Time

Observing these graphics we see that for small messages Reed-Solomon Codes are the fastest, but as the message grows its decoding time increases substantially.

Low-Density Lattice Codes' decoding time grows linearly with message size, however it is much slower than Low-Density Parity Check and Repeat-Accumulate codes, which also exhibit linear time complexity for decoding. This is a very interesting observation. When we first started working on this project we expected that LDLCs would be superior in every aspect. But practice showed us a complete different reality. Binary codes are much faster, and given the difference in magnitude and the smoothness of its growth we believe this to be a real advantage.

6.2. Comparing correction capability

To compare the codes according to correction capability, we again generate 5000 random messages of size varying from 500 to 2000, added noise, and then tried to correct them. One difference here is that we decided not to include Reed-Solomon Codes because their correction ability is known beforehand. We did not hperform simulations with Repeat-Accumulate Codes; it will be a future work to complete the comparisons adding these codes.

 © 2017 Sociedade Brasileira de Computação

Low-Density Lattice Codes and Low-Density Parity-Check Codes both presented good results.

The noise we added was approaching both codes' limits, meaning that if it was a little higher, neither of them would be able to correct. But within the expected range they performed fine.

6.3. Comparing entropy

Another interesting criterion for comparing these codes is regarding their entropy, that is, how each one spreads its encoded messages in the coding space. That comparison was made only with LDPCs and LDLCs, since they were the most similar in construction (even if the theory behind is completely different) and the obvious difference of one being a binary code and the other a continuous one.

To do so, we treated the all zero message $\overrightarrow{0}$ as the codified message. We added noise to it and then tried to correct it, meaning we tried to correct noise alone, and observed how the codes behaved.

The process of adding noise, and trying to correct it was repeated around 5000 times using each code. After doing that we counted how many times each 'decoded message' appeared. Having these values, the minimum entropy was calculated according to:

$$H_{min} = \frac{count(m_{max})}{\sum_{i=0}^{n} count(m_i)} \tag{2}$$

Here $count(x)$ is the number of times message e was observed and n is the total number of different observed messages.

LDPC decoded all of his messages to the original one $\overrightarrow{0}$, which was to be expected given that the all zero message is a valid codeword for any LDPC instance, due to that its minimum entropy is 0!

LDLC gave us a more interesting result, we didn't get 2 equal messages, meaning that we found a total of 5000 different messages by correcting 5000 'noisy messages'. So according to the formula above, we got a total min entropy of 0.0002203 (we discarded messages that were not successfully corrected to valid code words).

7. Conclusions

With this work we concluded that LDPC codes' performance is still superior when it comes to speed, with Repeat-Accumulate Codes coming right after it.

Both LDPCs and LDLCs show great potential when it comes to correction capability, Reed-Solomon codes are the most trustworthy since its decoding capabilities are pre-defined and we can always make it as large as needed, Repeat-Accumulate codes are very similar to LDPCs but we still can't say much about its correcting capabilities, but due to its simplicity it is worth investigating in the future.

© 2017 Sociedade Brasileira de Computação

8. Acknowledgments

HdRD and JvdG were partially supported by grants from FAPEMIG (Universal 2013 and PROBIC). VPP was partially supported by the Brazilian agency CAPES. All three were partially supported by a grant from LG Electronics Brasil.

References

Bennett, C. H., Brassard, G., Crépeau, C., and Maurer, U. M. (1995). Generalized privacy amplification. *IEEE Transactions on Information Theory*, 41(6):1915–1923.

Bose, R. C. and Ray-Chaudhuri, D. K. (1960). On a Class of Error Correcting Binary Group Codes. *Information and control*, 3(1):68–79.

Brassard, G. and Salvail, L. (1993). Secret-key reconciliation by public discussion. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 410–423. Springer.

Brassard, G. and Salvail, L. (1994). Secret-Key Reconciliation by Public Discussion. In *advances in Cryptology-EUROCRYPT 93*, pages 410–423. Springer.

Buhan, I., Doumen, J., Hartel, P., and Veldhuis, R. (2007a). Constructing Practical Fuzzy Extractors Using QIM. Technical report, Technical Report TR-CTIT-07-52, Centre for Telematics and Information Technology University of Twente.

Buhan, I., Doumen, J., Hartel, P., and Veldhuis, R. (2007b). Fuzzy Extractors for Continuous Distributions. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 353–355. ACM.

Chang, Y.-J., Zhang, W., and Chen, T. (2004). Biometrics-Based Cryptographic Key Generation. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, volume 3, pages 2203–2206. IEEE.

Dodis, Y., Ostrovsky, R., Reyzin, L., and Smith, A. (2008). Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM journal on computing*, 38(1):97–139.

Gallager, R. (1962). Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28.

Goodrich, M. T., Sirivianos, M., Solis, J., Tsudik, G., and Uzun, E. (2006). Loud and Clear: Human-Verifiable Authentication Based on Audio. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 10–10. IEEE.

Johnson, S. J. (2009). *Iterative error correction: Turbo, low-density parity-check and repeat-accumulate codes*. Cambridge University Press.

Juels, A. and Sudan, M. (2006). A Fuzzy Vault Scheme. *Designs, Codes and Cryptography*, 38(2):237–257.

Kirovski, D., Sinclair, M., and Wilson, D. (2007a). The Martini Synch. Technical report, Technical Report MSR-TR-2007-123, Microsoft Research.

Kirovski, D., Sinclair, M., and Wilson, D. (2007b). The Martini Synch: Using Accelerometers for Device Pairing. Technical report, Citeseer.

 © 2017 Sociedade Brasileira de Computação

Kumar, A., Saxena, N., Tsudik, G., and Uzun, E. (2009). Caveat emptor: A comparative study of secure device pairing methods. In *Seventh Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2009, 9-13 March 2009, Galveston, TX, USA*, pages 1–10.

Kurkoski, B. M., Dauwels, J., and Loeliger, H.-A. (2009). Power-constrained communications using ldlc lattices. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 739–743. IEEE.

Lin, S. and Costello, D. J. (2004). *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Linnartz, J.-P. and Tuyls, P. (2003). New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates. In *Audio-and Video-Based Biometric Person Authentication*, pages 393–402. Springer.

MacKay, D. J. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge university press.

Mayrhofer, R. (2007). The Candidate Key Protocol for Generating Secret Shared Keys from Similar Sensor Data Streams. In *Security and Privacy in Ad-hoc and Sensor Networks*, pages 1–15. Springer.

Mayrhofer, R. and Gellersen, H. (2007). Shake Well Before Use: Authentication Based on Accelerometer Data. In *Pervasive computing*, pages 144–161. Springer.

Mayrhofer, R. and Gellersen, H. (2009). Shake Well Before Use: Intuitive and Secure Pairing of Mobile devices. *Mobile Computing, IEEE Transactions on*, 8(6):792–806.

Poltyrev, G. (1994). On coding without restrictions for the awgn channel. *IEEE Transactions on Information Theory*, 40(2):409–417.

Premnath, S. N., Gowda, P. L., Kasera, S. K., Patwari, N., and Ricci, R. (2014). Secret Key Extraction Using Bluetooth Wireless Signal Strength Measurements. In *Sensing, Communication, and Networking (SECON), 2014 Eleventh Annual IEEE International Conference on*, pages 293–301. IEEE.

Reed, I. S. and Solomon, G. (1960). Polynomial Codes Over Certain Finite Fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304.

Schurmann, D. and Sigg, S. (2013). Secure Communication Based on Ambient Audio. *Mobile Computing, IEEE Transactions on*, 12(2):358–370.

Sommer, N., Feder, M., and Shalvi, O. (2008). Low-density lattice codes. *IEEE Transactions on Information Theory*, 54(4):1561–1585.

Sommer, N., Feder, M., and Shalvi, O. (2009). Shaping methods for low-density lattice codes. In *Information Theory Workshop, 2009. ITW 2009. IEEE*, pages 238–242. IEEE.

Tuyls, P., Akkermans, A. H., Kevenaar, T. A., Schrijen, G.-J., Bazen, A. M., and Veldhuis, R. N. (2005). Practical Biometric Authentication with Template Protection. In *Audio- and Video-Based Biometric Person Authentication*, pages 436–446. Springer.

Verbitskiy, E. A., Tuyls, P., Obi, C., Schoenmakers, B., and Skoric, B. (2010). Key Extraction from General Nondiscrete Signals. *Information Forensics and Security, IEEE Transactions on*, 5(2):269–279.

© 2017 Sociedade Brasileira de Computação

Zheng, G., Li, W., and Zhan, C. (2006). Cryptographic Key Generation from Biometric Data Using Lattice Mapping. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 4, pages 513–516. IEEE.

 © 2017 Sociedade Brasileira de Computação

Detecção de Canvas Fingerprinting em Páginas Web baseada no Modelo Vetorial

Pablo A. da P. Elleres[1], **Adria M. de Oliveira** [1], **Eduardo L. Feitosa**[1]

[1]pabloelleres14@gmail.com, adriah.menezes@gmail.com, efeitosa@icomp.ufam.ed.br

Abstract. *This paper presents a method for evaluating Canvas Fingerpriting scripts in Web pages. The method consists in calculating the similarity between a database with 100 queries related to Canvas Fingerpriting and databases with pages considered as benign and malignant. The results show high levels of similarities with a canvas base (99%), a base of phishing pages (92%), a base with pages from the DMOZ directory (91%), and a base with pages of Alexa.com (97%).*

Resumo. *Este trabalho apresenta um método para realizar avaliação de scripts Canvas Fingerpriting em páginas Web. O método consiste em realizar o cálculo da similaridade entre uma base com 100 consultas reconhecidamente ligadas à Canvas Fingerpriting e bases de dados com páginas tidas como benignas e malignas. Os resultados mostram altos níveis de similaridades com uma base de Canvas (99%), uma base de páginas phishing (92%), uma base com páginas do diretório DMOZ (91%) e uma base com páginas do site Alexa.com (97%).*

1. Introdução e Motivação

Embora recente, o uso de técnicas de *Website Fingerprinting* tornou-se comum como forma de identificação ou re-identificação de usuários na *Web*. Parte desse sucesso se deve as iniciativas de limitar o uso de cookies, mas sua popularização pode ser atribuída as várias possibilidades (formas e tecnologias) de implementação e execução. Isso porque *Website Fingerprinting* recolhe um conjunto de propriedades de um dispositivo (endereço IP, tamanho da tela, hardware, entre outros) e dos softwares instalados (versões, tipos de fontes, entre outros), a partir de um navegador [Saraiva et al. 2016], para caracterizar um usuário. A questão é que esse "perfil" é obtido tipicamente sem o consentimento dos usuários, o que gera uma série de preocupações. Mesmo que alguns usuários não vejam problemas, a posse de tais informações podem trazer consequências potencialmente desastrosas para a privacidade das pessoas e, em casos mais graves, pode envolvê-las em furtos, fraudes e ataques maliciosos.

Inicialmente, os artefatos de *fingerprinting* exploravam as possibilidades (métodos e propriedades) acessíveis apenas via linguagens como JavaScript e ActionScript (Flash). *Timezone*, *user-agent* e *geolocation* são bons exemplos de propriedades utilizadas nos *fingerprintings* elaborados nessas linguagens. Entretanto, com o tempo, novas tecnologias passaram a ser empregadas para identificar um usuário/dispositivo. É o caso do *fingerprinting* baseado na tecnologia HTML 5 Canvas, que faz uso de propriedades e métodos gráficos. A ideia de um artefato Canvas *fingerpriting* é instruir o navegador a desenhar linhas, textos, figuras geométricas, entre outros, que a posteriori são convertidos em um identificador único.

 © 2017 Sociedade Brasileira de Computação

Mas, se elaborar um *fingerprinting* é relativamente fácil e simples, sua detecção não é. A questão não é, simplesmente, ao encontrar uma característica (um método, uma propriedade ou um atributo) de *fingerprinting* afirmar que o site realiza *Website Fingerprinting*, mas sim quão intrusiva ou perigosa essa característica é. Tomando Canvas como exemplo, o simples fato do site desenhar imagens ou textos não o caracteriza como *fingerprinting*, mas sim a posteriori captura da imagem desenhada e o envio para um servidor [Ximenes et al. 2016].

É neste cenário de incerteza que este artigo propõe um método capaz de detectar e avaliar scripts Canvas *fingerprinting* em páginas Web, com base na extração de características relevantes do conteúdo estático do documento *Web*, por meio de técnicas de recuperação da informação, a fim de apresentar um ranqueamento destes scripts por níveis de similaridade. Desta forma, é possível não só conhecer os scripts originais, mas também o seu grau de periculosidade, informando aos usuários quando acessarem páginas *Web* com este tipo de mecanismo. O método proposto também analisa um conjunto de características relevantes que podem ser extraídas de páginas *Web*, de acordo com sua capacidade de gerar ataques ou ameaças à privacidade dos usuários *Web*, permitindo sua classificação por níveis de similaridade. Esta classificação por nível de similaridade é interessante não só por oportunizar uma comparação entre os scripts, mas principalmente porque poderá servir para uma futura implementação de um plug-in.

Para validar a proposta, uma prova de conceito foi implementada e fez uso de scripts obtidos em bases reais da Internet, como, por exemplo, da Universidade de Princeton (base Canvas), do Phishtank, do Dmoz e do site Alexa.com. Os resultados mostraram altos níveis de similaridade nas bases avaliadas, comprovando que o método criado, baseado no cálculo de similaridade do modelo vetorial, é capaz de detectar Canvas *fingerprinting* em páginas *Web*.

2. Fundamentação Teórica

Esta seção apresenta os conceitos e fundamentos empregados neste trabalho.

2.1. *Website Fingerprinting*

No âmbito *Web*, o termo *fingerprinting* veio a tona em 2009 quando Mayer [Mayer 2009] observou que as características de um navegador e seus *plugins* podiam ser identificados e o usuário rastreado. Em 2010, Eckersley [Eckersley 2010] mostrou que informações (atributos) fornecidas pelo navegador dos usuários eram suficientes para identificar a grande maioria das máquinas que navegam na Internet. Para tanto, desenvolveu um algoritmo para investigar o grau em que os navegadores modernos estão sujeitos às técnicas de *fingerprinting*. Dos mais de 470.000 usuários que participaram de seu projeto público (Panopticlick[1]), 84% tiveram seus navegadores identificados.

Formalmente, é importante salientar a diferença entre os termos *Fingerprint* e *Fingerprinting*. De acordo com a RFC 6973 [Cooper et al. 2013], o primeiro é definido como "um conjunto de elementos de informação que define um dispositivo ou uma instância de uma aplicação" e o segundo como "o processo pelo qual um observador ou atacante identifica, de maneira única e com alta probabilidade, um dispositivo ou uma instância de um aplicativo com base em um conjunto de múltiplas informações".

[1] http://panopticlick.eff.org

 © 2017 Sociedade Brasileira de Computação

Neste artigo, os termos *Website Fingerprinting* e *Fingerprinting* serão usados para representar essas técnicas de identificação com foco na *Web*. Mais detalhes sobre o assunto podem ser encontrados nos trabalhos de [Saraiva et al. 2014] e [Saraiva et al. 2016].

2.2. HTML5 Canvas

Canvas é um elemento da HTML5 que fornece uma área da tela que pode ser utilizada via programação, proporcionando acesso a um conjunto completo de funções de desenho e geração dinâmica de gráficos [W3C 2015].

No que diz respeito a *fingerprint*, um típico script Canvas primeiro desenha um texto com fontes, tamanho e cores de fundo variados [Acar et al. 2014]. Em seguida, chama o método *toDataURL*, da API Canvas, para obter os dados de pixel da tela em formato *DataURL* - uma representação codificada em Base 64 dos dados de pixel binários. Por fim, obtém o *hash* dos dados de pixel codificado de texto, que serve como *fingerprint*. Mowery e Shacham [Mowery and Shacham 2012] observaram que usando Canvas é possível relacionar o navegador com o hardware e o sistema operacional.

2.3. Modelo Vetorial

O modelo vetorial, também conhecido como espaço vetorial, trata documentos e consultas como características num espaço vetorial n-dimensional, sendo a distância vetorial usada como medida de similaridade. Formalmente, um documento d_j e uma consulta de usuário q são representados como vetores com t dimensões. O modelo calcula o grau de similaridade ente o documento e a consulta sob forma de correlação entre os vetores d_j e q, podendo ser medida, por exemplo, pelo cosseno do ângulo entre esses dois vetores [Baeza-Yates 2013].

A Figura 1 ilustra um exemplo do modelo vetorial ao posicionar um documento no espaço vetorial de três dimensões, cada uma representando um índice. Cada documento é visto como termos no espaço n-dimensional através dos eixos (x=4, y=5 e z=3). O ponto de interseção entre estes termos é representado pelo documento DOC = (5, 4, 3).

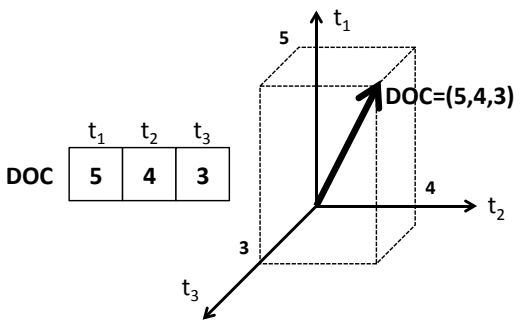


Figura 1. Exemplo de uso do Método Vetorial. Fonte: Adaptado de [Ramiro et al. 2005]

Neste trabalho, um documento é um script Canvas *fingerprinting* qualquer e uma consulta é um dos scripts rotulados como *fingerprinting*.

© 2017 Sociedade Brasileira de Computação

2.4. Detecção de *Knee Points*

Este trabalho emprega o conceito de detecção de *Knee Points* (em tradução livre, pontos de joelho), definida por Satopää et al. [Satopää et al. 2011], com a finalidade de encontrar um ponto em uma função na qual não haverá tanta variação de resultados e realizar um corte nos resultados de similaridade naqueles em que o algoritmo não considerar relevantes. Satopää et al. [Satopää et al. 2011] destacam que detecção de *Knee Points* é um processo inerentemente heurístico que independe da aplicação, ou seja, em uma definição coerente ele pode ser aplicável a qualquer sistema. Contudo, existe uma grande dificuldade em defini-lo formalmente, pois este pode ser "bom o suficiente" em um sistema, mas pode não ser "bom o suficiente" em outro.

Neste artigo, o emprego de *Knee Points* permite reduzir a quantidade de amostras sem a perda na qualidade dos dados. O exemplo a seguir elucida o uso de *Knee Points* neste trabalho. Ao realizar a comparação entre uma consulta com, por exemplo, 20.000 documentos (scripts), supõem-se que são obtidos mais de 4.000 scripts similares a esta consulta. Neste caso, o algoritmo *Knee Points* verifica um ponto da curvatura da função em que não há tanta representatividade (devido a invariância dos resultados de similaridade) e elimina estas comparações, resultando em, por exemplo, 1.000 scripts similares.

3. Trabalhos Relacionados

Nesta seção são apresentas pesquisas relacionadas ao uso da API Canvas para *fingerprinting* visando a identificação de usuários/dispositivos em páginas Web.

Englehardt e Narayanan [Englehardt and Narayanan 2016] utilizaram a plataforma OpenWP para vasculhar os sites que fazem parte do top 1 milhão da Alexa.com, com o objetivo de analisar quais deles rastreiam seus usuários por *fingerprinting*. Embora tenham testado várias formas de *fingerprinting* (API AudioContext, API Battery, CookieSync e WebRTC), foram detectados 14.371 sites, da base de 1 milhão (cerca de 1,6 %), contendo as propriedades e métodos Canvas como *toDataURL*, *getImageData*, *save* e *restore*.

Laperdrix et al. [Laperdrix et al. 2016] realizaram um estudo sobre os atributos mais relevantes para extrair o *fingerprinting* do dispositivo do usuário e analisar a eficácia da técnica tanto em dispositivos móveis quanto em computadores. Criaram um script de *fingerprinting* que faz uso de 10 atributos utilizados no trabalho de Eckersley [Eckersley 2010] e mais 7 atributos adicionais provenientes da tecnologia Web (Java Script, HTML5 e Canvas), hospedado no site AmIUnique.org. O site registrou 118.934 acessos, o que permitiu aos autores identificar, de maneira única, 89.4% dos dispositivos analisados.

Ximenes et al. [Ximenes et al. 2016] desenvolveram um mecanismo baseado em Canvas para identificar usuários na Web resistente às contramedidas que bloqueiam ou alteram o comportamento do navegador. O TARP *Fingerprinting* implementa três grupos distintos de instruções gráficas Canvas, para permitir diferentes níveis de entropia dos dados coletados. Foram coletadas 64.086 assinaturas (cada assinatura equivale a uma instância do TARP), na qual, por meio do cálculo da entropia, era verificado o uso ou não de contramedidas. Os resultados obtidos provaram a eficácia da solução em burlar as contramedidas e mostraram que quanto maior o número de instruções gráficas empregadas, maior a entropia.

 © 2017 Sociedade Brasileira de Computação

Bursztein et al. [Bursztein et al. 2016] desenvolveram e avaliaram uma técnica capaz de detectar e prevenir ataques automatizados de clientes maliciosos (possivelmente automatizados) em lojas de aplicativos, através de desafios elaborados em Canvas. A intenção foi identificar classes de dispositivos, unindo os atributos do navegador e do sistema operacional. Os desafios baseiam-se em quatro operações de desenhos (primitivas gráficas): *arc()*, *strokeText()*, *bezierCurveTo()*, *quadraticCurveTo()*. Desta forma, os autores realizaram dois experimentos: um em ambiente fechado, contendo 272,198 dispositivos, e outro em ambiente aberto, com a participação anônima de 52 milhões de usuários. Em ambos os casos, a ferramenta conseguiu fazer a distinção das classes dos dispositivos com 100% de acurácia, aplicando-se para todas as combinações de navegadores e sistemas operacionais (sejam estes móveis ou desktops).

Nakibly et al. [Nakibly et al. 2015] apresentaram técnicas de *Device Fingerprinting* baseadas em HTML5 e características de hardware, implementadas no site fingerprintme.herobo.com. O *fingerprinting* proposto realiza três fases de medições (verificar a frequência de relógio, verificar o número de núcleos e verificar outros parâmetros que afetam o desempenho da GPU). Cada fase das medições deu-se por meio da API Canvas *requestAnimationFrame*. Como resultados, os autores foram capazes de identificar 130 dispositivos unicamente, onde 34 destes foram "fingerprintados" mais de uma vez.

Acar et al. [Acar et al. 2014] investigaram os 100.000 sites mais populares do Alexa.com a procura de scripts *fingerprinting*, incluindo os baseados em Canvas. Dentre os métodos utilizados para detectar o Canvas *fingerprinting*, destacam-se *toDataURL*, *fillText* e *strokeText*. O resultado mostrou que dos 100.000 sites analisados, 5.5% utilizam a tecnologia HTML5 Canvas para realizar *Website Fingerprinting*, sendo 95% dos scripts de Canvas pertencentes a um único provedor (*addthis.com*), e os demais 5% restantes pertencem a outros 20 provedores (11 de companhias terceirizadas e 9 desconhecidos).

3.1. Discussão

A Tabela 1 sumariza os trabalhos apresentados, incluindo o método proposto neste artigo.

Tabela 1. Comparação entre Trabalhos com Canvas Fingerprinting

Artigo	Método de Detecção	Propriedades/Métodos Canvas	Fonte de Dados
[Englehardt and Narayanan 2016]	Ferramenta OpenWPM	toDataURL, getImageData, save e restore	OpenWPM
[Laperdrix et al. 2016]		fillText, fillStyle, fillRect, toDataURL, strokeText, globalCompositeOperation, lineTo, arc, canvas.Text, getImageData	AmIUnique
[Ximenes et al. 2016]	Entropia Ponderada e de Shannon	toDataURL	Tarp FP
[Bursztein et al. 2016]	Máxima Entropia	arc, strokeText, bezierCurveTo, quadraticCurveTo, createRadialGradient, shadowBlur, shadowColor	Picasso
[Nakibly et al. 2015]		requestAnimationFrame	fingerprintme
[Acar et al. 2014]		DataURL, fillText, strokeText	alexa.com
Canvas FP no modelo vertorial	Método Vetorial e Knee points	**Propriedades** fillStyle, Canvas.font, textBaseline, Canvas.width, Canvas.height, strokeStyle, globalAlpha, lineWidth, lineCap, lineJoin, miterLimit, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, globalCompositeOperation, textAlign e **Métodos** fillRect, fillText, lineTo, arcTo, beginPath, clearRect, createImageData, createPattern, createRadialGradient, measureText, putImageData, quadractiveCurveTo, restore, rotate, scale, setTransform, strokeRect, strokeText, getImageData, toDataURL, getElementById('Canvas'), Canvas.getContext, createElement('Canvas'), getElementsByTagName('Canvas')	Diversas

© 2017 Sociedade Brasileira de Computação

De modo geral, percebe-se que a maioria das pesquisas apresentadas focam provar a existência do problema, desenvolvendo, em sua maioria, algum site para realizar esta identificação dos usuários/dispositivos. Além disso, a maioria das pesquisas discutidas tem uma abordagem on-line e não faz uso de scripts.

Em relação às características (propriedades e métodos Canvas), percebe-se que algumas são bastante recorrentes, mas outras são mais raras, fato que proporciona uma atenção maior a estas com um número de incidências menor. Por exemplo, as propriedades *toDataURL* e *getImageData*, presentes nos trabalhos de [Englehardt and Narayanan 2016], [Laperdrix et al. 2016], [Ximenes et al. 2016] e [Acar et al. 2014], são elencadas no trabalho de Saraiva [Saraiva et al. 2016] como características de alta periculosidade, por evidenciar a prevalência maior da existência ou não do Canvas *fingerprinting*. Neste trabalho não são utilizadas somente essas 17 propriedades/métodos mencionados, mas sim outras 24 propriedades para uma análise mais abrangente.

Já sobre a classificação e o ranqueamento, é importante salientar que não existem trabalhos que empregam um método para predizer os níveis de similaridade. Tudo leva a crer que este trabalho é o primeiro a fazer isso. Acerca das bases de dados utilizadas, os trabalhos ou utilizam o site Alexa.com ou coletaram seus próprios dados. Já este trabalho emprega 4 bases de dados distintas: Canvas, PhishTank, Alexa.com e DMOZ.

4. Método Proposto

O método proposto objetiva detectar scripts Canvas *fingerprinting* em páginas Web, empregando o modelo vetorial, a partir de um conjunto de características de interesse, para calcular a similaridade dos scripts e ranquea-los. A Figura 2 exemplifica o método proposto, apresentando suas etapas de funcionamento.

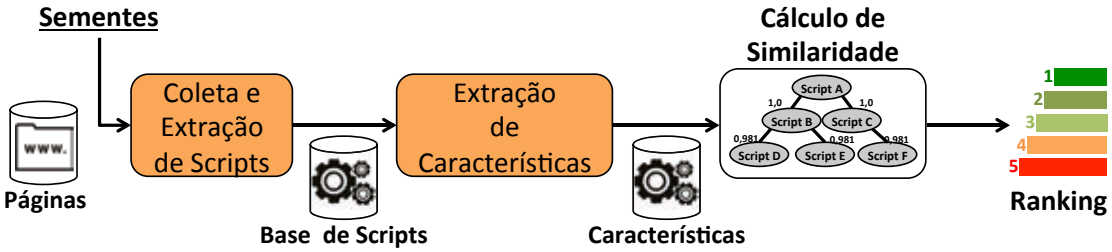


Figura 2. Método Proposto

A etapa de **Coleta e Extração de Scripts** corresponde a coleta e extração dos scripts integrantes do código HTML das páginas Web, tipicamente fazendo uso de um crawler. A etapa seguinte, **Extração das Características**, objetiva identificar os termos (objetos, propriedades e métodos) relacionados à Canvas *fingerprinting* presentes nos scripts coletados na etapa anterior. Para tanto, emprega um dicionário de termos, previamente estabelecido com base na literatura acadêmica, composto por 41 termos.

Na próxima etapa, **Cálculo de Similaridade**, o modelo vetorial calcula o grau de similaridade de uma consulta (propriedades e métodos Canvas *fingerprinting* existentes em cada script) em relação a um documento prévio (da base Canvas, por exemplo) através da similaridade entre cossenos. Para melhorar o entendimento desse processo, a Tabela

 © 2017 Sociedade Brasileira de Computação

2 ilustra o resultado do cálculo de similaridade da consulta `http://2016election.com` em comparação aos 8.000 scripts da base Canvas (documentos).

Tabela 2. Exemplo de Valor de Similaridade na Base Canvas

Consulta	Ranqueamento	Script na Base	Nível de Similaridade
http://2016election.com	1	Pistolsfiringbloy	0.9802
	2	Akdirahost	0.9539
	...	...	...
	324	Fieldgulls	0.4209
	...	...	...
	1724	Creativeplanetnetwork	0.1001
	...	...	...
	7992	Makespace	0.00368
	7993	Profilepic	0.00242

Nesta tabela 2, os scripts da base mais similares a consulta são ordenados de acordo com a similaridade, em ordem decrescente. É necessário mencionar que o método vetorial realiza o cálculo da similaridade até para aqueles scripts que possuem pelo menos um termo igual ao da consulta. Entretanto, só são contabilizados e validados para este método, os valores que estejam entre o intervalo de similaridade de 0.1 até 1.0.

A quarta e última etapa é o **Ranking**, onde é realizado o rankeamento dos scripts cuja similaridade foi calculada na etapa anterior. Em outras palavras, o método vetorial calcula a similaridade e gera um ranqueamento inicial, que o algoritmo *Knee Points* usa para definir um corte e apresentar os mais relevantes. Tomando a consulta usada na Tabela 2 como exemplo (`http://2016election.com`), o método vetorial encontrou 3.724 scripts similares na base de controle e os ranqueou. Em seguida, após a execução do algoritmo *Knee Points* obteve-se uma redução para 324 scripts.

4.1. Detalhes de Implementação

De maneira prática, a implementação do método proposto utiliza:

- Script em linguagem PHP (versão 5.5.15) como Crawler, onde para cada site visitado, são coletados todos os scripts em JavaScript e armazenados em um único arquivo por site;
- Script em linguagem Python 3.4 para extrair as características, por meio de expressões regulares, nos scripts JavaScript. É importante mencionar que ele não é capaz de lidar com scripts ofuscados e mal formados;
- Script em Python 3.4 para realizar tanto o cálculo da similaridade quanto o rankeamento, resultando em um arquivo texto rotulado para cada consulta.

5. Protocolo Experimental

Esta seção descreve o protocolo experimental necessário para avaliação do método proposto.

5.1. Ambiente

Os experimentos realizados foram executados em uma estação de trabalho Intel Core i7 de 2.7 Ghz, com 8 GB de memória RAM e disco SATA de 1 TB de armazenamento sob a plataforma Linux, distribuição Ubuntu 11.10.

© 2017 Sociedade Brasileira de Computação

5.2. Bases de Dados

Este trabalho empregou quatro (4) bases de dados para realização dos experimentos: Canvas, Phishtank, DMOZ e Alexa. A base **Canvas** é oriunda do trabalho de Englehardt et al. [Englehardt and Narayanan 2016]. Assim, a base foi composta por 8.000 scripts coletados nos meses de agosto a novembro de 2016. A base **Phishtank**[2] é composta por scripts de sites relacionados a *phishing* na Internet. É composta por 2.050 scripts coletados entre novembro e dezembro de 2016.

A base **DMOZ**[3] contém scripts de sites considerados benignos e é composta por 596 scripts. Os scripts da base foram coletados no mês de agosto de 2016. A base **Alexa**[4], da empresa Amazon, é composta por sites ranqueados pela quantidade de usuários que os visitam em determinado período. Foram escolhidos apenas sites do Brasil relacionados a conteúdo adulto, redes sociais e compras, uma vez que Nikiforakis et al. [Nikiforakis et al. 2013] afirma que estes tipos de sites fazem uso de técnicas de *fingerprinting*. A base é composta por 1.478 scripts, coletados no mês de setembro de 2016.

Por fim, foi montada uma base de consultas para realizar a comparação com as bases de dados. Desta forma, foram escolhidos 100 scripts comprovadamente de Canvas, de forma aleatória, do repositório da Universidade de Princeton. Vale ressaltar que embora seja o mesmo local de origem da base Canvas, os 100 scripts não se repetem (não parte da base Canvas).

5.3. Avaliação

Para melhor apresentar os resultados obtidos, foram propostos três (03) cenários de experimentação. O **Cenário 1** mede a similaridade entre os scripts da base de dados e das 100 consultas, quantificando o número de vezes que aparecem no topo do *ranking*. Para exemplificar, considere que o site `http://www.google.com.br` esteja em uma das bases de dados analisadas. O método vetorial, ao fazer a comparação dos scripts desta base de dados com os scripts das 100 consultas, irá retornar que o script do site mencionado apareceu 10 vezes no ranking. A intenção é demonstrar, de maneira generalizada, a quantidade de vezes que os sites (scripts das bases de dados) apareceram no ranqueamento.

O **Cenário 2** reflete o inverso do primeiro. Ele visa demonstrar quais das 100 consultas mais se repetem na comparação com as bases de dados. O **Cenário 3** apresenta o nível de similaridade entre os scripts da base de dados analisada e as 100 consultas. Além do valor da similaridade, calculado pelo modelo vetorial, foi empregada a classificação de risco da ISO 27005 [ISO 2008], onde um nível de risco alto corresponde a uma similaridade entre 80% e 100%, um nível de risco médio corresponde a valores entre 50% e 79% de similaridade e um nível de risco baixo à valores menores à 50%.

6. Resultados

Esta seção explicita os resultados alcançados e subdivide-se de acordo com os cenários.

[2]http://www.phishtank.com/
[3]http://DMOZ.org/
[4]http://www.alexa.com/

 © 2017 Sociedade Brasileira de Computação

6.1. Resultados do Cenário 1

A Tabela 3 apresenta os valores da aplicação dos scripts das bases em relação as 100 consultas, considerando apenas aqueles scripts com mais de 20 repetições no ranqueamento e com similaridade igual à 1.0 (100%).

Tabela 3. Similaridade entre Scripts

Base	Total de Scripts	Scripts	Ranqueamentos
Canvas	8.000	guidingtech, ftlauderdalewebcam, preppyrunner, embassy-pages, nzhuntingandshooting	29
		sandraandwoo, filmhafizasi, dcclothesline, cleverlyinspired	24
		eastcoastcreativeblog	23
Phishtank	2.050	caraudioacapulco	41
		sigarabirakmak	40
		info-setting2016, ricardoeletro2, infobel, maisponto, lagos-statenews	29
		replacementroofingtx, rcacas, davinciresidence	24
DMOZ	596	mypet-memorial	44
		sonicyoga, modern-rocket, mdsafrica	43
		ipanta-rhei	42
		bobthealien, cottonclouds, shesmoke, plum	29
Alexa	1.478	iadulto, caadf, agorams, aiesec, bigshopping	41
		biomedicinapadrao, amofilmeshd, batepapo, cidade-brasil, bussolaescolar	29

É possível notar na Tabela 3 que, para a base Canvas, três grupos de scripts (páginas) apresentaram maior incidência no ranqueamento, obtendo respectivamente 29, 24 e 23 vezes. Em outras palavras, os scripts $guidingtech$, $ftlauderdalewebcam$, $preppyrunner$, $embassypages$ e $nzhuntingandshooting$ apresentaram similaridade de 100% com 29 das 100 consultas. E assim sucessivamente para os outros grupos de script. Na base Phishtank, quatro grupos de scripts foram ranqueados respectivamente, 41, 40, 29 e 23 vezes. Na base DMOZ, quatro grupos de scripts foram ranqueados 44, 43, 42 e 29 vezes, respectivamente. Por fim, na base Alexa, dois grupos se destacaram com 41 e 29 repetições.

Um ponto a ser destacado na Tabela são os valores que as bases Alexa e DMOZ obtiveram. Por se tratar de uma base "benigna", não se esperavam sites ranqueados mais de 10 vezes, mas a DMOZ obteve quatro grupos de scripts com respectivamente, 44, 43, 42 e 29 repetições. Uma explicação para tal fato, um resultado de similaridade tão alto em comparação com as consultas Canvas, tem a ver com o uso das caraterísticas de *fingerprinting* estarem presentes nos mais diversificados tipos de sites, alguns apenas para ajustar conteúdos, mas outros aptos à capturar dados dos usuários que os visitam. Por outro lado, a base Alexa que possui sites cujo conteúdo é mais ligado a *fingerprinting*, apresentou apenas dois grupos de scripts com 41 e 29 repetições, respectivamente.

Por fim, apenas para conhecimentos, os sites listados na Tabela 3 são relacionados a: (i) elaboração de vídeos, notícias, desenhos, filmes, decoração, guia técnico, consulados, entre outros, na base Canvas; (ii) notícias, serviços, design, decoração, e-commerce, entre outros, na base Phishtank; (iii) pet shop, religião/espiritualidade, oferta de serviços audio- visuais e serviço de teste de DNA, na base DMOZ; e (iv) conteúdo adulto, notícias, e-commerce, médico, filmes, bate papo, principais cidades do Brasil e escolar, na base Alexa.

© 2017 Sociedade Brasileira de Computação

6.2. Resultados do Cenário 2

A Figura 3 ilustra as as 10 consultas mais relevantes para as quatro bases de dados.

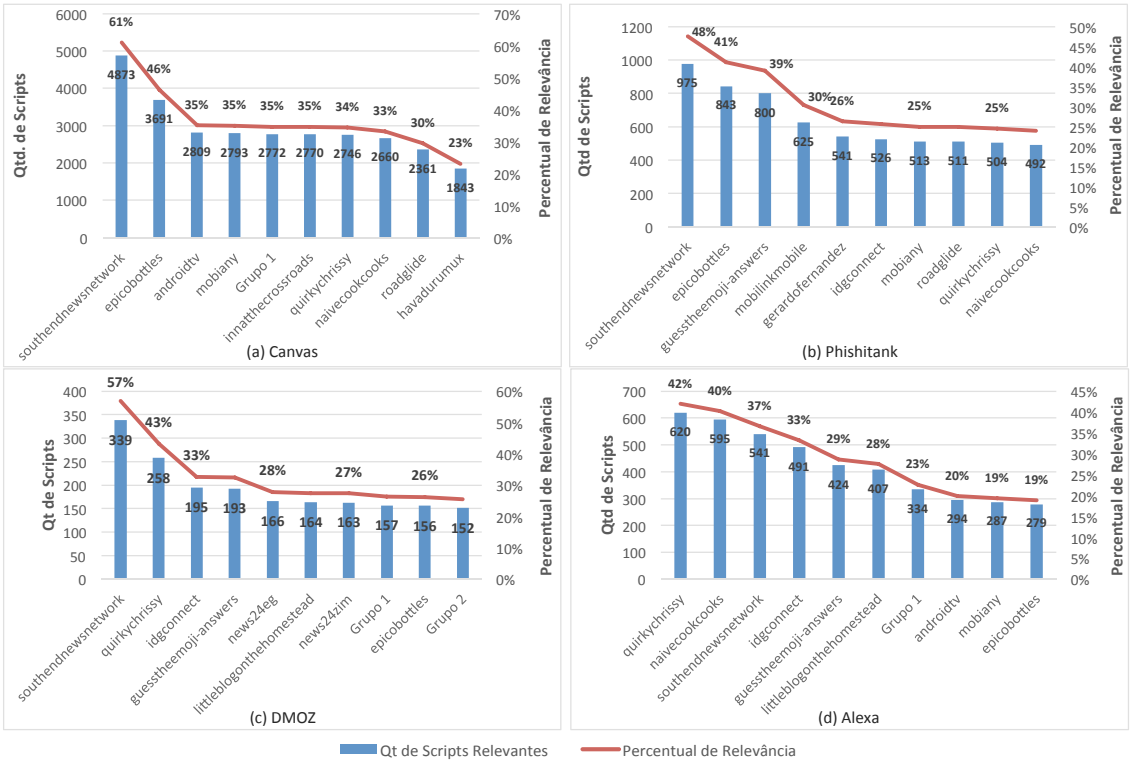


Figura 3. Consultas mais Relevantes

.

Percebe-se na Figura 3 que a base Canvas (a) possui duas consultas (`http://southendnewsnetwork.com` e `http://epicobottles.de`), as quais obtiveram um percentual de relevância de 61% e 46% respectivamente, sendo a primeira relacionada 4.873 scripts da base e a segunda um total de 3.691 scripts. Na base Phishtank, as mesmas consultas são as mais relevantes, tendo percentuais de 48% (para 975 scripts) e 41% (para 843 scripts). Já para a base DMOZ, o script da consulta `http://southendnewsnetwork.com` também foi o mais relevante, obtendo um percentual de 57% com 339 scripts da base, a qual possui um total de 593 scripts. O outro script dessa base é a consulta `http://quirkychrissy.com`, que apresenta um nível percentual de relevância de 43%. Por fim, na base Alexa, o script da consulta `http://quirkychrissy.com` tem resultado bastante similar ao da base de dados DMOZ, com 42% de relevância. E a segunda consulta, script `naivecookcooks`, tem relevância de 40%.

Vale explicar que o grupo 1 presente na Figura 3 (a) representa 40 scripts que obtiveram relevância de 35% em relação as 100 consultas. Os grupos 1 e 2 na Figura 3 (c) representam, respectivamente, 2 e 40 scripts que obtiveram, ambos, relevância de 26% em relação as 100 consultas. Por fim, o grupo 1 na Figura 3 (d) representa 2 scripts que obtiveram relevância de 23% em relação as 100 consultas.

O fato dessas consultas serem tão relevantes para cada uma das bases de dados mencionadas se deve à semelhança entre as características prevalentes entre elas.

© 2017 Sociedade Brasileira de Computação

Todas possuem em comum características como *fillText*, *textBaseline*, *toDataURL*, *clearRect*, *getImageData*. A consulta `http://southendnewsnetwork.com` possui três características a mais (*scale*, *textAlign*, *restore*). A consulta `http://epicobottles.de` têm apenas *scale* como característica adicional. Para a consulta `http://quirkychrissy.com` existe a mais as características *rotate* e *restore* e, por fim, a consulta `http://naivecookcooks.com` tem somente a característica *restore*. O interessante sobre algumas dessas características é que o trabalho de Saraiva et al. [Saraiva et al. 2016] destaca que duas delas, *getImageData* e *toDataURL*, são classificadas com de alta periculosidade e empregadas em *fingerprinting* maliciosos.

6.3. Resultados do Cenário 3

A função deste experimento é demonstrar os níveis percentuais de similaridade obtidos aplicando-se as 100 consultas no modelo vetorial para cada uma das quatro bases de dados. A Tabela 4 apresenta, de maneira sumarizada, os resultados obtidos nas 4 (quatro) base de dados, elucidando os níveis percentuais de similaridade obtidos em cada uma.

Tabela 4. Nível Percentual de Similaridade

Base	Qt. Sites	Baixo	Médio	Alto
Canvas	8.000	0%	1%	**99%**
Phishtank	2.050	2%	6%	**92%**
DMOZ	596	1%	8%	**91%**
Alexa	1.478	2%	1%	**97%**

Nota-se que todas as bases obtiveram os maiores níveis de scripts de sites com Alta similaridade. Tal fato é explicado pela grande presença de elementos Canvas em páginas *Web* hoje em dia e o fato das 100 consultas serem reconhecidamente ligadas a Canvas *fingerprinting*.

Um resultado não esperado ocorreu com a bases DMOZ. Dita benigna, ela obteve 91% de similaridade no nível alto. Para entender melhor as similaridades obtidas na base DMOZ, um grafo de similaridade foi montado, onde retângulos representam as consultas e os círculos brancos representam os documentos das bases de dados. Foi utilizada uma escala de cores onde vermelho representa similaridade alta (90% à 100%), amarelo representa similaridade média (60% à 89%) e verde similaridade baixa (0% à 59%).

A Figura 4 apresenta parte de um grafo de similaridade, na qual é possível visualizar 4 consultas que se relacionam com os 11 scripts da base de dados DMOZ, todas com similaridade alta variando entre 80% e 100%. Nota-se que a Tabela 5 apresenta os resultados das 4 consultas em relação aos 11 scripts da base DMOZ, destacando o nível de similaridade exposto na figura anterior.

Tabela 5. Similaridade entre 4 Consultas com a Base DMOZ

	Consultas			Consulta		Consulta
Scripts da Base	news24eg	news24zim	Scripts da Base	littleblogonthehomestead	Scripts da Base	idgconnect
pancero	0,928	0,944	kingsford	0,941	cambo	0,918
jnproductions	0,928	0,944	jnproductions	0,939	doublezoot	0,864
smoking-meat	0,923	0,939	kingsford	0,934	manningdigital	0,863
ontariogenomics	0,908	0,924	kamadojim	0,929	kingsford	0,863
atmananda	0,902	0,918	simonsmd	0,927	simonsmd	0,862

© 2017 Sociedade Brasileira de Computação

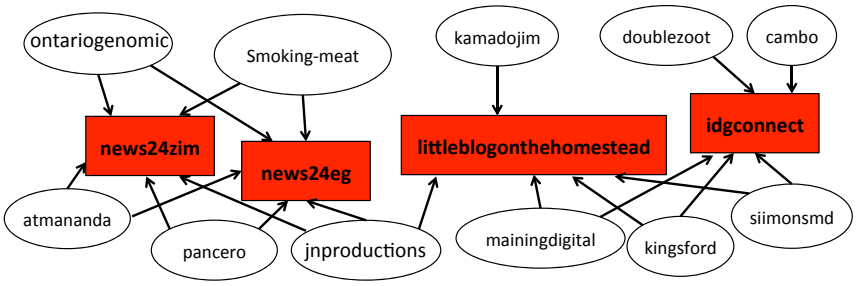


Figura 4. Grafo Parcial do Nível de Similaridade para 4 Consultas.

7. Validação

Para validar o método proposto, utilizou-se o trabalho de Saraiva et al. [Saraiva et al. 2016], que fez uma classificação de severidade de risco de ataques *fingerprinting*. Os autores destacam duas propriedades Canvas como sendo de alta periculosidade: $Canvas.toDataUrl()$ e $Canvas.getImageData()$. O método $Canvas.getImageData()$ retorna um objeto *ImageData* que copia os dados de pixel para o retângulo especificado em área Canvas enquanto o método $Canvas.toDataURL()$ permite obter o conteúdo da tela do navegador. Ambos tem alto risco de obtenção de dados dos usuários *Web*.

A Figura 5 ilustra, para cada base de dados, a ocorrência dessas duas características Canvas de alta periculosidade.

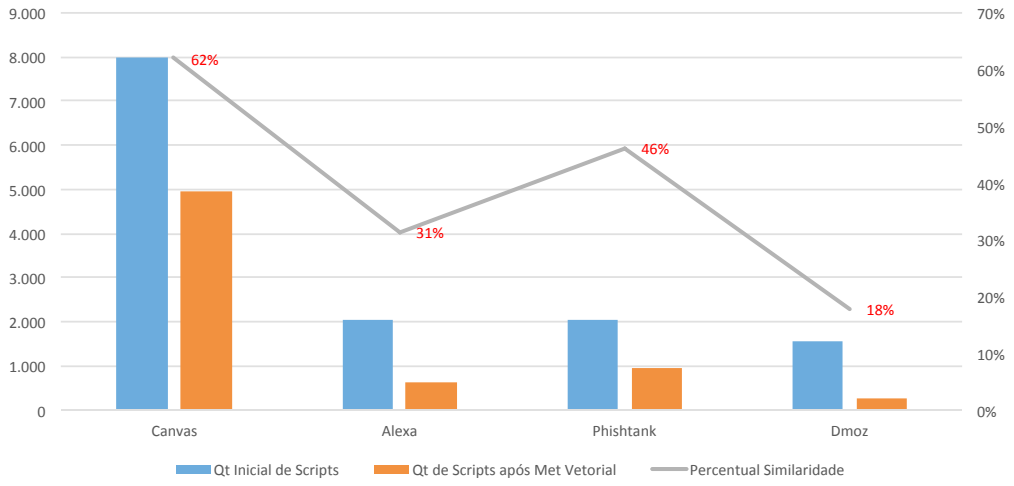


Figura 5. Gráfico do Nível de Similaridade para Duas Características Canvas

.

Os resultados mostram os percentuais de similaridade entre as bases de dados originais (quantidade original de scripts) e a aplicação do modelo vetorial. Após a aplicação do método vetorial, obteve-se 62% de similaridade (utilizando-se apenas as duas características mencionadas no trabalho de [[Saraiva et al. 2016]), com níveis de similaridade variando entre 0.1 e 1.0.

Os valores para as bases Alexa, Phisitank e DMOZ são, respectivamente, 31%, 46% e 18%. Assim, percebe-se que as bases "maliciosas" destacam-se neste critério.

 © 2017 Sociedade Brasileira de Computação

Essa validação serviu para provar que o método proposto é capaz de detectar as características Canvas *fingerprinting* nos scripts das páginas *Web*. Diante do exposto, pode-se afirmar que o método proposto nesta pesquisa, o qual tem a finalidade de ranquear pelo nível de similaridade os scripts das bases de dados, realizando as comparações com consultas Canvas *fingerprinting*, alcançou o objetivo inicial da pesquisa.

8. Considerações Finais

Este artigo propôs um método para analisar *scripts* de Canvas *fingerprinting* em páginas *Web* e informar aos usuários o nível de similaridade entre a página e as consultas que poderão ser prejudiciais a sua privacidade. Deste modo, o conjunto de características (propriedades e métodos) Canvas *fingerprinting* foram avaliados e o nível de similaridade foi calculado.

Para tanto, quatro bases de dados foram averiguadas sendo esta, uma maneira de provar a existência do Canvas *fingerprinting* nos *scripts* das páginas *Web* analisadas. Por meio de testes, em três cenários, os resultados demonstram que há um alto nível de similaridade entre as quatro bases de dados e as consultas Canvas.

8.1. Trabalhos Futuros

A comunidade de segurança no âmbito mundial, frequentemente têm realizado pesquisas sobre *fingerprinting* por meio de novos experimentos, proporcionando não só a sociedade, mas principalmente a comunidade científica inovações tecnológicas. Assim, esta pesquisa elenca alguns trabalhos futuros como forma de retomada do assunto, conforme destaque a seguir:

- Utilizar técnicas de entropia para verificar as características (propriedades e métodos) que possuam uma maior relevância para a detecção de *fingeprinting*;
- Manter e categorizar uma base de scripts para disponibilizar a comunidade acadêmica nos seus estudos referentes ao *fingerprinting*;
- Propor um mecanismo para verificar características de *fingerprinting* ofuscadas em sites *Web*.

Referências

Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., and Diaz, C. (2014). The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. *Proceedings of the 2014 ACM SIGSAC CCS*, pages 674–689.

Baeza-Yates, R. &. R.-N. (2013). *Recuperação de Informação*. Bookman.

Bursztein, E., Malyshev, A., Pietraszek, T., and Thomas, K. (2016). Picasso: Lightweight Device Class Fingerprinting for Web Clients. *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 93–102.

Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and Smith, R. (2013). Privacy Considerations for Internet Protocols. RFC 6973 (Informational). `http://www.ietf.org/rfc/rfc6973.txt`.

Eckersley, P. (2010). How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg. Springer-Verlag.

© 2017 Sociedade Brasileira de Computação

Englehardt, S. and Narayanan, A. (2016). Online Tracking: A 1-million-site Measurement and Analysis. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, (1):1388–1401.

ISO (2008). *ISO/IEC 27005: Information technology-Security techniques -Information security risk management*. ISO.

Laperdrix, P., Rudametkin, W., and Baudry, B. (2016). Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, pages 878–894.

Mayer, J. R. (2009). Internet Anonymity in the Age of Web 2.0. *Thesis*, pages 1–103.

Mowery, K. and Shacham, H. (2012). Pixel Perfect : Fingerprinting Canvas in HTML5. *Web 2.0 Security & Privacy 20 (W2SP)*, pages 1–12.

Nakibly, G., Shelef, G., and Yudilevich, S. (2015). Hardware Fingerprinting Using HTML5. *Computing Research Repository (CoRR)*, abs/1503.0.

Nikiforakis, N., Kapravelos, a., Joosen, W., Kruegel, C., Piessens, F., and Vigna, G. (2013). Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. *2013 IEEE Symposium on Security and Privacy*, pages 541–555.

Ramiro, T. B., Oliveira, E., Azevedo, L. L., Monteiro, V., and Teixeira, S. (2005). Atribuindo títulos de assuntos na categorização automática de documento. *Congresso Brasileiro de Biblioteconomia, Documentação e Ciência da Informação*.

Saraiva, A. R., de Oliveira, A. M., and Feitosa, E. L. (2016). Determinando o Risco de Fingerprinting em Páginas Web. *Anais do XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2016*, pages 254–267.

Saraiva, A. R., Elleres, P. A. d. P., Carneiro, G. d. B., and Feitosa, E. L. (2014). Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. *Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2014*, pages 50–97.

Satopää, V., Albrecht, J., Irwin, D., and Raghavan, B. (2011). Finding a "kneedle"in a haystack: Detecting knee points in system behavior. *Proceedings - International Conference on Distributed Computing Systems*, pages 166–171.

W3C (2015). Html canvas 2d context. `https://www.w3.org/TR/2dcontext/`.

Ximenes, P., Correia, M., Mello, P., Carvalho, F., Franklin, M., and Andrade, R. (2016). TARP Fingerprinting: Um Mecanismo de Browser Fingerprinting Baseado em HTML5 Resistente a Contramedidas. *Anais do XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 100–113.

 © 2017 Sociedade Brasileira de Computação

Resumos Estendidos - Seção Técnica 1 - Malware

© 2017 Sociedade Brasileira de Computação

Static Analysis on Disassembled Files: A Deep Learning Approach to Malware Classification

Dhiego Ramos Pinto[1], **Julio Cesar Duarte**[1]

[1]Seção de Ensino de Engenharia de Computação – Instituto Militar de Engenharia
Praça Gen. Tibúrcio, 80 - Praia Vermelha, Rio de Janeiro - RJ, 22291-270

{dhiegorp,duarte}@ime.eb.br

Abstract. *The cybernetic environment is hostile. An infinitude of gadgets with access to fast networks and the massive use of social networks considerably raised the number of vectors of malware propagation. Deep Learning models achieved great results in many different areas, including security-related tasks, such as static and dynamic malware analysis. This paper details a deep learning approach to the problem of malware classification using only the disassembled artifact's code as input. We show competitive performance when comparing to other solutions that use a higher degree of knowledge.*

1. Introduction

The ascending market acceptance of cloud computing, the widespread use of social networks, smartphones and other devices connected to the Internet, considerably increased the vectors of malware propagation throughout the years [Jang-Jaccard and Nepal 2014]. With the expansion of the number of users and devices, it is possible to observe an increase on the number of attacks [Verison 2016, Symantec 2016, McAfee 2015, McAfee 2014]. Generally speaking, malware can be categorized as a class of threats, like viruses, trojans, ransomware, spyware, worms, and bots, among others, whose objective, in a nutshell, is the subversion of a system, or part of a system, in which it was inserted in a non-authorized manner [McGraw and Morrisett 2000]. Those threats employ all sorts of techniques to self-propagate and attack, exploiting vulnerabilities on hardware, operational systems, networks, and applications, thus becoming the primary weapon for on-line illicit activities [Jang-Jaccard and Nepal 2014, Damshenas et al. 2013].

Malware analysis is the process by which malicious code is detected and classified, and its potential impact discovered. Detection of malicious code is a challenging task that usually is long and meticulous, requiring a specialist that uses all sorts of tools and techniques to analyze a program. It's typical to malware developers to augment their codes with evasion procedures that can emulate innocuous program's behaviors or creates difficulties to the investigation process, like code obfuscation, polymorphic code, encryption and more [Mangialardo and Duarte 2015, Marpaung et al. 2012]. Besides that, the growing volume of programs to be investigated surpasses the human limits, creating a delay between the analysis procedure and definitive treatment. Hence, the demand for autonomous and intelligent tools becomes evident.

Machine learning algorithms have been widely used to solve malware's classification and detection problems with a noted rate of success. Traditional machine learning methods are heavily dependent on feature engineering, currently requiring a lot of experimentation and a strong knowledge about the applied domain. However, Deep Learning

 © 2017 Sociedade Brasileira de Computação

models are known to learn hierarchical representations directly from data, achieving success superior to traditional methods in different areas of application [LeCun et al. 2015]. This work presents a methodology to employ Deep Learning Autoencoders to learn representative features directly from operational codes (opcodes) extracted from disassembled Windows Portable Executable (PE) files from Microsoft Malware Classification Challenge dataset [Microsoft 2015]. Our best deep neural network model achieved an overall accuracy of 99,55%, a competitive result in relation to the Challenge's winner performance.

2. Malware Analysis

Malware Analysis is an investigation process that seeks to detect and classify malicious codes and identify its goals. It can be characterized as Static or Dynamic depending on the objectives and the set of techniques used to perform the task. The Static Analysis does investigations on source code or file structure level. To accomplish this, an analyst can use anti-viruses tools, file identification hashes, string information, analysis of API calls or file headers and, ultimately, the use of tools for reverse engineering – analyzing the disassembled code. It is important to notice that during this process, the program is not executed; the objective is to extract information directly from the file and the data available within it. Otherwise, on Dynamic Analysis, the focus is on the behavior of a running program. In this approach, a program is executed in a controlled virtual environment, a sandbox, where its actions are monitored [Mangialardo and Duarte 2015]. This work focuses on Static Analysis, which is simpler and quicker to achieve.

3. Deep Learning Autoencoders

An Autoencoder or Autoassociator is a feed-forward neural network, trained in unsupervised manner, with the goal to approximate an identity function, in other words, given the input data, the network will try to learn how to copy the input values to its output. It is divided into two functions: an encoder, $h = f(x)$, that is responsible for codifying the input x into a new representation h, and a decoder, $r = g(h)$, which tries to reconstruct the original data from the hidden layer.

An approach to learning a useful representation is restricting the hidden layer dimension to be smaller than the size of the input dimension. With a constrained h, the network would learn the most salient features in order to reconstruct x with minimal error. An autoencoder with these characteristics is called undercomplete. The opposite approach, the overcomplete, in which h is not constrained by the network's input dimension can be used with the risk that, given the extended capacity, the network could copy the input data without learning useful features. In this case, other regularization techniques must be applied, such as a sparsity penalty function applied upon h (sparse autoencoders) or corrupting x and training the autoencoder to undo the corruption (denoising autoencoders) [Goodfellow et al. 2016].

In order to pre-initialize a MLP, a Deep Autoencoder can be assembled in two ways: The first, like a shallow autoencoder version, but not restricted to use just one hidden layer; or, second, using the stacking technique, in which shallow autoencoders, with just one hidden layer, are trained individually and their results' representations are used as the input to another autoencoder to be trained on and so forth [Vincent et al. 2010,

© 2017 Sociedade Brasileira de Computação

Bengio et al. 2013, Goodfellow et al. 2016]. The union of these learned representations forms a deeper autoencoder.

4. Related Work

In recent years we have seen the use of Deep Learning architectures applied to Malware detection and classification, generally benefiting from previous knowledge that helps to achieve the task. [Hardy et al. 2016] developed a framework for malware detection. The system is divided into two phases: Extraction and conversion of Windows API calls from PE files into 32-bit signatures and pre-training Stacked Autoencoders to further classification. The method showed superior performance in relation to traditional machine learning methods. [Yuan et al. 2014] used features acquired via static and dynamic analysis applied to Deep Belief Networks to detect malware on Android applications. The approach achieved the best accuracy over traditional methods. [Dahl et al. 2013] developed a large-scale malware detection system, based on Deep Belief Networks and Random Projections for dimensionality reduction on selected features. The best result was attained with an ensemble of nine neural networks.

5. Autoencoders for Malware Classification

Here, we present details about an approach to malware static analysis employing a deep autoencoder to pre-initialize a deep multilayer perceptron network for classification of malwares into families. This process can be seen as a pipeline, from the extraction of raw data of the disassembled files until the execution of the trained neural network for prediction's evaluation.

5.1. "Bag of Opcodes", Vectorization and TF-IDF weighting

After extracting the byte opcodes from each disassembled file, a bag-of-opcodes (bag-of-words like) for unigrams and bigrams was generated and its terms were subsequently counted and stored as vectors.

The *Term Frequency - Inverse Document Frequency (TF-IDF)* weighting is then applied to each vector. This technique provides a way to select and analyse the opcodes by its frequencies, which can be related to multiple kinds of malware, acting like some sort of "signature" for malicious code. The TF-IDF is computed for a term t by the formula $tfidf\,(d,t) = tf(t) \times idf\,(d,t)$, where IDF is calculated as $idf(d,t) = \log\left[\frac{(1+n)}{1+df(d,t)}\right] + 1$, n is the total number of documents and $df(d,t)$ is the number of documents that contains the term l [Pedregosa et al. 2011, Manning et al. 2008]. This new vector representation is used to train and validate our networks.

Following a hold-out cross-validation strategy to evaluate the models, the generated vectors are randomly assigned to a particular subset, training or validation, guaranteeing that 25% of the samples from each malware class are present on validation subset.

5.2. Models

On the next phase, Deep Autoencoder networks are configured, trained and evaluated upon the experimentation set. The pre-trained Autoencoder's parameters are used to produce a new network, a Multilayer Perceptron (MLP), which is trained and evaluated on

 © 2017 Sociedade Brasileira de Computação

the same dataset with the objective of classification. In this approach, we choose to implement deep autoencoders due to it's architectural similarities with a MLP, thus becoming simple to use the trained weights from the former to initialize the parameters of the latter. All autoencoders produced in the experiments falls in the category of undercomplete autoencoders. Undercomplete autoencoders compress highly dimensional data into lower dimensional representations, forcing the network to learn high quality features directly from input data [Goodfellow et al. 2016]. The choices for the number of layers and number of neurons per layer in each model were based on previous experimentations in which we choose to detail in this study only the deepest networks that achieved an interesting result.

6. Experiments

This section describes the neural networks architecture and hyper-parameters used over the experiments. All experiments were programmed using Keras Framework with Tensorflow as backend [Chollet et al. 2015, Abadi et al. 2015]. When not mentioned, it is assumed that the neural network layers' parameters used default values initialized by the framework. Scikit-learn library was employed on the n-grams extraction, terms counting, vectorization and TF-IDF weighting [Pedregosa et al. 2011].

6.1. Dataset

In 2015, Microsoft sponsored a challenge - Microsoft Malware Classification Challenge (BIG 2015) - through Kaggle website where the goal was to classify the provided malware samples into nine different families [Microsoft 2015]. The samples were divided into two sets: the training set, in which every sample was labeled and a validation set containing unlabeled samples. The training set contains two files per sample ID: a ".byte" that contains a hexadecimal representation of the malware's binary without the PE header and a ".asm" with disassembled code and various information extracted from the binary by a professional disassembler program - IDA Pro Disassembler [Hex-Rays 2017].

The distribution of samples throughout the malware families in the original set is as follows (Class – Number of Samples): 1) *Ramnit* – 1541, 2) *Lollipop* – 2478, 3) *Kelihos_ver3* – 2942, 4) *Vundo* – 475, 5) *Simda* – 42, 6) *Tracur* – 751, 7) *Kelihos_ver1* – 398, 8) *Obfuscator.ACY* – 1228 and 9) *Gatak* – 1013, accumulating 10868 samples in total. For the sake of experimentation, only samples of three classes in the original labeled dataset were selected to create our experimentation set, composed only of the disassembled ".asm" files. The criteria adopted to choose these classes were to use the samples from the two most frequent families, *Lollipop and Kelihos_ver3*, and from the rarer one, in this case, *Simda*. Given this selection, for each file, we execute an extraction procedure where only the hexadecimal bytes that represent the *opcodes* and their parameters are separated to another file (Figure 1).

6.2. Performance Measures

The Performance Measures selected to evaluate the experiments are: number of **True Positive (TP)** samples, number of **True Negative (TN)** samples, number of **False Positive (FP)** samples, number of **False Negative (FN)** samples , **Accuracy**, **Precision**, **Recall** and **F1 Score**.

© 2017 Sociedade Brasileira de Computação

```
.text:00419BA4 8B 5D 08                mov     ebx, [ebp+arg_0]
.text:00419BA7 8D 5C 9D E0             lea     ebx, [ebp+ebx*4+var_20]
.text:00419BAB 8B 33                   mov     esi, [ebx]
.text:00419BAD 8B CE                   mov     ecx, esi
.text:00419BAF 23 CF                   and     ecx, edi
```

Figure 1. Excerpt of a disassembled file from the original dataset [Microsoft 2015]. Byte opcodes and parameters are in bold.

6.3. Networks' topologies and parameters

In respect to the topological aspect, each experiment had a particular configuration for the number of layers and the number of neurons, or units, per layer. Three autoencoder networks were created considering the opcodes vectors, two for unigrams and one for bigrams vectors, respectively (**Figure 2**). In the experiment ***"Unigram 1"***, the autoencoder has 19 hidden layers with input and output layers having 96 units. The MLP was built using the trained encoder parameters adding another layer for classification, resulting in a network with 12 layers in total (96, 86, 76, 66, 56, 46, 36, 26, 16, 9, 3 and 3 units). For ***"Unigram 2"***, the autoencoder has 59 hidden layers with input and output layers having 96 units. The MLP built has 32 layers (96, 93, 90, 87, 84, 81, 78, 75, 72, 69, 66, 63, 60, 57, 54, 51, 48, 45, 42, 39, 33, 30, 27, 24, 21, 18, 15, 12, 9, 6, 3 and 3 units) . The last architecture, for experiment ***"Bigram"***, the autoencoder has 19 hidden layers, with input and output layers having 9216 units. The MLP built has 31 layers (9216, 8216, 7216, 6216, 5216, 4216, 3216, 2216, 1216, 600, 3 and 3 units). Different setups were also evaluated on smaller subsets of the dataset, but these were the ones that showed best initial results.

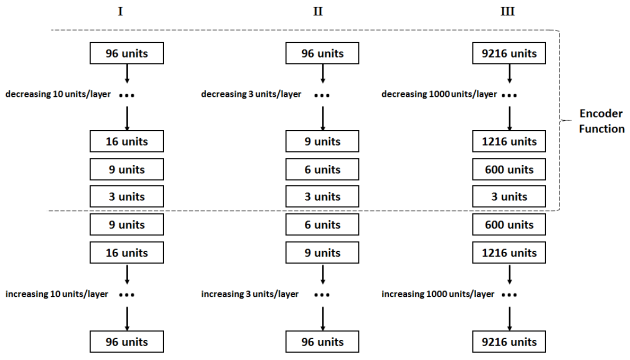


Figure 2. Autoencoders' Topologies for I. Unigram 1; II. Unigram 2 and III. Bigram.

The *Rectified linear unit* (ReLU) activation function was used for all layers for the three autoencoders and posteriorly on the MLPs. ReLU overcome the use of traditional sigmoidal activation functions for deep neural networks and helped to achieve better results in several datasets in the most diverse domains [Schmidhuber 2015]. The Sigmoid function was choosed for the MLP's classification layer. The Loss function used for autoencoders was the *Mean Squared Error* and the *Cross-entropy* for the MLPs. With regards to the training and validation phase, we configured a loop of 1,000 epochs, using mini-batches of 32 samples. Early stop criteria was adopted, based on the minimization of the loss value measured on network's validation, with a delta of at least 0.01, for autoen-

 © 2017 Sociedade Brasileira de Computação

coders, and the maximization of the accuracy value measured on network's validation, with a delta of at least 0.01, for the MLPs. The training stops if the criteria analyzed were not reached in a span of a hundred epochs. The optimizer used was the *Stochastic Gradient Decent* (SGD), with a learning rate value of 0.01.

6.4. Results

After training all deep autoencoders and assembling all classifiers, we executed the MLP training and evaluation, resulting in the collected measurements as shown in Table 1.

Table 1. Performance measurements by class.

Experiment	Classes	TP	TN	FP	FN	Recall	Precision	Accuracy	F1 Score
Unigram 1	Lollipop	617	747	0	3	99,51%	100,00%	99,78%	99,75%
	Kelihos_ver3	736	617	14	0	100,00%	98,13%	98,97%	99,05%
	Simda	0	1356	0	11	0,00%	–	99,19%	0,00%
Unigram 2	Lollipop	0	747	0	620	0,00%	–	54,64%	0,00%
	Kelihos_ver3	736	0	631	0	100,00%	53,84%	53,84%	69,99%
	Simda	0	1356	0	11	0,00%	–	99,19%	0,00%
Bigram	Lollipop	619	747	0	1	99,83%	100,00%	99,92%	99,91%
	Kelihos_ver3	736	623	8	0	100,00%	98,92%	99,41%	99,45%
	Simda	3	1355	1	8	27,27%	75,00%	99,34%	40,00%

The first experiment, that used unigram features, ***Unigram 1***, showed good results in classifying samples from the *Lollypop* and *Kelihos_ver3* families – mistaken three samples from the former and classifying every sample from the latter – the dataset's most representative families, although this network was not capable to classify any of the *Simda* examples provided.

Unigram 2, the deepest architecture tested, achieved the worst performance over all experiments. It classified every sample of the dataset as a member of the *Kelihos_ver3* family.

However, the network trained with bigram features – ***Bigram experiment*** – achieved the best results for all families in the experimentation dataset, being capable of classifying almost every *Lollipop* sample, misclassifying just by one sample, and every *Kelihos_ver3* sample provided. It stills correctly classified three examples of *Simda*, the rarest family of the dataset, as seen in subsection 6.1.

In comparison with the challenge's winner's approach, our 'Bigram' model showed a good performance, with the winner's solution achieving 99,83% of average accuracy against an average accuracy of 99.55% of our model. To the best of our knowledge, it was not possible to proper compare our results by class against the winner's solution, using other performance measure than the accuracy. Despite that fact, our 'Bigram' model showed great rates for the two most frequent classes, *Lollypop* and *Kelihos_v3*, with a recall of 99,83% and 100,00% respectively, showing a diminutive number of false positives and false negatives as can be seen in Table 1. The winner's approach relied heavily on feature selection and engineering and used a 4-fold cross validation over the same labeled dataset used in our experiments. Respecting the differences in our dataset, that used just samples from three classes of the original dataset, our approach showed competitive results and indicates a promising solution to the malware classification problem.

© 2017 Sociedade Brasileira de Computação

7. Conclusion

This paper presents a methodology using byte opcodes and their parameters as features in the form of a "bag-of-opcodes" with TF-IDF weighting extracted from disassembled malware files, applied to Deep Autoencoders for pre-training Multi-layer Perceptrons to classify malwares into families. It was shown in a series of experiments that even without any former knowledge about specificities of each malware and even losing semantic content when analyzing opcodes as just counts and frequencies of terms in a document, allied with the high capability of deep learning models in learning complex patterns and hierarchical representations that this method could actually achieve promising results, with our best experiment attaining an overall accuracy of 99.55%.

Furthermore, there are still possibilities where this work could be extended. For future work, we hope to run new experiments with the complete dataset, gathering other 'n-gram' terms. Moreover, in order to improve our results, we expect to run a specificity analysis to find the best parameters for our network models. In order to better evaluate the generalization capacity of our approach we will execute k-fold cross validation on future experiments. Another possibility is experimenting with word embedding techniques to try to extract semantic content from the disassembled instructions. Other deep learning architectures, like Convolutional Neural Networks or Recurrent Neural Networks, are considered to be employed to solve the malware classification problem.

References

Abadi, M. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. `http://tensorflow.org/`. Software available from tensorflow.org.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`. [Online; accessed 2-August-2017].

Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3422–3426. IEEE.

Damshenas, M., Dehghantanha, A., and Mahmoud, R. (2013). A survey on malware propagation, analysis, and detection. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, 2(4):10–29.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hardy, W., Chen, L., Hou, S., Ye, Y., and Li, X. (2016). Dl4md: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Mining (DMIN)*, page 61. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Hex-Rays (2017). IDA Pro Disassembler. `https://www.hexrays.com/products/ida/index.shtml`. [Online; accessed 2-August-2017].

 © 2017 Sociedade Brasileira de Computação

Jang-Jaccard, J. and Nepal, S. (2014). A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, 80(5):973–993.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Mangialardo, R. J. and Duarte, J. C. (2015). Integrating static and dynamic malware analysis using machine learning. *IEEE Latin America Transactions*, 13(9):3080–3087.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). Scoring, term weighting and the vector space model. *Introduction to information retrieval*, 100:2–4.

Marpaung, J. A., Sain, M., and Lee, H.-J. (2012). Survey on malware evasion techniques: State of the art and challenges. In *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, pages 744–749. IEEE.

McAfee (2014). Net losses: Estimating the global cost of cybercrime. intel security (mcafee). `http://www.mcafee.com/us/resources/reports/rp-economic-impact-cybercrime2.pdf`. [Online; accessed 2-August-2017].

McAfee (2015). Previsões do mcafee labs sobre ameaças em 2016. `http://www.mcafee.com/br/resources/reports/rp-threats-predictions-2016.pdf`. [Online; accessed 2-August-2017].

McGraw, G. and Morrisett, G. (2000). Attacking malicious code: A report to the infosec research council. *IEEE Softw.*, 17(5):33–41.

Microsoft (2015). Microsoft Malware Classification Challenge (BIG 2015). `https://www.kaggle.com/c/malware-classification`. [Online; accessed 2-August-2017].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Symantec (2016). Internet Security Threat Report. `https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf`. [Online; accessed 2-August-2017].

Verison (2016). 2016 data breach investigations report. `http://www.verizonenterprise.com/verizon-insights-lab/dbir/2016/`. [Online; accessed 2-August-2017].

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408.

Yuan, Z., Lu, Y., Wang, Z., and Xue, Y. (2014). Droid-sec: Deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM.

© 2017 Sociedade Brasileira de Computação

Mitigando a Evasão de Instrumentadores Binários Dinâmicos

Ailton Santos Filho[1], **Arthur Binda Alves** [1], **Isaque Vieira** [1], **Eduardo L. Feitosa**[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
CEP 69.077-000 – Manaus – AM – Brasil

{assf,aba,ibv2,efeitosa}@icomp.ufam.ed.br

Abstract. *This paper presents countermeasures against the most recent evasion techniques in Dynamic Binary Instrumentation (DBI). They use features of the Intel Pin instrumentation tool to ensure the transparency of the analyzes. An evaluation of proposed countermeasures in comparison to state-of-the-art techniques was performed. The current limitations and the future of this work are also presented.*

Resumo. *Este artigo apresenta contramedidas contra as mais recentes técnicas de evasão (analysis-aware) em Instrumentação Binária Dinâmica (DBI). Elas utilizam recursos do instrumentador Intel Pin para garantir a transparência das análises. Uma avaliação das contramedidas propostas em comparação as tecnicas tidas como estado da arte foi realizada. As limitações existentes e o futuro desse trabalho também são apresentados.*

1. Introdução e Motivação

No atual estágio da "guerra" contra *malwares*, os *frameworks* de Instrumentação Binária Dinâmica (*Dynamic Binary Instrumentation*, DBI) têm sido soluções empregadas na análise de segurança de aplicações. Isto porque tais ferramentas permitem modificar ou adicionar código de máquina em programas ou aplicações e assim avaliar o desempenho e a corretude, detectando falhas em tempo de execução. Por isso, eles vêm sendo utilizados na detecção de *shellcode* [Mohaisen et al. 2015], *taint analysis* [Stamatogiannakis et al. 2014], análise de código auto-modificante [Giacobazzi et al. 2014], entre outros.

Infelizmente, à medida que os *framework* DBI ganharam popularidade, os *malwares* incorporaram técnicas para detectar quando estão sendo instrumentados (analisados ou avaliados). Assim, começaram a incorporar técnicas de evasão[1] em seus códigos, que os permitem reconhecer quando estão sendo executados em um ambiente real ou em um ambiente de instrumentação.

A ideia é que, usando *APIs* (*Application Programming Interface*) do sistema operacional, o *malware* seja capaz de perceber quando está sendo instrumentado e, assim, possa mudar seu comportamento ilegítimo. *Malwares* com esse perfil de funcionamento são conhecidos como aplicações conscientes de análise (*analysis-aware applications*) [Rodriguez et al. 2016]. Foi com base nesse conceito que Rodriguez et *al.* propuseram uma ferramenta (PinVMShield) para detectar técnicas de evasão

[1]Neste trabalho, o termo técnica de evasão engloba todos os métodos usados por *malwares* para evitar sua detecção, análise e compreensão.

 © 2017 Sociedade Brasileira de Computação

usadas por *malwares* conscientes da análise em ambientes *sandbox* ou virtualizados. A PinVMShield, baseada no instrumentador Pin [Intel 2016], é uma ferramenta capaz de enganar binários maliciosos e evitar que técnicas de detecção e evasão de VMs-*sandbox*, ditas *analisys-aware*, logrem êxito.

Recentemente, Sun et *al.* [Sun et al. 2016] demonstraram, em uma apresentação na Black Hat, entitulada "Break Out of The Truman Show", novas técnicas de evasão independentes das APIs do sistema operacional. Essas técnicas permitem que um *malware*, ao tomar ciência de que está sendo instrumentado, faça uso do *framework* DBI para executar ações maliciosas fora do contexto de análise do instrumentador, obviamente sem ser detectado. Entretanto, ao analisar as técnicas propostas por Sun et *al.*, e outras técnicas mais antigas, percebeu-se que algumas contramedidas podem ser tomadas. É neste contexto que este trabalho propõe contramedidas para evitar técnicas de evasão em *frameworks* DBI. Para tanto, empregando a ferramenta PVMShield como base, novas mitigações foram implementadas, incluindo para as técnicas de evasão propostas por Sun et *al.*.

As contribuições deste trabalho são duas. A primeira é a proposição e prova de técnicas anti-evasão de DBI, baseadas em duas das ideias de Sun et *al.*: *Code Cache* - região da memória onde os trechos de código instrumentados ficam armazenados - e *Thread Local Storage* (TLS) - que permite que *threads* de um mesmo processo tenham acesso a uma estrutura de dados comum, capazes de mitigar até os métodos de evasão do estado da arte. A segunda é o desenvolvimento de um protótipo para a defesa da transparência de instrumentadores binários dinâmicos com todo código fonte disponível[2].

2. Evadindo Instrumentadores Binários Dinâmicos

Dentre as técnicas de evasão apresentadas em [Sun et al. 2016], este trabalho foca em mitigar a detecção por *Code Cache* e por *Thread Local Storage*. A técnica de detecção por *Thread Local Storage* foi selecionada pela sua simplicidade de funcionamento, além de ser claramente uma técnica de evasão baseada no uso de APIs do Windows. Já a detecção por Code Cache foi escolhida devido ao seu impacto nas soluções que utilizam *frameworks* DBI, como o Pin, uma vez que permite a subversão do fluxo de execução dos instrumentadores, como mostrado por Sun et *al.*

2.1. Detecção por *Code Cache*

Quando *frameworks* DBI realizam análises em tempo de execução, não permitem que o código original da aplicação sendo instrumentada seja executado. Os trechos de código executados são os provenientes de regiões conhecidas como *Code Cache*, regiões especiais, alocadas pelos *frameworks* DBI, que contêm o código do programa alvo após ser modificado pelo instrumentador.

Sun et *al.* perceberam que os *Code Caches* do Pin possuem características diferentes em relação à outras páginas de memória, o que permite sua identificação. São elas: (i) As permissões da página de memória são sempre de execução, leitura e escrita; (ii) Os dados alocados no início de cada *Code Cache* são sempre valores hexadecimais pré-definidos (0xFEEDBEAF); (iii) O tamanho da página de memória é fixo em um valor pré-determinado (0x40000 é o valor padrão).

[2] `https://github.com/ailton07/PinVMShield`

© 2017 Sociedade Brasileira de Computação

Além disso, também perceberam que o Pin aloca *Code Caches* de acordo com sua necessidade no espaço de endereços do processo alvo, o que cria duas outras formas de detectar suas *Code Caches*. A primeira forma é executar intencionalmente uma sequência de instruções que ocorra apenas uma vez durante a execução do programa e checar todo o espaço de endereços do seu processo, a fim de contabilizar o número de ocorrências da sequência. Se houver mais de uma ocorrência, existe um *Code Cache*. Essa técnica foi nomeada como Detecção de *Code Cache* por código. A Figura 1 ilustra a prova feita por Sun et al. da detecção de *Code Cache*.

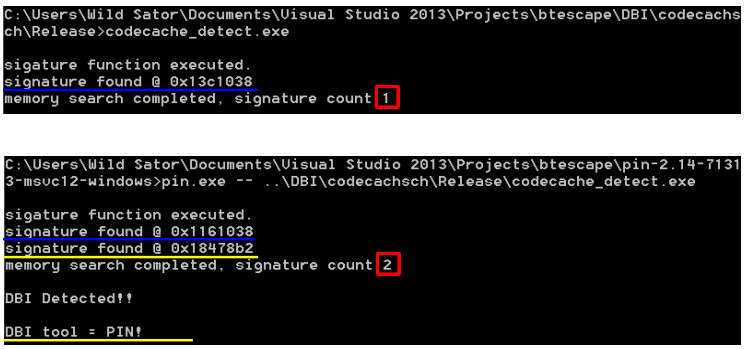


Figura 1. Exemplo de Detecção do *Code Cache*. Fonte: [Sun et al. 2016]

A segunda, nomeada de Detecção de *Code Cache* por dados, assume que a aplicação executaria intencionalmente uma instrução com um dado conhecido único no operando da instrução e checaria em todo o espaço de endereços do seu processo o número de ocorrências do dado. Caso haja mais de uma ocorrência, isso seria um indício de *Code Cache*. Provas de conceito das técnicas de detecção de *Code Cache* por código e por dados foram desenvolvidas pelos autores deste trabalho e encontram-se disponíveis[3] para uso. Também existe uma prova de conceito baseada na detecção das três características dos *Code Caches* enumaradas acima, que encontra-se disponível [4].

2.2. Detecção por Thread Local Storage

Thread Local Storage (TLS) é uma funcionalidade que permite que threads de um mesmo processo tenham acesso a uma estrutura de dados comum na forma de um vetor de até 1088 posições [Microsoft 2017]. É um recurso largamente utilizado para reduzir a necessicade do sincronismo entre threads de um programa. Sabendo que o Pin faz uso dessa funcionalidade, Sun et *al.* demonstraram que é possível uma aplicação sendo instrumentada determinar se está sob análise através de uma consulta e da contagem do número de *slots* sendo utilizados na TLS. Uma prova de conceito das técnicas de detecção por *Thread Local Storage* foi desenvolvida pelos autores deste trabalho e encontra-se disponível[5] para uso.

Na Figura 2, o terminal a esquerda representa a execução normal de uma aplicação e o terminal a direita a mesma aplicação sendo instrumentada pelo Pin.

[3]https://github.com/ailton07/ActiveDetectionPinWithCodeCache
[4]https://github.com/ailton07/ActiveDetectionPinWithFEEDBEAF
[5]https://github.com/ailton07/TLS_DetectPin

 © 2017 Sociedade Brasileira de Computação

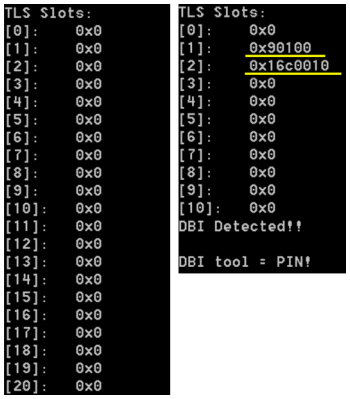


Figura 2. Detecção do Pin via TLS. Fonte: [Sun et al. 2016]

3. Implementação

Este trabalho detecta e mitiga técnicas de evasão por *Code Cache* e por TLS. Seu desenvolvimento foi voltado para o instrumentador Intel PIN por ser o *framework* DBI com menor custo computacional imposto sobre a aplicação instrumentada [Luk et al. 2005] e na qual a *PinVMShield* foi empregada para viabilizar a implementação das contramedidas propostas.

3.1. Contramedida para Detecção por *Code Cache*

Em seu artigo, Sun et *al.* [Sun et al. 2016] apresentam uma série de técnicas de evasão de DBI por detecção de *Code Cache*. A primeira se foca em determinar se uma página de memória do processo é um *Code Cache* a partir de um conjunto pré-determinado de características. Em outras palavras, as permissões de página de memória são uma característica utilizada para confirmar a existência de um *Code Cache*. De forma prática, eles argumentam que a principal maneira de se obter o estado atual das permissões de uma página de memória é através da API do Windows *VirtualQuery*.

A contramedida proposta contra essa forma de detecção de *Code Cache* emprega a função *ReplaceWinAPI*, do PinVMShield, para alterar os retornos das chamadas a *VirtualQuery*, a fim de não retornar as informações reais de um *Code Cache*. Vale ressaltar que embora Sun et *al.* comentem que podem existir outras maneiras de se determinar as permissões de uma página de memória, essas permissões não são um critério suficiente por si só para identificar um *Code Cache*.

A segunda forma de verificar a presença de um *Code Cache* é através do tamanho da página de memória, empregando novamente a API *VirtualQuery* para determinar o tamanho da página de memória. A contramedida proposta também utiliza a função *ReplaceWinAPI* para impedir que o processo sendo instrumentado tome conhecimento do real tamanho da página de memória do *Code Cache*. O Pin fornece um conjunto APIs para mapeamento de suas *Code Caches*, entre elas destacam-se *TRACE_CodeCacheAddress* e *TRACE_CodeCacheSize*, que são utilizadas na contramedida.

A terceira forma é uma técnica de força bruta, com elevado custo computacional, que procura em todo o espaço de memória do processo por esses padrões de conteúdo que identificam um *Code Cache*. A contramedida proposta utiliza a API *INS_AddInstrumentFunction* do Pin para desenvolver um método que atua ativamente

© 2017 Sociedade Brasileira de Computação

sobre cada instrução do programa sendo instrumentado, a fim de impedir que sejam feitos acessos diretos ao espaços de memória dos *Code Caches*. É importante esclarecer que essa é uma solução de elevado custo computacional, assim como a técnica de evasão proposta por Sun et *al.*

Entretanto, sabendo do custo computacional desta técnica de evasão, os autores deste artigo propõem uma versão alternativa que não demanda o esforço computacional de procurar em todo o espaço de memória do processo. Para tanto, utiliza-se a API do Windows *VirtualQuery* para percorrer as páginas de memória, ao invés do espaço de memória, em busca do valor inicial do *Code Cache*. Contudo, neste caso, a contramedida proposta para a segunda técnica de Sun et *al.*, que utiliza a função *ReplaceWinAPI*, do PinVMShield, para impedir que a API do Windows *VirtualQuery* retorne páginas de memória de *Code Caches*, pode ser aplicada para inibir a evasão.

É importante destacar que o uso da função *ReplaceWinAPI* do PinVMShield também é a contramedida padrão para lidar com as questões apresentada na Seção 2.1: (i) das permissões da página de memória, descritas pelo campo *AllocationProtect* da estrutura *Memory Basic Information do Windows*; (ii) dos dados alocados no início de cada *Code Cache*, que são sempre os valores hexadecimais (0xFEEDBEAF, por exemplo); e (iii) do tamanho da página de memória é fixo em um valor pré-determinado (valor padrão 0x40000).

3.2. Contramedida para Detecção por Thread Local Storage

A detecção por *Thread Local Storage* (TLS) é uma técnica que detecta a presença dos *frameworks* DBI a partir da quantidade de posições sendo utilizadas no TLS. A princípio, a política utilizada por Sun et *al.* é de se um programa não utiliza a TLS, então não deveria haver nenhuma posição do TLS sendo utilizada. No entanto, foi observado que essa política pode acarretar falsos positivos.

Em seu trabalho, Sun et *al.* propuseram o uso das APIs TLS fornecidas pelo sistema operacional, sendo elas no Windows: *TlsAlloc*, *TlsGetValue*, *TlsSetValue* e *TlsFree*. Mais especificamente, a API *TlsGetValue* poderia ser utilizada para obter o valor de cada posição TLS e contabilizar quantas estão sendo utilizadas. A contramedida proposta utiliza a função da API *ReplaceWinAPI* do PinVMShield para garantir que durante as consultas ao TLS, a aplicação sendo instrumentada não seja capaz de obter os valores das posições utilizadas pelo Pin.

A alteração do retorno da API *TlsGetValue* pode ser detectada pela aplicação alvo da instrumentação, caso ela faça uso das APIs *TlsGetValue* e *TlsSetValue*. No entanto, se nem todos os espaços TLS estiverem em uso, é possível utilizar uma lógica de deslocamento de posições para garantir a transparência da técnica de mitigação. Por exemplo, caso o Pin esteja utilizando as cinco primeiras posições do TLS e a aplicação sob análise tente utilizar a primeira posição do TLS, basta mapear essa tentativa de uso para uma posição diferente das que o *framework* está utilizando.

4. Avaliação e Resultados Iniciais

Esta Seção apresenta e discute os resultados iniciais dos experimentos realizados. Para tanto, o protocolo experimental adotado e o ambiente de experimentação são detalhados. Em seguida, os resultados da efetividade de bloqueio (mitigação) das contramedidas

 © 2017 Sociedade Brasileira de Computação

propostas são discutidos. Por fim, uma análise do desempenho das contramedidas propostas em relação ao *PVMShield* é apresentada. Vale ressaltar que apenas o *PVMShield* é avaliado porque o trabalho de Sun et *al.* não possui contramedidas implementadas.

4.1. Ambientes de experimentação

Tanto os testes de bloqueio quanto às análises de desempenho foram realizados em um equipamento com Processador Intel Core i5-3230M 2.60 GHz, 4GB de memória RAM, sistema operacional Windows 7 Ultimate x64 based, compiladores Microsoft C/C++ Optimizing Compiler Version 15.00.30729.01 para x64 com Microsoft Visual Studio 2008, Microsoft C/C++ Optimizing Compiler Version 16.00.30319.01 para 80x86 com Microsoft Visual Studio 2010 e Pin 2.14 kit (71313) para MSVC10.

4.2. Eficácia

Para todas as técnicas de evasão de instrumentadores apresentadas na Seção 2 foram desenvolvidas provas de conceito funcionais capazes de detectar a presença do Pin, mesmo alvo de detecção do trabalho de Sun et *al.* Vale ressaltar que os códigos dessas provas de conceito estão disponíveis e estão listados neste artigo.

De forma direta, para cada técnica de detecção apontada, uma prova de conceito (contramedida implementada) e o *PinVMShield* (Rodriguez et *al.*) foi testada para verificar a capacidade de bloqueio. A Tabela 1 sumariza esses resultados, comprovando a eficácia das contramedidas em impedir que o Pin fosse detectado. O caractere *X* indica que a técnica de evasão não foi mitigada, enquanto o *OK* indica que a técnica de evasão foi mitigada com sucesso. Considerando que tanto o PinVMShield, quanto as técnicas de Sun et *al.* foram publicados em 2016, é esperado que o PinVMShield ainda não considerasse essas técnicas.

Tabela 1. Técnicas de Evasão impedidas pelo protótipo.

gray!50 **Técnica de Detecção**	**Pin**	**PinVMShield**	**Proposta**
Code Cache por Código proposta por Sun et al.	X	X	OK
Code Cache por Dado proposta por Sun et al.	X	X	OK
Code Cache por 0xFEEDBEAF proposta por Sun et al.	X	X	OK
Code Cache por Código alternativa	X	X	OK
Code Cache por Dado alternativa	X	X	OK
Code Cache por 0xFEEDBEAF alternativa	X	X	OK
Thread Local Storage	X	X	OK

4.3. Desempenho

Para medir os impactos das contramedidas propostas, comparando-as com a *PVMShield*, foi utilizado o SPEC CPU2006, uma suíte de *benchmarks*, aceita como padrão por indústrias do setor de computação, que se destina a testar exaustivamente o processador, sistema de memória e compilador de um sistema. Trabalhos como [Ferreira et al. 2014, Zeng 2015, Luk et al. 2005], este último dos próprios desenvolvedores do Pin, utilizam o SPEC CPU.

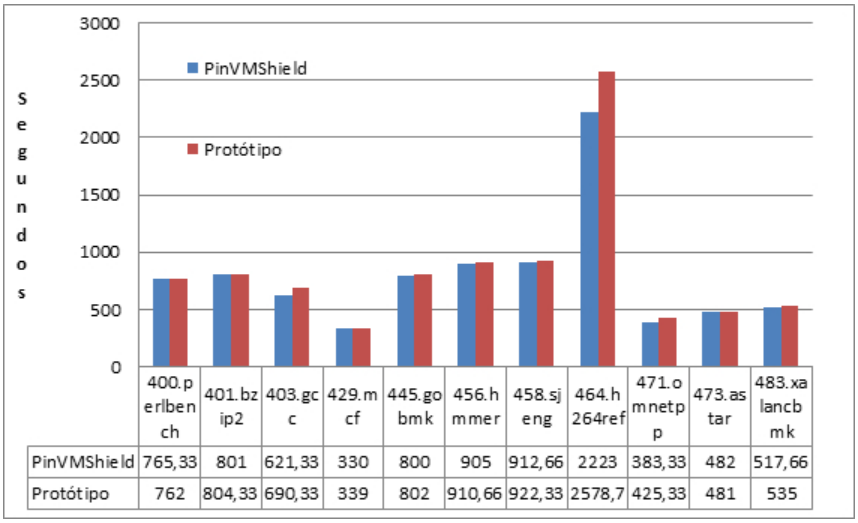
Foi utilizado o conjunto de componentes de testes SPECint, assim como em [Luk et al. 2005]. O SPECint2006 possui um conjunto de doze (12) aplicações de testes diferentes, que podem ser utilizadas para se obter métricas de desempenho. Ressalta-se

© 2017 Sociedade Brasileira de Computação

que apenas onze (11) são utilizáveis aqui, visto que a aplicação *462.libquantum* do SPEC não é aplicável em ambientes com Microsoft Visual C++, como o deste trabalho.

O SPEC foi configurado para: (i) utilizar o tamanho de entrada de dados de referência; (ii) utilizar o modo reportável que executa cada aplicação de teste três vezes; e (iii) executar em conjunto com o *PinVMShield* original sobre cada uma das onze aplicação de testes. Após isso, os experimentos foram repetidos com o SPEC configurado para executar cada aplicação de teste com as contramedidas criadas. Vale ressaltar que não foi incluída no teste de desempenho a contramedida da técnica de evasão sem uso de APIs do Windows, uma vez que os custos computacionais, da contramedida e da técnica de evasão, são impraticáveis no mundo real.

A Figura 3 apresenta a comparação dos tempos de execução para o *PinVMShield* e para as contramedidas propostas. Cada barra no histograma representa o tempo de execução de uma das abordagens para cada aplicação de teste do SPEC. Abaixo da barra tem-se o nome da aplicação e o tempo em segundos da execução.



	400.perlbench	401.bzip2	403.gcc	429.mcf	445.gobmk	456.hmmer	458.sjeng	464.h264ref	471.omnetpp	473.astar	483.xalancbmk
PinVMShield	765,33	801	621,33	330	800	905	912,66	2223	383,33	482	517,66
Protótipo	762	804,33	690,33	339	802	910,66	922,33	2578,7	425,33	481	535

Figura 3. Comparativo entre os tempos de execução do PinVMShiedl e do Prototipo.

Pode-se observar na Figura 3 que as diferenças do tempo de execução entre as abordagens são bem sensíveis. Para obter um comparativo se a diferença entre os tempos é significativa, fez-se um teste de média zero que forneceu que a média da diferença entre os tempos de execução é 24.211 segundos, com uma precisão de 95% e com o intervalo de confiança entre (-17.6593 ; 66.0815), o que nos diz que não há diferença sensível entre as duas abordagens. Para as aplicações *400.perlbench* e *473.astar* o tempo de execução do protótipo foi inferior ao tempo do PinVMShield, no entanto, não foi realizado nenhum procedimento para a otimização desses testes.

5. Andamento do Trabalho

O controle do acesso às informações do sistema continua sendo um ponto crítico para a defesa da transparência de ferramentas de análise, em especial os instrumentadores binários dinâmicos. Este trabalho, ainda em andamento, demonstrou possibilidades de combate à diferentes técnicas de evasão para DBI, inclusive as mais recentes.

© 2017 Sociedade Brasileira de Computação

Foram desenvolvidas neste trabalho um conjunto de contramedidas (todas com prova de conceito e futuramente agregadas em uma ferramenta), tomando como base o *framework PinVMShield*, para diferentes técnicas de "analysis-aware". Foi mostrado que as contramedidas desenvolvidas são capazes de evitar a ação de técnicas emergentes de evasão de instrumentadores binários dinâmicos sem acarretar perdas consideráveis de desempenho.

No que tange à continuidade deste trabalho pretende-se: (i) organizar as implementações em uma única ferramenta; (ii) realizar um estudo comparativo com ferramenta *Arancino* [Polino et al. 2017], que possui funcionalidades semelhantes a *PinVMShield*; (iii) propor novas contramedidas para técnicas existentes, focando inicialmente nas técnicas levantadas por Sun et *al.* [Sun et al. 2016] não tratadas neste trabalho; (iv) desenvolvimento de novas técnicas de evasão de instrumentadores binários dinâmicos.

Referências

Ferreira, M. T., Santos Filho, A., and Feitosa, E. (2014). Controlando a Frequência de Desvios Indiretos para Bloquear Ataques ROP. In *Anais do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2014*. SBC.

Giacobazzi, R., Simon, A., and Zennou, S. (2014). Challenges in analysing executables: Scalability, self-modifying code and synergy (dagstuhl seminar 14241). In *Dagstuhl Reports*, volume 4.

Intel (2016). Pin 2.14 user guide. `https://software.intel.com/sites/landingpage/pintool/docs/71313/Pin/html/`.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN*, pages 190–200, New York, NY, USA. ACM.

Microsoft (2017). Thread local storage. `http://goo.gl/U0etlH`.

Mohaisen, A., Alrawi, O., and Mohaisen, M. (2015). Amal: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security*, 52:251–266.

Polino, M., Continella, A., Mariani, S., D'Alessio, S., Fontana, L., Gritti, F., and Zanero, S. (2017). Measuring and defeating anti-instrumentation-equipped malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 73–96.

Rodriguez, R. J., Gaston, I. R., and Alonso, J. (2016). Towards the detection of isolation-aware malware. *IEEE Latin America Transactions*, 14(2):1024–1036.

Stamatogiannakis, M., Groth, P., and Bos, H. (2014). Looking inside the black-box: capturing data provenance using dynamic instrumentation. In *International Provenance and Annotation Workshop*, pages 155–167. Springer.

Sun, K., Li, X., and Ou, Y. (2016). Break out of the truman show. In *BlackHat*. `goo.gl/vo63g8`.

Zeng, J. (2015). Bunary code reuse: A dynamic analysis based approac. Master's thesis, University of Texas, Texas.

© 2017 Sociedade Brasileira de Computação

Detecção de Desfiguração de Sites por Similaridade em Imagens

Henry Mendes de Jesus [1], José Eduardo Malta de Sá Brandão[2]

[1]UNICEUB – Brasília – DF – Brasil

[2] Instituto de Pesquisa Econômica Aplicada - IPEA – Brasília – DF – Brasil.

{hmjbetah@gmail.com,je.brandao@ipea.gov.br}

Abstract. *This paper proposes a new approach for web page defacement detection. It's based on image analysis, identifying the symmetry between the initial state of the web page and the current state. This work differs from the others, due to its simplicity, maintaining effectiveness.*

Resumo. *Esse trabalho propõe uma nova abordagem de detecção de desfiguração de páginas web, baseada na análise de imagens, identificando a simetria entre o estado inicial da página web e o estado atual. Esse trabalho difere dos demais, devido à sua simplicidade, mantendo a eficácia.*

Introdução

Um dos ataques mais comuns na Internet é a desfiguração de página web, que de acordo com Kanti et al. (2011), "ocorre quando o intruso de uma forma maliciosa altera a página Web substituindo o seu conteúdo com material ofensivo ou provocativo". Além de ter seu sistema web comprometido, a reputação da empresa é afetada, causando muitas vezes prejuízos financeiros. Portanto, é de suma importância que haja mecanismos apropriados para detecção de ataques deste tipo.

Nesse artigo é proposto um novo mecanismo de detecção de ataques de desfiguração de páginas web, baseado na comparação de imagens obtidas a partir do sítio web monitorado.

A seguir é apresentada uma revisão da literatura contendo as principais técnicas de detecção de ataques e os principais trabalhos relacionados à proposta. A terceira seção descreve o mecanismo de detecção por similaridade de imagens, sua implementação e os resultados obtidos. Na última seção são relatadas as conclusões.

2. Técnicas de Detecção

Existem diversas maneiras de detectar desfiguração de websites, cada qual com um grau de eficiência. A seguir é apresentado um resumo das principais técnicas de detecção presentes na literatura.

 © 2017 Sociedade Brasileira de Computação

2.1. Comparação por *Hash*

Uma das técnicas mais utilizadas é a **comparação por *hash.*** Segundo Oriyano (2014), o *hashing* permite que qualquer mudança nos dados seja facilmente detectada, mesmo que em uma pequena proporção, pois o resultado será um *hash* completamente diferente do original. Um exemplo desse tipo de técnica pode ser observado no trabalho de Kanti et al. (2011). O artigo apresenta uma forma de detectar a desfiguração de páginas através de comparação de *hashes* das páginas web obtidos por meio dos algoritmos MD5 ou SHA. A proposta de Kanti et al. (2011) avalia somente do conteúdo de texto, ignorando as imagens contidas na página web. A proposta apresentada nesse artigo também usa a comparação por *hash* para analisar texto e imagens de forma única e simplificada.

2.2. Árvore DOM (HTML)

O Modelo de Objetos de Documento (*DOM - Document Object Model*) é análogo a uma árvore de objetos que compõem uma página HTML (Philippe et al. 2017). Também é uma biblioteca que permite aos desenvolvedores manipular a estrutura de um documento de hipertexto. Uma forma de conseguir detectar uma mudança, em uma página de Internet, é armazenar a estrutura e os objetos, comparando com uma versão recente. Uma varredura é feita sentido *top-down* na Árvore DOM, verificando cada objeto entre os dois arquivos. No final do processo podem ser definidas métricas, como número de elementos alterados, para verificar se houve realmente uma desfiguração.

Um exemplo de aplicação dessa técnica está no trabalho de Kumar, Rishi e Singh (2013), que teve como objetivo detectar mudanças nos componentes de uma página web. A solução proposta extrai componentes que compõem uma página web e verifica a integridade deles comparando suas versões originais. Eles ficam armazenados em um repositório de componentes web. O mecanismo é composto por quatro módulos, Sistema de Monitoração, Extração de Componentes web, Pré-processamento de Textos e Imagens, Verificação de Integridade e Repositório de Componentes web. Seu modo de detecção de páginas desfiguradas é a forma de uso da técnica de *polling*. Essa técnica é empregada periodicamente em certos grupos de dados para verificação de mudanças. Dessa forma, é verificada a integridade dos dados com algoritmos de *hash*. A proposta de Kumar, Rishi e Singh (2013) precisa armazenar muitos dados sobre os objetos monitorados, divergindo da proposta do presente artigo, que necessita armazenar poucos dados, tornando a comparação mais simples.

2.3. Comparar a diferença das requisições HTML

Outra técnica consiste em comparar o conteúdo HTML de duas requisições ao mesmo site. Como pode haver conteúdo dinâmico em cada requisição, deve-se estabelecer um limite de aceitação de mudanças. Um exemplo dessa técnica é apresentado por Verma e Sayyad (2015) a fim de impedir que uma página desfigurada seja entregue ao cliente. Para isso, eles utilizaram o servidor web Apache. O mecanismo utiliza a função *handler* do Apache para receber o conteúdo da requisição HTTP, gerando um *hash* para os arquivos relacionados ao website no servidor, que é armazenado em um banco de dados. Os *hashes* armazenados do banco de dados são comparados com os gerados em tempo de execução. Da mesma forma que o trabalho de Kanti et al. (2011), a técnica não avalia alterações em imagens.

© 2017 Sociedade Brasileira de Computação

2.4. Detecção de páginas desfiguradas em um Sistema Especialista

Nesse artigo, para resolver o problema da alta taxa de falsos positivos, os Davanzo et al. (2011) propõem uma forma Sistema Especialista. Na etapa de treinamento foram analisados três meses de amostras de 320 tipos de desfiguração de páginas web coletadas do site ZoneH.org. Dessa maneira, foi desenvolvido um protótipo de framework para realizar o processo de detecção. Em seu modo de operação, a URL do site a ser detectado é utilizada como entrada para diversos tipos de sensores, que capturam e classificam os elementos da página web na forma de vetores com valores numéricos. Esses vetores servem de entrada para a comparação utilizando a base de conhecimento.

2.5. Palavras Comuns

Quando ocorre um caso de desfiguração de páginas é comum que os atacantes deixem assinaturas. Geralmente com termos conhecidos. A técnica de detecção consiste na busca por palavras ou termos específicos, como "*Hacked*" ou "Hackeado", a partir de um dicionário. Porém, pode haver textos bem mais elaborados. Como ocorre com as técnicas analisadas anteriormente, essa também não detecta alterações em imagens.

2.6. Detecção de desfigurações pelo reconhecimento de imagens

A técnica desenvolvida por Borgolte e Kruegel (2015) propõe uma forma de detecção por meio da Inteligência Artificial. O sistema baseado em uma rede neural aprende, analisando as assinaturas que os atacantes deixam em uma página desfigurada. Isso é possível por meio de arquivos de imagem gerados através de um dos bancos de dados mais famosos de páginas desfiguradas: THC.org. Para o aprendizado as imagens obtidas são divididas em partes menores. Elas não podem ser muito pequenas para não aumentar as chances de falsos positivos. Elas não podem ser muito grandes, pois aumentaria de uma forma considerada o tempo de aprendizado. Os métodos utilizados para detecção seriam inicialmente pela captura dos logotipos utilizados por grupos de atacantes, erros tipográficos ou gramaticais, *leetspeak*, combinação de letras e combinação de cores. Portanto, a proposta de Borgolte e Kruegel (2015) busca assinaturas de ataques, enquanto a proposta do presente artigo identifica as alterações da página web, sem a necessidade de conhecer os ataques.

3. Detecção por Simetria de Imagens

A proposta apresentada nesse trabalho adota uma nova abordagem de análise de imagens, identificando a simetria entre o estado inicial da página web e o estado atual. Esse trabalho difere dos demais, devido à sua simplicidade. Não é necessário armazenar ou analisar todo o conteúdo da página web e a técnica permite avaliar páginas estáticas e dinâmicas, conforme pode ser observado a seguir.

3.1. Modelo de detecção

A técnica consiste em transformar toda a página web em uma "fotografia", como seria observada pelo usuário em seu browser. Em seguida a imagem gerada é dividida em

 © 2017 Sociedade Brasileira de Computação

partes menores e é calculado, cada uma das partes, um valor *hash*, que é armazenado em uma base de dados.

Conforme ilustrado na Figura 1, periodicamente são feitas novas capturas. Para cada captura é gerada uma nova imagem e uma nova subdivisão, usando os mesmos parâmetros da imagem original. Os valores *hash* de cada parte são comparados com os dados originais. O valor percentual da alteração da página inicial é obtido por meio da divisão do número de partes (*hashes*) diferentes, pelo total de partes da imagem. É considerado um ataque quando o valor percentual de alteração da página extrapola o indicador limite estabelecido pelo administrador.

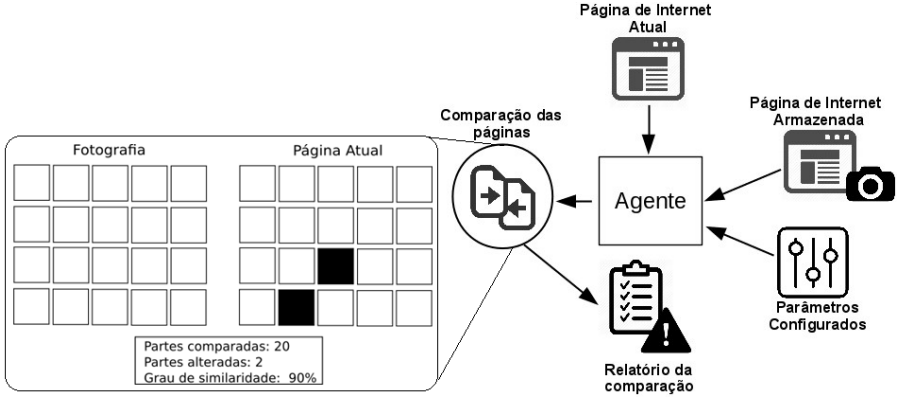


Figure 1. Mecanismo de detecção por similaridade de imagens

Além da comparação das partes da imagem, o modelo prevê uma verificação das resoluções da imagem completa. Nessa comparação, utilizam-se suas alturas e larguras como pontos de referência. Caso esses parâmetros sejam diferentes de um limite estabelecido é gerado o alerta. Essa técnica é bastante útil na identificação de ataques em páginas estáticas. Mesmo as páginas dinâmicas costumam possuir limites de tamanho, principalmente na largura.

Cabe ressaltar que a curacidade dos resultados depende das ferramentas adotadas para a geração e divisão das imagens das páginas. Todas as imagens deverão ser obtidas usando as mesmas ferramentas, com os mesmos parâmetros de configuração, para evitar distorções nos dados obtidos.

3.2. Implantação e testes

O mecanismo proposto foi implementado em um protótipo totalmente funcional, chamado de ***DefaceSpy***. O protótipo utilizou ferramentas *Open Source* para redução de custos. Foi implementado na linguagem de programação *PHP* e alguns módulos em *Shell Script.* O processo gráfico de criação de arquivos de imagem foi feita pela ferramenta *wkhtmltoimage*[1]. A divisão da imagem gerada, em partes, é feita pela

[1] https://wkhtmltopdf.org

© 2017 Sociedade Brasileira de Computação

ferramenta *ImageMagik*[2]. O protótipo foi implementado no sistema operacional Debian GNU/Linux 8.7.0 64 bits.

Para validar o protótipo foram feitos testes de desfiguração total e parcial em um ambiente controlado. Conforme ilustrado na Figura 2, a imagem da esquerda corresponde à página original, enquanto à imagem da direita corresponde à página capturada para análise. Observa-se uma alteração parcial na página.

Figure 2. Comparação de páginas

A Figura 3 apresenta o texto com o resultado da análise de uma desfiguração parcial da página de teste, demonstrando a eficácia da proposta.

```
################ DefaceSpy Versao:0.0.1 ################
[*] Dimensoes da Pagina Atual x Snapshot: OK
[*] Total de partes obtidas (Pagina Atual): 176
[*] Total de partes obtidas (Snapshot): 176
[*] Resultado da comparacao Snapshot e a versão Atual
>> Limite toleravel de alteracoes na pagina: 90%
>> Total de Partes do snapshot: 176
>> Total de Partes da versão Atual: 176
Similaridade entre o Snapshot e a versão Atual: 54.55%
Resultado: Similaridade abaixo da tolerancia de 90%.

ATENCAO! PAGINA POSSIVELMENTE DESFIGURADA.

[*] Gerado relatorio: defacespy_07052017130705.html
```

Figure 3. Resultado da análise nos testes

[2] http://www.imagemagick.org/script/index.php

© 2017 Sociedade Brasileira de Computação

Além desse teste, também foram realizados testes com alterações da resolução da página e com a desfiguração total. Em todos houve a identificação dos ataques.

3.2. Considerações

O protótipo implementado e os resultados dos testes demonstraram a efetividade do modelo proposto.

Uma questão a ser tratada no futuro diz respeito à definição do indicador de similaridade em páginas dinâmicas. Para cada caso deve-se realizar ajustes, dependendo da periodicidade das atualizações de página e do grau de modificação. O uso de mecanismos de aprendizado de máquina pode ser bastante útil para a automatização do cálculo do indicador.

Uma das limitações dessa proposta é a impossibilidade de identificar alterações não visíveis da página, como a infecção de scripts para ataques aos clientes. Para esses casos seria necessário incluir outras técnicas de detecção que analisassem também o código HTML da página. A associação de outras técnicas deverá ser abordada em trabalhos futuros.

Conclusão

O presente trabalho analisou a literatura contendo os principais métodos de detecção de desfiguração de páginas web e propôs um novo e simplificado mecanismo automatizado de detecção de ataques, baseado na simetria de imagens.

O modelo proposto foi implantado em um protótipo totalmente funcional, que demonstrou a eficácia da proposta.

Pretende-se, em trabalhos futuros, desenvolver técnicas automatizadas para o cálculo dos limites do indicador de similaridade, possivelmente utilizando inteligência artificial.

Referências

Borgolte, K., Kruegel, C.; Vigna, G. Meerkat: “Detecting Website Defacements through Image-based Object Recognition”. In: USENIX Security Symposium. 2015. p. 595-610.

Davanzo, G., Medvet, E., Bartoli, A (2011). “Anomaly detection techniques for a web defacement monitoring servisse”. Expert Systems with Applications, v. 38, n. 10, p. 12521-12530.

Gurjwar, K.R., Sahu, D.R., Tomar, D.S. (2013). “An approach to reveal website defacement”. International Journal of Computer Science and Information Security, v. 11, n. 6, p. 73.

Kanti, T., Richariya, V., Richariya, V. (2011). “Implementing a Web browser with Web defacement detection techniques”. World of Computer Science and Information Technology Journal (WCSIT), v. 1, n. 7, p. 307-310.

© 2017 Sociedade Brasileira de Computação

Oriyano, J.M. (2014). CEH v8: Certified Ethical Hacker Version 8 Study Guide. Wiley & Sons, USA.

Philippe, H., Lauren, W. (2017). “What is the Document Object Model?”, https://www.w3.org/TR/DOM-Level-2- Core/introduction.html.

Verma, R.K.; Sayyad, S. (2015). “Implementation of Web Defacement Detection”. International Journal of Innovations in Engineering and Technology (IJIET), v.6, p. 134-141.

 © 2017 Sociedade Brasileira de Computação

Um Estudo Prático sobre o Potencial do Ataque Slowloris a partir de Dispositivos Móveis

Lucas O. C. Aversari[1], Raoni Kulesza[1], Josilene A. Moreira[1]

[1]Centro de Informática – Universidade Federal da Paraíba (UFPB)
Caixa Postal 5.115 – 58.051-900 – João Pessoa – PB – Brasil

astecz360@gmail.com, raoni@lavid.ufpb.br, josilene@ci.ufpb.br

Abstract. *DDoS attacks have become a real and frequent threat to the sites and services offered globally. This work evaluates the potential of running one of these attacks, the worldwide known Slowloris, from Android mobile devices. Generally run from desktops, Slowloris has been adapted by the authors to run on mobile devices. The attacks were fired from one and two smartphones and with variable intensity (250 to 1000 attackers). Server availability metrics, tts, CPU consumption, and memory were evaluated. It has been found that the harmful potential of the mobile version is as high as the desktop version, consuming very little CPU and memory resources.*

Resumo. *Os ataques DDoS têm se tornado uma ameaça real e frequente aos sites e serviços oferecidos no cenário mundial. Este trabalho avalia o potencial da execução de um destes ataques, o Slowloris, mundialmente conhecido, a partir de dispositivos móveis Android. Geralmente executado a partir de desktops, o Slowloris foi adaptado pelos autores para rodar em dispositivos móveis. Os ataques foram disparados a partir de um e dois smartphones e com intensidade variável (250 a 1000 atacantes). Avaliaram-se as métricas disponibilidade do servidor, tts, consumo de CPU e memória. Verificou-se que o potencial danoso da versão móvel é tão alto quanto da versão desktop, consumindo pouquíssimos recursos de CPU e memória.*

1. Introdução

As diversas facilidades oferecidas pelo uso de redes sociais em computadores e dispositivos móveis fazem com que estas possuam números cada vez maiores de usuários e grupos com interesses semelhantes, representando um cenário propício para as mais diversas atividades maliciosas como a realização de ataques de negação de serviço, ou DDoS (do inglês, *Distributed Denial of Service*), envio de *spam*, violações de privacidade e disseminação de *malware* [1]. Nesse contexto, se encaixa uma tendência emergente conhecida como *hacktivismo*, onde grupos hackers realizam ataques e roubo de informações como forma de protesto político ou social [3]. O objetivo visado por estes protestos é a atenção da mídia, do governo e da comunidade Web.

Segundo a Microsoft [5], ataques DDoS são a maior ameaça para os administradores de serviços Web, sendo extremamente populares devido à sua facilidade de implementação e condução, além da difícil atribuição de autoria, devido à sua natureza distribuída. Isso faz com que pessoas, empresas e até mesmo governos façam uso de tal ameaça como forma de guerra cibernética [6]. Na atualidade a maioria

© 2017 Sociedade Brasileira de Computação

dos ataques DDoS é dirigida à camada de aplicação, com pouca geração de tráfego e baixo consumo de recursos do atacante, também conhecidos como *Slow DDoS*, tendo como o principal representante o Slowloris, implementado por Robert Hansen [7]. Este trabalho visa avaliar o potencial do ataque DDoS Slowloris disparado a partir de dispositivos móveis, a fim de compreender sua capacidade de afetar servidores web.

2. Ataques DDoS e Slowloris

Considerando o cenário de ataques de negação de serviço a partir de dispositivos móveis, podem ser encontradas várias aplicações baseadas na ferramenta desktop LOIC (do inglês, *Low-Orbit Ion Cannon*) [11], desenvolvida pelo grupo de hacktivistas Anonymous, capazes de realizar ataques DDoS do tipo inundação. No entanto, ataques de inundação não se mostram adequados e nem escaláveis ao cenário dos dispositivos móveis, devido à grande quantidade de banda e recursos computacionais por eles consumida. Para este contexto, os ataques slow DDoS são mais adequados.

O Slowloris (HTTP GET) é um ataque slow DDoS realizado através do protocolo HTTP, explorando suas características e vulnerabilidades. O principal foco é o aproveitamento da característica de *timeout* de um servidor Web, que é o tempo máximo em segundos que uma requisição permanece no *buffer* de aplicação até ser atendida. Segundo Cambiaso, Papaleo e Aiello [2] e Karim, Shah e Salleh [14], tal *timeout* pode ser descoberto por meio de um *sniffer* de rede, sendo uma tarefa relativamente simples.

O ataque usando Slowloris é de difícil detecção por parte dos sistemas anti-DDoS, pois consegue gerar tráfego a uma taxa baixa e de pequeno volume. O Slowloris tenta manter muitas conexões com o servidor Web alvo pelo maior tempo possível [15]. O mesmo inicia solicitando várias conexões TCP ao servidor, onde, em cada uma delas, o atacante envia uma requisição GET incompleta. Na etapa seguinte, em algum instante próximo do timeout, um novo cabeçalho incompleto é enviado para manter a conexão ativa. Com a continuidade do ataque, o número de conexões tende a aumentar, consumindo assim todos os recursos da fila de atendimento da aplicação, levando o servidor a um estado de indisponibilidade, não conseguindo, com isso, responder às próximas requisições [7]. Em um cenário de um ataque slow DoS colaborativo utilizando dispositivos móveis ou fixos, todos os dispositivos executando a ferramenta de ataque têm como alvo indisponibilizar um serviço hospedado em um determinado domínio (URL), geralmente um servidor Web. A indisponibilidade tende a se manter enquanto durar o ataque [13].

3. Metodologia

3.1 A Ferramenta de Ataque para Dispositivos Móveis

A ferramenta de ataque adaptada para permitir que os ataques fossem disparados a dispositivos móveis Android é uma versão do utilitário Slowloris que utiliza o mesmo *script* da versão desktop, desenvolvida pelos autores. O código em *perlscript* da versão original foi encapsulado pela biblioteca perlDroid [16] e conta com interface gráfica para simplificar o manuseio. Todos os componentes são empacotados dentro do arquivo de aplicação (.apk), podendo ser instalado e executado em dispositivos Android acima da versão 2.3. A interface projetada para o aplicativo contém apenas campos para inserção da URL a ser atacada e seleção da intensidade do ataque, além de um botão

 © 2017 Sociedade Brasileira de Computação

para iniciar o ataque. O papel da interface é a passagem de parâmetros para o módulo da biblioteca perlDroid, que por sua vez, executa o *perlscript* original da versão *desktop*: "*slowloris.pl*".

São usadas quatro diferentes configurações do ataque nos experimentos: *lite, normal, extreme* ou *uber*, variando entre 250, 500, 750 e 1000 atacantes. Os atacantes são as threads responsáveis pelos envios de requisições HTTP GET para o alvo, com um timeout de 35s definido entre as mesmas. Tal configuração justifica-se pelo fato de o timeout HTTP padrão da maioria dos servidores Web disponíveis no mercado ser de 40s e, segundo a Netcraft, servidores de médio porte são aqueles que possuem capacidade para atender requisições de 750 usuários simultaneamente [17]. As intensidades variáveis permitem que haja a ocupação de 33% (lite), 66% (normal), 100% (extreme) e 133% (uber) do buffer de atendimento do servidor.

3.2 Experimentos e Cenários

Foram realizados experimentos em um cenário composto por um servidor Web, dispositivos móveis executando a ferramenta de ataque Slowloris e um computador executando a ferramenta de simulação de clientes honestos Siege. Utiliza-se o servidor Web Apache 2, configurado, tipicamente, como um servidor de médio porte, capaz de atender 750 clientes simultaneamente [17]). O *timeout* HTTP do servidor foi mantido na sua configuração padrão de instalação de 40 segundos. A escolha do servidor justifica-se através de dados apresentados pela Netcraft, que mostram que aproximadamente 46% dos sites da Internet são hospedados em servidores Apache 2 [17].

Para simulação dos clientes honestos, o software *Siege HTTP load test* foi escolhido, como utilitário de *benchmark* e carga no servidor, devido a sua grande presença e referência na literatura [17]. Em todos os testes, 250 clientes honestos foram simulados pelo Siege como usuários constantes do servidor Web, mantendo uma taxa de ocupação legítima de aproximadamente 33% do *buffer* de atendimento (taxa de ocupação em casos de uso normal para esta categoria de servidores [17]).

As métricas coletadas no servidor foram a disponibilidade média do domínio (taxa de clientes honestos que consegue efetivamente acessar o site) e o tempo de serviço (TTS, do inglês, *time-to-service*) médio, que representa o intervalo até que elas sejam atendidas pelo servidor. Assim, é possível analisar o impacto na qualidade de serviço dos clientes [13]. Adicionalmente, mais duas métricas foram coletadas: consumo de CPU e memória usados pelas ferramentas de ataque e pelo servidor Web Apache. Todos os testes de ataque tiveram duração capturada de 120 minutos. Cada experimento foi realizado trinta e três vezes (33) e então obtida a média de cada métrica, resultando num nível de confiança de 95%.

Os mesmos parâmetros de configuração foram usados em chamadas ao "*slowloris.pl*" na versão *desktop*. O *hardware* utilizado nos testes foi:

- Atacantes: (2x) Xperia J (MSM7227A@1.00 Ghz, 512 Mb RAM, Android 4.1.2);
- Servidor Web: HP PC 5850 (Athlon 64 X2@3.00 Ghz, 4 Gb RAM, Ubuntu 16.10);
- Clientes: MacBook [2009] (Core2Duo@2.20 Ghz, 6 Gb RAM, OSX 10.12 Sierra);.

© 2017 Sociedade Brasileira de Computação

4. Resultados

4.1 Avaliando o Impacto do Ataque sobre o Servidor

4.1.1. Ataques Utilizando Um Dispositivo Móvel

A Tabela 1 apresenta os resultados de disponibilidade e tts utilizando um aparelho celular para os ataques. Para fins de comparação são mostrados os resultados para a versão Slowloris *desktop*, com os mesmos parâmetros de configuração.

Tabela 2: Testes com apenas um dispositivo atacante

***Slowloris* Mobile**			***Slowloris* Desktop**		
Intensidade do Ataque	Disponibilidade (%)	TTS (s)	**Intensidade do Ataque**	Disponibilidade (%)	TTS (s)
Lite	100,00	0,08	**Lite**	100,00	0,06
Normal	98,97	0,29	**Normal**	99,68	0,23
Extreme	0,00	∞	**Extreme**	0,00	∞
Uber	0,00	∞	**Uber**	0,00	∞

Os resultados da versão móvel são semelhantes à versão tradicional. As mínimas variações nos resultados se devem às atividades executadas pelo servidor Web no momento dos ataques. Percebe-se que, tanto para a versão mobile como para a versão desktop, o servidor só se torna indisponível a partir de ataques cuja intensidade definida é "Extreme", já que o número de atacantes é igual ao número máximo de clientes que podem ser atendidos simultaneamente pelo servidor, negando assim, serviço a clientes legítimos.

Outra observação importante é o impacto no tempo de serviço (tts) gerado pelo ataque a partir do momento em que todas as posições do servidor encontram-se ocupadas. Isso ocorre no cenário descrito a partir de ataques "Normal", onde temos 500 atacantes e 250 clientes usufruindo de um buffer de atendimento de 750 posições no servidor. Ocorre um aumento no tts devido ao fato de que novas requisições feitas pelos clientes tem que esperar para serem atendidas, já que todas as posições disponíveis encontram-se em uso.

4.1.2. Ataques Utilizando Dois Dispositivos Móveis

Percebe-se que, neste cenário, a efetividade do ataque é bem maior do que utilizando apenas um dispositivo. Quando dois dispositivos executando o ataque na intensidade "Normal" (500 atacantes) de forma colaborativa atingem o servidor, o mesmo já se torna indisponível, pois só possui capacidade para atender 750 clientes simultaneamente. É possível perceber o potencial danoso do ataque, principalmente quando realizado de forma colaborativa e distribuída.

Tabela 3: Testes utilizando dois dispositivos atacantes

***Slowloris* Mobile**			***Slowloris* Desktop**		
Intensidade do Ataque	Disponibilidade (%)	TTS (s)	**Intensidade do Ataque**	Disponibilidade (%)	TTS (s)
Lite	99,48	0,10	**Lite**	99,98	0,11
Normal	0,00	∞	**Normal**	0,00	∞
Extreme	0,00	∞	**Extreme**	0,00	∞
Uber	0,00	∞	**Uber**	0,00	∞

© 2017 Sociedade Brasileira de Computação

4.2 Consumo de Recursos pela Ferramenta de Ataque

Serão expostos abaixo os dados da utilização de memória e CPU pela ferramenta de ataque móvel quando configurada em cada uma das intensidades de ataque. Percebe-se um uso muito baixo de memória e CPU, mantendo-se abaixo dos 10 Mb e 10%, respectivamente, em todos os casos. Neste caso, são mostrados os resultados apenas da versão para dispositivo móvel, que possui mais restrições nos recursos de CPU e memória. Entretanto, avalia-se este consumo nas diferentes configurações de intensidade dos ataques.

4.2.1 Consumo de Memória

Percebe-se que o uso de memória fica em torno dos 7,75 Mb para os cenários Lite, Normal e Extreme. A baixa utilização de memória se deve ao fato do Android tentar otimizar ao máximo a área de *heap* das *threads* iguais, identificando áreas compartilhadas por elas. Já na versão Uber do ataque, o uso de memória permaneceu próximo aos 10 Mb, acima dos demais, devido à alocação de mais memória, necessária para as 1000 *threads* de ataque. O baixo uso de memória tanto na versão desktop, como na versão móvel, se deve à baixa complexidade das threads atacantes criadas, as quais realizam a simples atividade de enviar um HTTP GET periodicamente (Figura 1).

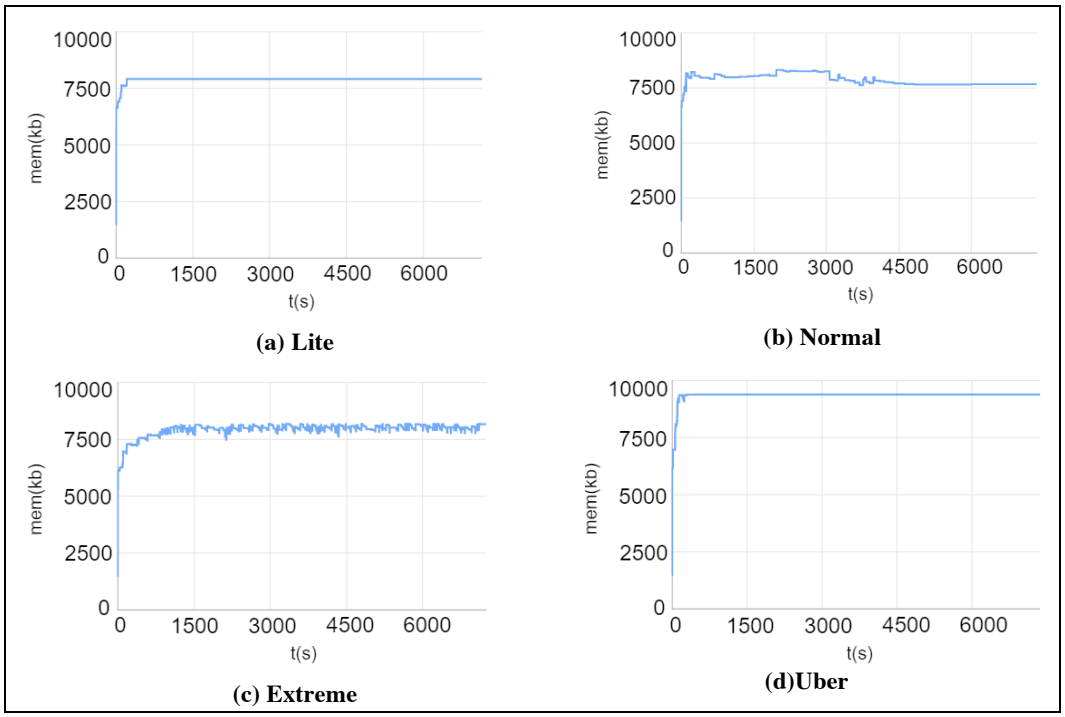


Figura 1: Utilização de memória nas diversas intensidades dos ataques

4.2.2 Consumo de CPU

Verifica-se que uso de CPU tem um pico inicial em torno de 3% no cenário Lite e depois mantém-se abaixo de 2%. Nos cenários Normal e Extreme o pico inicial atinge cerca de 5% e depois cai para menos de 2% também. Este aumento no pico de CPU no

© 2017 Sociedade Brasileira de Computação

momento da inicialização ocorre devido à maior quantidade de *threads* criadas. Apenas no cenário Uber é que o percentual mantém-se acima de 6%. O maior uso de CPU ocorre devido à sobrecarga causada pela virtualização de um número relativamente grande de *threads* (Figura 2).

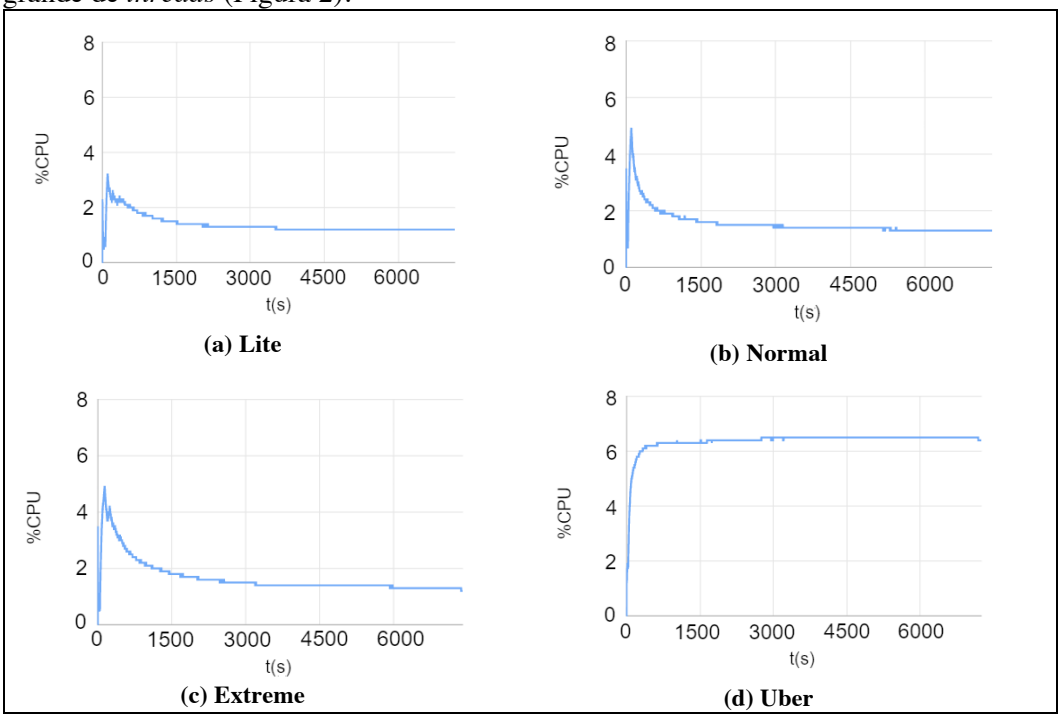


Figura 2: Consumo de CPU nas diversas intensidades dos ataques

5. Conclusões

Sabendo que os dispositivos móveis representam uma grande fatia da massa de dispositivos que acessam a Internet na atualidade, o presente artigo teve como objetivo avaliar o potencial do ataque DDoS Slowloris a partir de dispositivos móveis. Ficou demonstrado que, mesmo no ambiente limitado de um hospedeiro móvel e sem dispor de todas as funcionalidades de um kernel Linux padrão, a execução do *script* original de ataque *slow* DDoS por um dispositivo Android é viável em termos de desempenho e uso de recursos, podendo causar grandes prejuízos a serviços Web.

Com apenas uma instância em execução da ferramenta proposta, é possível indisponibilizar um serviço Web considerado de médio porte. Tal capacidade de realização e portabilidade de ataques permite que uma única pessoa possa utilizar dos seus dispositivos Android para ações maliciosas, prejudicando centenas de outros usuários que desejam acessar legitimamente os serviços disponíveis no alvo. Assim, usuários podem fazer uso de seus dispositivos móveis para colaborar com ataques DDoS difundidos por grupos de ativistas em redes sociais, representando um potencial cenário de ataques colaborativos, os quais precisam ser conhecidos e evitados.

Podemos identificar como trabalho futuro aumentar a escala das avaliações para números mais próximos de um ataque distribuído e/ou colaborativo real, utilizando redes sociais e massas de usuários.

© 2017 Sociedade Brasileira de Computação

Referências

[1] MAKRIDAKIS, A.; ATHANASOPOULOS, E.; ANTONIADES, D.; IOANNIDIS, S.; MARKATOS, E.. Understanding the Behavior of Malicious Applications in Social Networks. IEEE Networks, 2010.

[2] CAMBIASO, E.; PAPALEO, G.; CHIOLA, M.; AIELLO, M. Mobile executions of Slow DoS Attacks. Logic Jnl IGPL, V. 24, No. 1, p. 54–58, 16/10/2015.

[3] R. GANDHI; A. SHARMA; W. MAHONEY; W. SOUSAN; Q. ZHU; P. LAPLANTE. Dimensions of cyber-attacks: Cultural, social, economic, and political," Technology and Society Magazine, IEEE, vol. 30, no. 1, pp. 28–38, 2011.

[4] S. KUMAR; K. M. CARLEY. Approaches to Understanding the Motivations Behind Cyber Attacks, em Intelligence and Security Informatics. (ISI), 2016 IEEE International Conference em, Tucson, Arizona USA, Set. 2016.

[5] MICROSOFT. Microsoft Library: Security Issues with IP. Disponível em: https://technet.microsoft.com/en-us/library/cc959354.aspx. Acesso em 20/11/2016.

[6] S. KUMAR; K. CARLEY.. Understanding DDoS Cyber-Attacks using Social Media Analytics. IEEE Networks, 2016.

[7] DANTAS, Y, G. Estratégias para tratamento de ataques de negação de serviço na camada de aplicação em redes IP. 2015. 77f. Trabalho de Conclusão de Curso (Especialização) Curso de Ciência da Computação, Universidade Federal da Paraíba, João Pessoa/PB, 2015.

[8] J. LEWIS; S. BAKER. The economic impact of cybercrime and cyber espionage, Center for Strategic and International Studies, Washington,DC, pp. 103–117, 2013.

[9] G. MEZZOUR, Assessing the Global Cyber and Biological Threat, dissertação Ph.D., Symantec Research Labs, 2015.

[10] B. LIU; L. ZHANG. A survey of opinion mining and sentiment analysis, em Mining text data. Springer, 2012, pp. 415–463.

[11] LOIC – Low Orbit Ion Cannon. Disponível em: https://play.google.com/store/apps/details?id=genius.mohammad.loic. Acesso em 12/05/2017.

[12] OP TANGO DOWN, Disponível em: http://www.anonymousbrasil.com/tango-down/. Acesso em 10/06/2017.

[13] DANTAS, Y, G.; NIGAM, V.; FONSECA, I. A Selective Defense for Application Layer DDoS Attacks, Intelligence and Security Informatics Conference (JISIC), Hague, 8/12/2014.

[14] A. KARIM; S. A. A. SHAH; R. B. SALLEH; M. ARIF; R. M. NOOR; S. SHAMSHIRBAND. Mobile Botnet Attacks – an Emerging Threat: Classification, Review and Open Issues. Trans. on Internet and Information Systems, v. 9, 2015.

[15] SLOWLORIS DDoS Attack. Disponível em: https://security.radware.com/ddos-knowledge-center/ddospedia/slowloris/. Acesso em 20/11/2016.

[16] PERLDROID. Disponível em: https://code.google.com/archive/p/perl-android-apk/. Acesso em 20/11/2016.

[17] NETCRAFT. NetCraft WebServer Survey. Disponível em: https://news.netcraft.com/archives/2016/09/19/september-2016-Web-server-survey.html. Acesso em 20/11/2016.

© 2017 Sociedade Brasileira de Computação

Um Método para Detecção de *Bots* Sociais Baseado em Redes Neurais Convolucionais Aplicadas em Mensagens Textuais

Paulo A. Braz[1], **Ronaldo R. Goldschmidt**[1]

[1]Seção de Engenharia da Computação (SE/8) – Instituto Militar de Engenharia (IME)
22.290-270 – Rio de Janeiro – RJ – Brazil

{paulo.braz,ronaldo.rgold}@ime.eb.br

Abstract. *Currently, social networks are subject to social bots that perform malicious activities such as the dissemination of fake news. Some research to detect this type of malware is based on statistics extracted from the content of posted messages. Once the extraction of statistics may lead to information loss, this work aims to present experimental evidences that the use of original textual messages can improve detection accuracy. To this end, we propose a method that applies a convolutional neural network to identify suspicious messages. Good preliminary results with Twitter data indicate adequacy of the proposed method.*

Resumo. *Atualmente, as redes sociais estão sujeitas a ações de bots sociais que executam atividades maliciosas como a disseminação de notícias falsas. Algumas pesquisas voltadas à detecção desse tipo de malware se baseiam em estatísticas extraídas a partir do conteúdo das mensagens postadas. Como a extração de estatísticas pode ocasionar perda de informação, este trabalho tem como objetivo apresentar evidências experimentais de que o uso de textos originais das mensagens pode melhorar a precisão de detecção. Para tanto, propõe-se um método que aplica uma rede neural convolucional para identificar mensagens suspeitas. Resultados preliminares utilizando dados do Twitter se mostraram promissores, fornecendo indícios de adequação do método proposto.*

1. Introdução

As redes sociais têm figurado como um espaço cada vez mais relevante no expediente de obtenção e disseminação de informações via Internet. De acordo com [Badri Satya et al. 2016], bilhões de usuários utilizam diariamente as mídias sociais, compartilhando informações sobre suas vidas e expressando opiniões através de mensagens, *likes* e avaliações. Por exemplo, dos cerca de 3,7 bilhões de usuários atualmente conectados à Internet, 1,5 bilhão utiliza o Facebook e aproximadamente 360 milhões possuem conta no Twitter[1].

As redes sociais encontram-se suscetíveis a ações dos *bots* sociais, contas automatizadas capazes de interagir com humanos e produzir conteúdo automaticamente. Contas deste tipo podem ser utilizadas para manipulação de opiniões de usuários das redes sociais por meio de atividades maliciosas como disseminação de boatos e notícias falsas, adulteração ou mesmo concepção de estatísticas de percepção pública, dentre outras [Ratkiewicz et al. 2011].

[1]**Statista** - www.statista.com/statistics/

 © 2017 Sociedade Brasileira de Computação

A detecção de *bots* sociais é a tarefa de identificar contas automatizadas e diante deste contexto, faz-se necessário detectar e mitigar os efeitos nocivos destas entidades. Neste cenário, diversos trabalhos de pesquisa em detecção de *bots* sociais seguem a abordagem baseada em atributos ou características das contas (do inglês, *feature-based approach*) [Ferrara et al. 2016]. Os atributos a que se refere o nome desta abordagem são informações que descrevem o comportamento das contas das redes sociais. Número de amigos ou seguidores de uma conta, quantidade de contas que cada conta segue, número de mensagens propagadas em uma janela de tempo, quantidade de caracteres na descrição do perfil da conta, quantidade de fotos postadas, dentre outros, são exemplos de atributos comportamentais utilizados na detecção de *bots* sociais [Lee et al. 2011], [Davis et al. 2016] e [Wang et al. 2016]. Diante disso, é válido salientar que a maioria desses trabalhos aplica técnicas de aprendizado de máquina para, a partir dos dados sobre as contas de uma rede social, buscar construir modelos de classificação binária que sejam capazes de identificar com a maior precisão possível quais delas são *bots* sociais [Lee et al. 2011], [Davis et al. 2016].

Além de informações comportamentais sobre as contas das redes sociais, alguns dos trabalhos da abordagem de detecção de *bots* sociais baseada em atributos também consideram dados estatísticos sobre os textos das mensagens propagadas pelas contas, tais como: quantidade de links nas mensagens, número de menções a terceiros via "@nomeusuário", etc [Yang et al. 2014] e [Lee et al. 2011]. Nesses trabalhos, os textos brutos[2] das mensagens são submetidos a uma etapa de engenharia de atributos que é responsável por extrair dados estatísticos de interesse. No entanto, como qualquer consolidação de dados pode ocasionar perda de informação [Beaudry and Renner 2012], o presente trabalho levanta como hipótese que, ao utilizarem estatísticas para representar os textos das mensagens, os trabalhos relacionados possam ter eliminado indícios úteis para uma detecção de *bots* sociais mais precisa.

Assim sendo, o objetivo deste artigo é apresentar um conjunto de evidências experimentais de que a utilização de textos brutos de mensagens postadas por contas de redes sociais pode contribuir para produzir modelos de detecção de *bots* sociais mais precisos do que aqueles que se baseiam exclusivamente nos atributos comportamentais dessas contas. Para tanto, o artigo propõe um método de detecção de *bots* sociais que utiliza uma rede neural convolucional[3] [LeCun et al. 2004] adaptada para analisar os textos brutos das mensagens a fim de identificar quais são suspeitas de terem sido produzidas por *bots*. Em seguida, aplica-se um modelo que busca classificar cada conta como *bot* ou não *bot* com base nos atributos comportamentais e no percentual de mensagens da conta em questão que tenham sido consideradas suspeitas pela etapa anterior. Os resultados obtidos em um experimento preliminar envolvendo dois subconjuntos de contas e mensagens do Twitter se mostraram promissores, fornecendo indícios de adequação do método proposto.

O restante do artigo segue organizado da seguinte forma: a seção 2 apresenta a formalização do método proposto, descrevendo cada uma das etapas envolvidas; a seção 3 exibe um detalhamento acerca do experimento realizado e dos resultados preliminares

[2]Neste artigo, a expressão "texto bruto"será utilizada para referenciar o conteúdo original de uma mensagem que não tenha sido submetido a nenhum tipo de pré-processamento ou de engenharia de atributos.

[3]Este tipo de rede pertence a uma classe de algoritmos de aprendizado de máquina capazes de identificar características complexas a partir de características mais simples [Bezerra 2016]

© 2017 Sociedade Brasileira de Computação

obtidos; e, por fim, a seção 4 expõe as considerações finais do trabalho e aponta para possíveis alternativas de trabalhos futuros.

2. Método Proposto

Seja *N* uma rede social representada por um grafo dirigido $G(V, E)$, onde cada vértice $v \in V$ representa uma conta (ou usuário) de N e cada aresta $e \in E$ corresponde a um par ordenado entre duas contas u e v, representado por $e = (u, v)$, que indica que u segue v. Suponha que cada conta u possui um conjunto de mensagens $M_u = \{m_{u,1}, m_{u,2}, ..., m_{u,u_k}\}$ postadas por ela em N. O conjunto de todas as mensagens de N é representado por $M_N = \bigcup_{i=1}^{|V|} M_{u_i}$, onde $|V|$ corresponde à cardinalidade de V. Considere ainda que cada conta u possui uma lista ordenada de atributos $(u.a_1, u.a_2, ..., u.a_r, u.c)$, sendo $u.a_i$ comportamentais e $u.c$ um atributo binário que informa se u é *bot* ou não. O método proposto possui quatro etapas conforme ilustrado na Figura 1. A descrição de cada uma delas encontra-se detalhada a seguir.

- Etapa 1 - Seleção de Mensagens e Contas - Processo responsável por construir três conjuntos que serão utilizados pelas demais etapas: V', M'_{Treino} e M'_{Teste}. Inicialmente, considera-se que $V' = \emptyset$, $M'_{Treino} = \emptyset$ e $M'_{Teste} = \emptyset$. A construção dos conjuntos é realizada da seguinte forma: executa-se uma seleção aleatória sem reposição de x_{treino} mensagens de M_N. Tais mensagens são armazenadas em M'_{Treino}. Para cada mensagem $m \in M'_{Treino}$, recupera-se a conta u tal que $m \in M_u$ e atualiza-se $V' = V' \cup \{u\}$. Após a formação de V' e M'_{Treino}, inicia-se a construção de M'_{Teste}. Para cada $u \in V'$, são selecionadas aleatoriamente x_{teste} mensagens $m \in M_N$ tais que $m \notin M'_{Treino}$. Tais mensagens são incluídas em M'_{Teste}. Desta forma, ao final do procedimento, tem-se que $M'_{Treino} \cap M'_{Teste} = \emptyset$. Cabe ressaltar que tanto x_{treino} quanto x_{teste} são variáveis definidas pelo usuário do processo. Em seguida, toda mensagem $m \in M'_{Treino} \cup M'_{Teste}$ é enriquecida com a classificação da conta u responsável por sua propagação, i. e., $u.c$. Tal informação será utilizada na etapa 2.

- Etapa 2 - Classificação de Mensagens - Esta etapa subdivide-se em dois passos. No primeiro, treina-se uma rede convolucional com o conjunto M'_{Treino}. Para tanto, a rede recebe como entrada o texto bruto de cada mensagem m, sendo a classificação da mensagem a saída desejada $u.c$ a ser aprendida pela rede. Uma vez concluído o treinamento da rede neural, o segundo passo consiste em aplicar o modelo aprendido pela rede a cada uma das mensagens contidas em M'_{Teste}, contabilizando, ao final, para cada conta u, o percentual de mensagens classificadas como sendo suspeitas de terem sido propagadas por *bots* sociais.

- Etapa 3 - Enriquecimento de V' - Esta etapa é responsável por acrecentar uma nova informação em cada uma das contas $u \in V'$. Assim, a lista ordenada de atributos que descreve u passa a ser $(u.a_1, u.a_2, ..., u.a_r, u.c, u.susp)$, onde $u.susp$ contém o percentual de mensagens de u classificadas como suspeitas na etapa anterior.

- Etapa 4 - Classificação de Contas - Dado um algoritmo de classificação S, esta etapa realiza um processo de validação cruzada com k conjuntos sobre V'. Este

 © 2017 Sociedade Brasileira de Computação

processo consiste em dividir V' em k conjuntos de contas para, em seguida, realizar k iterações. Em cada iteração, um dos k conjuntos é utilizado como conjunto de teste e os $k-1$ restantes utilizados para treinamento de S. O desempenho do modelo de classificação construído a cada iteração é armazenado. O processo se repete até que todos os k conjuntos tenham sido utilizados uma vez como conjunto de teste. Ao final do processo, é obtido o desempenho médio dos k modelos gerados. Em todas as iterações do processo, os atributos $a_1, a_2, ..., a_r, susp$ foram fornecidos como entradas e o atributo c como saída de S.

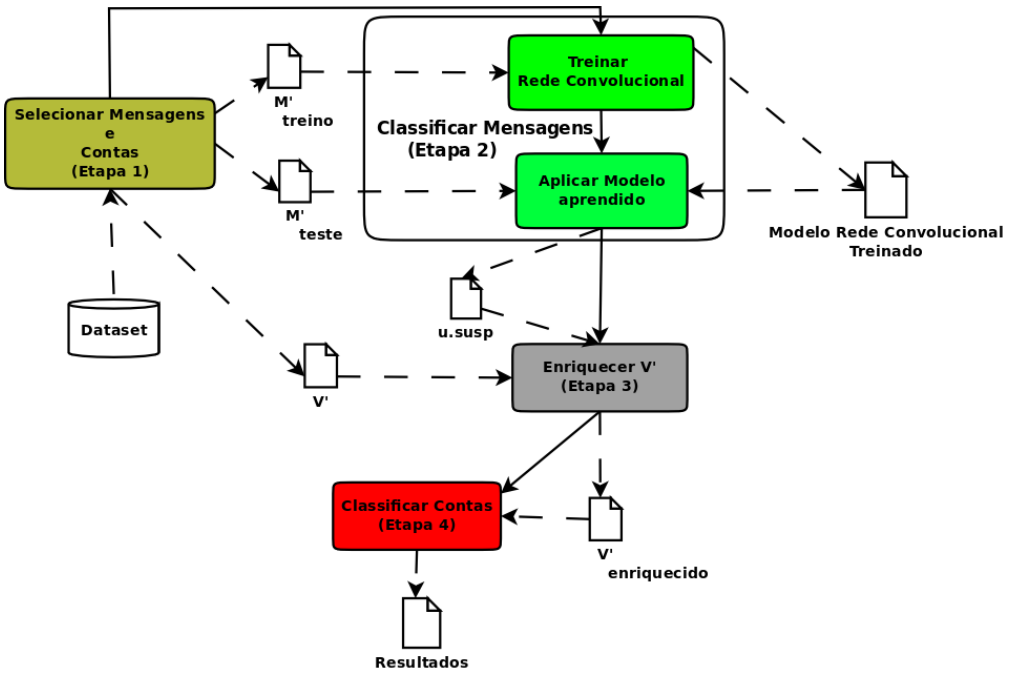


Figura 1. Visão Geral das Etapas do Método Proposto.

3. Experimento e Resultados Preliminares

A fim de avaliar o método proposto, foi realizado um experimento inicial com o mesmo *dataset* adotado em [Lee et al. 2011]. Trata-se de uma base de dados do *Twitter* com informações referentes ao período de dezembro de 2009 a agosto de 2010[4]. A base contém os atributos comportamentais indicados na Tabela 1, além dos textos de todas as mensagens publicadas no período. No total, o *dataset* contém aproximadamente 40 mil contas e 5,5 milhões postagens, sendo 87 postagens por conta, em média, aproximadamente. A escolha deste *dataset* deveu-se basicamente à disponibilidade dos atributos comportamentais em conjunto com os textos publicados pelas respectivas contas.

Diante do elevado volume de contas e mensagens disponíveis no *dataset* completo, o que demanda grande poder computacional e elevado tempo para o treinamento dos modelos de aprendizado de máquina, optou-se por utilizar subconjuntos menores na realização do treinamento do modelo para a geração dos resultados preliminares. Desta

[4]Neste *dataset*, cada conta contém um *label* que indica se a conta é falsa (*bot*) ou legítima.

© 2017 Sociedade Brasileira de Computação

Tabela 1. Atributos comportamentais disponíveis no *dataset* do experimento.

Atributos	Descrição
# usuários seguindo	número de usuários que a conta segue
#tweets	número de tweets postados pela conta
razão de #seguindo por #seguidores	razão de #usuários seguidos por #seguidores
#usuários seguidores	número de seguidores

forma, foram avaliados dois cenários: no primeiro foram utilizadas 2000 mensagens publicadas e no segundo 6000 mensagens para o treinamento e teste do classificador de mensagens. A seleção de mensagens e contas de ambos os cenários seguiu o procedimento descrito na etapa 1 do método proposto. A Tabela 2 apresenta um breve detalhamento estatístico sobre os dois cenários do experimento[5].

Tabela 2. Dados estatísticos sobre os cenários do experimento

	x_{treino}	x_{teste}	$x_{treino} + x_{teste}$	$\|V'\|$	Média Msg/Conta	Contas *bot*
Cenário 1	1000	1000	2000	53	37.7	50%
Cenário 2	3000	3000	6000	160	37.5	50%

A etapa de classificação de mensagens foi implementada em ambos os cenários com uma rede neural convolucional cuja configuração e arquitetura estão descritas na Tabela 3. A implementação deste modelo foi feita utilizando *Python* e *Keras* [Chollet et al. 2015].

Tabela 3. Arquitetura da rede convolucional

	Filtros	Kernel	Stride
2 Camadas	150	7X7	1
4 Camadas	150	3X3	1
Fully Connected	31 neurônios	–	–
Última camada	2 neurônios	–	–

As mensagens precisaram ser formatadas a fim de serem submetidas à rede na etapa 2. Para esta formatação foi utilizada a técnica *Char Quantization* proposta por [Zhang et al. 2015][6]. Desta forma, cada texto de mensagem publicada foi transformado em uma matriz binária de dimensão 150x64, uma vez que 150 é o limite máximo de caracteres para cada postagem no Twitter e 64 é a quantidade de caracteres considerados para mapeamento. A Figura 2 apresenta o conjunto de caracteres considerados para fins de mapeamento. Assim, as matrizes geradas foram esparsas. Cada linha de uma dada matriz correspondeu a um e, somente um, caracter da mensagem associada e, portanto, teve o *bit* setado para 1 exatamente na coluna relativa ao caracter identificado e 0 para as demais colunas da linha em questão.

[5]É válido ressaltar que foi adotada, em ambos os cenários, a mesma distribuição de classes encontradas no *dataset* original, em relação ao número de contas: 50% de contas do tipo *bot* e 50% de contas legítimas.

[6]A escolha por este tipo de formatação deveu-se basicamente pelo bom desempenho proporcionado por ela nos experimentos de classificação de texto relatados pelos autores.

 © 2017 Sociedade Brasileira de Computação

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
 'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '-', ',', ';', '.', '!', '?', ':', '/',
 '\',','_', '@', '#', '$', '%', '^', '&', '*', '+', '-', '=', '<', '>', '(', ')', '[', ']', '{', '}']
```

Figura 2. Conjunto de caracteres usado para mapeamento das mensagens.

Ainda na etapa de Classificação de Mensagens foi executado o procedimento de *holdout* (com 2/3 de M'_{Treino} para treino e 1/3 de M'_{Treino} para validação), para cada um dos cenários. Deste modo, com o classificador treinado utilizando M'_{Treino}, fez-se a apresentação de M'_{Teste} para o modelo efetuar a classificação de cada mensagem m e, por conseguinte, viabilizar o cálculo do percentual de mensagens classificadas como sendo geradas por *bots* para todas as contas da amostra $(u.susp)$. Deste modo, com as taxas geradas, pode-se enriquecer V', finalizando assim a execução da Etapa 3.

Para classificação das contas foram utilizados dois algoritmos de classificação: *Random Forest* [Ho 1995] e MLP (*Multilayer Perceptron*) [Rosenblatt 1961]. Em cada cenário, cada algoritmo foi submetido a um processo de validação cruzada com 10 conjuntos. Os algoritmos executados foram instanciados com suas configurações *default*, utilizando Weka [Witten et al. 2016]. Importante notar que foi adotada a mesma parametrização dos algoritmos durante todo o processo. A Tabela 4 apresenta os parâmetros adotados por cada algoritmo.

Tabela 4. Configuração utilizada para cada algoritmo de classificação

Algoritmo	Configuração
MLP	hidden layers= 4, learning rate=0.3, momentum=0.2, epochs=500
Random Forest	bag size percent = 100, batch size = 100, num iterations = 100

A fim de comparar a influência dos textos brutos no processo de detecção de *bots* sociais, os mesmos algoritmos de classificação utilizados na etapa 4 foram aplicados sobre o conjunto V' sem o enriquecimento realizado pelo método proposto. Também neste caso, foi realizado um processo de validação cruzada com 10 conjuntos.

A Tabela 5 mostra as acurácias obtidas pelos dois algoritmos na classificação de contas em duas situações: uma com V' contendo apenas os atributos comportamentais da Tabela 1 e a outra com V' enriquecido pelo método proposto para conter também o percentual de mensagens suspeitas associado a cada conta. Pode-se perceber que, de uma forma geral, os algoritmos de classificação aplicados em V' enriquecido apresentaram desempenho superior aos mesmos algoritmos aplicados em V' sem enriquecimento, em ambos os cenários. Os resultados obtidos são, portanto, evidências experimentais de que a utilização de textos brutos de mensagens de contas de redes sociais pode contribuir para produzir modelos de detecção de *bots* mais precisos do que aqueles que se baseiam somente nos atributos comportamentais dessas contas. Ademais, aplicando os algoritmos de seleção de atributos RELIEF [Kononenko 1994] e CorrelationAttributeEval [Witten et al. 2016] (utiliza o coeficiente de Correlação de Pearson), verificou-se que o atributo criado pelo método proposto ficou em primeiro lugar no que se refere ao ganho de informação para a tarefa de detecção de *bots* em ambos os cenários. Este fato reforça a importância de se investigar alternativas de utilização de textos brutos das mensagens na construção de modelos de detecção de *bots* sociais.

© 2017 Sociedade Brasileira de Computação

Tabela 5. Resultados do experimento: acurácia dos classificadores

Cenário	Enriquecimento de V'	MLP			Random Forest		
		Acc.	FPs	FNs	Acc.	FPs	FNs
1	Não	69.23%	11%	19%	**86.5**%	7%	5%
1	Sim	84.61%	7%	7%	**90.3**%	5%	3%
2	Não	80.5%	13%	6%	**89.3**%	4%	6%
2	Sim	81.76%	8%	9%	**90.56**%	3%	5%

4. Considerações Finais

Atualmente as redes sociais encontram-se suscetíveis a ações dos *bots* sociais, contas automatizadas capazes de interagir com humanos e produzir conteúdo automaticamente. Contas deste tipo podem ser utilizadas para manipulação de opiniões de usuários das redes sociais por meio de atividades maliciosas como disseminação de notícias falsas e adulteração de estatísticas de percepção pública, dentre outras.

Algumas pesquisas voltadas à detecção de *bots* sociais se baseiam em estatísticas extraídas a partir do conteúdo das mensagens postadas. Como a extração de estatísticas pode ocasionar perda de informação, este trabalho teve como objetivo apresentar evidências experimentais de que o uso de textos originais das mensagens pode melhorar a precisão de detecção. Para tanto, propôs-se um método que aplica uma rede neural convolucional para identificar mensagens suspeitas. Baseado em atributos comportamentais e no percentual de mensagens suspeitas identificadas em cada conta, o método proposto aplica um classificador binário para detectar a presença de *bots* sociais. Resultados preliminares com dados do Twitter se mostraram promissores, fornecendo indícios de adequação do método proposto, assim como evidências experimentais de que a utilização de textos brutos de mensagens publicadas por contas de redes sociais pode contribuir para produzir modelos de detecção de *bots* sociais mais precisos do que aqueles que se baseiam apenas nos atributos comportamentais dessas contas.

Como trabalhos futuros, destacam-se a utilização de todo conteúdo textual e de todas as contas contidas no *dataset* completo adotado no experimento deste artigo, o emprego de testes estatísticos para avaliar a existência de diferença de desempenho significativa entre os classificadores e a utilização de outros *datasets* e algoritmos de classificação para avaliar a robustez do método proposto. Ainda seria oportuno avaliar a semântica de todas as mensagens propagadas. Ademais, seria interessante ainda avaliar a adequação de outras técnicas de aprendizado profundo para a classificação das mensagens, tais como redes neurais recorrentes do tipo LSTM [Hochreiter and Schmidhuber 1997], por exemplo.

Referências

Badri Satya, P. R., Lee, K., Lee, D., Tran, T., and Zhang, J. J. (2016). Uncovering fake likers in online social networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 2365–2370, New York, NY, USA. ACM.

Beaudry, N. J. and Renner, R. (2012). An intuitive proof of the data processing inequality. *Quantum Info. Comput.*, 12(5-6):432–441.

 © 2017 Sociedade Brasileira de Computação

Bezerra, E. (2016). Introdução à aprendizagem profunda. `http://sbbd2016.fpc.ufba.br/sbbd2016/minicursos/minicurso3.pdf`.

Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

Davis, C. A., Varol, O., Ferrara, E., Flammini, A., and Menczer, F. (2016). Botornot: A system to evaluate social bots. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 273–274. International World Wide Web Conferences Steering Committee.

Ferrara, E., Varol, O., Davis, C., Menczer, F., and Flammini, A. (2016). The rise of social bots. *Commun. ACM*, 59(7):96–104.

Ho, T. K. (1995). Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Kononenko, I. (1994). Estimating attributes: analysis and extensions of relief. In *European conference on machine learning*, pages 171–182. Springer.

LeCun, Y., Huang, F. J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–104. IEEE.

Lee, K., Eoff, B. D., and Caverlee, J. (2011). Seven months with the devils: A long-term study of content polluters on twitter.

Ratkiewicz, J., Conover, M., Meiss, M. R., Gonçalves, B., Flammini, A., and Menczer, F. (2011). Detecting and tracking political abuse in social media.

Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY.

Wang, G., Zhang, X., Tang, S., Zheng, H., and Zhao, B. Y. (2016). Unsupervised clickstream clustering for user behavior analysis. In *SIGCHI Conference on Human Factors in Computing Systems*.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Yang, Z., Wilson, C., Wang, X., Gao, T., Zhao, B. Y., and Dai, Y. (2014). Uncovering social network sybils in the wild. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):2.

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.

© 2017 Sociedade Brasileira de Computação

A era dos Crypto Ransomwares: um estudo de caso sobre o WannaCry

Guilherme Baesso Moreira[1], Vanusa Menditi Calegario, M.Sc.[2], Julio Cesar Duarte, D.Sc.[1], Anderson F. Pereira dos Santos, D.Sc.[1]

[1]Instituto Militar de Engenharia – Rio de Janeiro – RJ – Brasil

[2]Universidade Federal do Rio de Janeiro - Rio de Janeiro – RJ – Brasil

gbaesso@gmail.com, vmenditi@yahoo.com.br, {duarte,anderson}@ime.eb.br

Abstract. *In a context of growing dependency on computational systems and the Internet, the institutions have been investing more and more resources to ensure the Information Security. Despite the countless efforts, incidents are still growing in frequency and severity, showing that more efficient incident response processes are needed. One of the threats in clear growth is the Crypto Ransomwares. This paper develops a case study of WannaCry – a peculiar ransomware that spread panic in early 2017 – with the goal of identifying opportunities for improvement in the incident response systems and processes.*

Resumo. *Num contexto de crescente dependência dos sistemas computacionais e da Internet, as instituições têm investido cada vez mais recursos para garantir a Segurança da Informação. Apesar dos inúmeros esforços, os incidentes continuam crescendo em frequência [CERT.br 2017] e gravidade, evidenciando a necessidade de processos mais eficientes de resposta a incidentes. Uma das ameaças em franca expansão são os Crypto Ransomwares. Este trabalho realiza um estudo de caso do WannaCry – um ransomware bastante peculiar que causou pânico no início de 2017 – com o objetivo de identificar oportunidades de melhoria nos sistemas e processos de resposta a incidentes.*

1. Introdução e Motivação

Segundo relatório da PwC Brasil, o investimento em Segurança da Informação tem crescido a um ritmo anual de 30% a 40% no país, atingindo cifras de até US$ 8 bilhões. Ao mesmo tempo, ataques cibernéticos causaram perda estimada de até R$ 20 bilhões ao ano na economia brasileira [O Globo 2015].

Embora as companhias tenham aumentado o seu investimento em Segurança da Informação, o número de incidentes continua crescendo [CERT.br 2017]. Inúmeras notícias sobre ataques cibernéticos pelo mundo reforçam a percepção de que o problema continua aumentando em frequência e gravidade. Dentro deste contexto, observa-se que a maioria dos ataques cibernéticos atuais possuem motivações políticas ou financeiras [Kharraz et al. 2015] [Reuters 2017] e muitos são financiados por Estados, como parte da ofensiva de uma verdadeira guerra cibernética [Healey 2016].

Ransomware é um tipo de *malware* que impede ou limita o usuário de acessar seu sistema, até que um "resgate" seja pago. Famílias mais modernas do *malware*, conhecidas como *crypto ransomware*, criptografam certos tipos de arquivos em sistemas

 © 2017 Sociedade Brasileira de Computação

infectados e forçam o usuário a pagar um "resgate" para obter a chave de decriptografia [Trend Micro 2016]. Não por acaso, os ataques com *ransomware* têm crescido fortemente [Economist.com 2017] [AYRAPETOV 2017] e estão se tornando mais sofisticados, passando a ter como alvo não apenas usuários finais, mas também sistemas industriais completos [Formby et al. 2017] [Cobb 2017] [Franceschi-Bicchierai 2016].

Diversos incidentes relacionados à infecção por *ransomware* vieram a público nos últimos anos [Gibbs 2016] [Winton 2016] [Smith 2016], porém o mais emblemático talvez tenha sido o WannaCry, devido a algumas características peculiares: **(i) Gerou grande impacto e pânico internacional** – infectou mais de 200 mil computadores em 150 países, interrompeu a operação de hospitais e empresas, e é visto como um dos maiores incidentes da atualidade [Reuters 2017]; **(ii) Kill Switch** -– esta não é uma característica propriamente nova em *malware*, mas no caso do WannaCry foi fundamental na contenção do ataque inicial, tendo evitado a propagação da epidemia e transformado o jovem pesquisador que o identificou numa celebridade; **(iii) Utilizou um código roubado da NSA** e trouxe polêmica sobre o tema de agências de defesa armazenarem código malicioso de vulnerabilidades não divulgadas; **(iv) Levou a Microsoft a desenvolver, extraordinariamente, atualizações para o Windows XP**, sistema operacional cujo suporte foi descontinuado em 8 de abril de 2014; **(v) Apesar do impacto, o *malware* não possui qualidades inovadoras**, pelo contrário: possui baixa complexidade e reaproveitou código de ameaças anteriores.

Estas características tornam o WannaCry um ótimo candidato para um estudo de caso. Além disso, o alcance e velocidade do ataque mostram a necessidade de uma melhor preparação para se responder rapidamente a futuras ameaças.

2. Objetivos

O objetivo é realizar um estudo de caso do WannaCry e, no contexto de um trabalho maior de dissertação de Mestrado, contribuir com a identificação de oportunidades de melhoria nos sistemas e processos preventivos e responsivos, subsidiando a construção de um *framework* de apoio para o desenvolvimento e operação de iniciativas de resposta a incidentes de Segurança da Informação.

3. Metodologia

Foi desenvolvida uma pesquisa de caso exploratória em uma empresa listada na Fortune 500 (lista com as 500 maiores empresas do mundo, segundo a revista Fortune), realizando um estudo analítico de combinação de padrão [Yin 2015] que compara as observações empíricas deste estudo com descobertas de trabalhos relacionados.

Os dados da pesquisa foram obtidos de documentações internas sobre o incidente, entrevistas informais com atores chave, bem como de informações publicamente disponíveis. Também foi realizada revisão bibliográfica de trabalhos relacionados que auxiliam na convergência das teorias defendidas.

4. Estudo de caso

Esta seção descreve detalhes sobre a ameaça e as observações do estudo de caso. Como se trata de um estudo de caso único e não foram encontrados outros estudos disponíveis com o mesmo nível de detalhes, as comparações se limitam a dados e estatísticas obtidos em fontes de dados abertas.

© 2017 Sociedade Brasileira de Computação

4.1. Visão geral do ransomware WannaCry

A ameaça é composta por duas partes: (i) Módulo *worm* – componente responsável pela propagação da ameaça e instalação do módulo *ransomware*; (ii) Módulo *ransomware* – componente responsável pelas atividades de extorsão da ameaça. [Symantec 2017]

(i) Módulo *worm*

O módulo *worm* utiliza o *exploit* EternalBlue para explorar duas vulnerabilidades do serviço SMBv1 no Windows (CVE-2017-0144 e CVE-2017-0145), e assim disseminar o módulo *ransomware*. Estas vulnerabilidades impactam todas as versões do Windows que tenham o SMBv1 habilitado, porém a Microsoft publicou em 14/03/2017 atualizações para corrigi-las. O *exploit* EternalBlue fazia parte de um conjunto de ferramentas e arquivos roubados da NSA pelo grupo *hacker* Shadow Brokers e foram publicadas na internet em abril de 2017, após tentativas infrutíferas de vender o material vazado.

Quando executado, o módulo *worm* tenta contactar um dos domínios a seguir. Caso algum deles esteja alcançável, a execução é interrompida imediatamente. Esta funcionalidade atua como uma chave de desligamento (*kill-switch*), permitindo que a ameaça seja parada e, provavelmente, foi incluída para burlar sistemas de defesa que utilizam *sandboxing*, pois estes sistemas costumam responder a requisições para qualquer URL.

- iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
- ifferfsodp9ifjaposdfjhgosurijfaewrwergwea.com

Estes domínios não estavam registrados quando a ameaça foi lançada. Um pesquisador registrou os domínios assim que os descobriu, fazendo com que esta versão da ameaça parasse de se disseminar. Posteriormente, surgiram versões da ameaça com variações nestas URLs e até mesmo sem o recurso de *kill-switch*.

O *worm*, verificando que as URLs não estão responsivas, registra-se como um serviço no Windows e tenta instalar o módulo *ransomware* como um arquivo de sistema (%windir%\tasksche.exe). O módulo *ransomware* está embutido no próprio binário do *worm*, portanto não precisa ser baixado pela internet, como ocorre com outras ameaças.

O módulo *worm* tenta, em seguida, baixar uma ferramenta para conexão à rede anônima TOR e tenta se conectar a alguns domínios TOR. Estes domínios são utilizados apenas para rastrear as infecções e prover um endereço de pagamento Bitcoin único, bem como as chaves de decriptografia, caso a vítima pague pelo resgate. Esta funcionalidade, porém, possui um bug e não é executada corretamente.

O *worm*, então, tenta se propagar, realizando *scan* por portas 445 abertas nos seguintes endereços: (i) IPs da mesma subrede do computador infectado; (ii) **IPs gerados aleatoriamente** – este foi considerado um dos principais vetores de infecção global do WannaCry. Uma vez identificado um computador com a porta 445 aberta, o *worm* tenta explorar as vulnerabilidades já informadas e, caso obtenha sucesso, o novo computador será infectado e todo o ciclo recomeça.

(ii) Módulo ransomware

Ao ser executado, o módulo *ransomware* instala-se como um serviço, que será executado sempre que o computador for reiniciado. Os nomes das pastas e do serviço criados utilizam uma sequencia aleatória de caracteres. Inicia-se a rotina de criptografia:

 © 2017 Sociedade Brasileira de Computação

- O módulo procura por vários tipos de extensões de arquivos específicas: documentos, multimídia, bancos de dados, arquivos compactados, etc.;
- Cada arquivo é criptografado com um chave simétrica AES individual. Cada chave AES é criptografada individualmente usando criptografia assimétrica RSA, com chave de 2048 bits;
- Os arquivos criptografados têm a extensão ".WCRY" adicionada ao seu nome original. Exemplo: "foto.jpg" é criptografado para "foto.jpg.WCRY";
- O módulo tenta deletar *shadow copies* (Cópia de Sombra de Volume do Windows, uma espécie de *backup* integrado no sistema operacional) dos arquivos criptografados, de maneira a impedir a recuperação dos arquivos originais pelos usuários;
- Concluída a rotina de criptografia, são extraídos arquivos com as instruções de pagamento em diversos idiomas, além do próprio binário que permite descriptografar amostras de arquivos, e é exibida a "mensagem de resgate".

A lógica de pagamento do módulo *ransomware* previa a geração de um endereço Bitcoin único para cada usuário infectado, utilizando a rede TOR, porém, devido a um *bug* apenas três endereços Bitcoin são disponibilizados e compartilhados com todas as vítimas, de maneira que **o atacante não tem como determinar quem pagou pelo "resgate"**.

Estas características são de apenas uma das variantes da ameaça. Outras variantes foram descobertas após o ataque inicial e podem surgir novas derivações do *ransomware*.

4.2. O estudo de caso

As informações coletadas foram inicialmente descritas em ordem cronológica, com detalhamento dos fatos e ações tomadas. Em seguida as ações foram agrupadas em fases, de acordo com o processo de resposta a incidentes proposto por [Grispos 2016].

4.2.1. Preparação

A fase de preparação consiste na criação de um time de resposta a incidentes e o provisionamento de ferramentas e recursos que permitam responder adequadamente aos incidentes de Segurança da Informação.

A empresa do estudo de caso possui, há mais de 10 anos, uma gerência responsável por aspectos estratégicos e técnicos de Segurança da Informação na companhia e é responsável pela resposta a incidentes. Equipes formais dedicadas para detecção e resposta a incidentes de Segurança da Informação foram estabelecidas aproximadamente 8 meses antes do incidente com o WannaCry.

4.2.2. Identificação

A fase de Identificação está relacionada à detecção ou reporte de incidentes numa organização. A detecção do incidente, no caso estudado, deu-se no dia 12/05/2017, que, por convenção, será referenciado como **dia 1**. As seguintes ações ocorreram:

(i) Na **manhã do dia 1**, a equipe de monitoração detectou um serviço com nome "estranho" (conjunto aleatório de caracteres) em um servidor virtual; (ii) A equipe contatou o responsável pelo servidor; (iii) Foi detectado que o servidor estava infectado pelo

© 2017 Sociedade Brasileira de Computação

ransomware WannaCry e a equipe de resposta a incidentes foi acionada; (iv) Uma hora depois, foram confirmados 12 servidores virtuais, no mesmo segmento de rede, infectados; (v) Agentes HIPS (sistema de prevenção de intrusão nos *hosts*) detectaram tráfego suspeito oriundo de filiais. Posteriormente, foi identificado que os agentes HIPS já possuíam "assinatura" que identificava e bloqueava a exploração das vulnerabilidades no SMBv1, **um importante elemento de contenção ao ataque**.

4.2.3. Contenção

A fase de contenção consiste nas ações para evitar que um incidente sobrecarregue recursos, aumente o seu dano ou seja disseminado. As seguintes ações foram implementadas:

DIA 1: (i) Os primeiros servidores infectados detectados faziam parte de uma mesma subrede, que foi isolada com o objetivo de conter a propagação a partir deles; (ii) Foram bloqueadas as portas exploradas pelo *ransomware* nas conexões com redes de filiais; (iii) Segmentos onde a TI não possuía autonomia para atuar, mas que apresentaram tráfego suspeito, foram isolados; (iv) *Hosts* não isolados que não foram contemplados nos casos anteriores foram desligados, quando possível; (v) Foi configurado o bloqueio da execução de binários previamente conhecidos do *ransomware* (amostras da companhia e do próprio fornecedor de antivírus); (vi) Foi iniciada nova distribuição das atualizações que corrigiam as vulnerabilidades, com reinicialização obrigatória; (vii) A primeira "vacina" foi disponibilizada pelo fornecedor de antivírus.

DIA 2: (i) Foram identificados 3 mil *hosts* com risco potencial de contaminação – com antivírus desatualizado e sem as atualizações de segurança. Em decisão conjunta com corpo gerencial, o time de resposta a incidentes optou por desconectar da rede estes equipamentos como medida para evitar a propagação do *ransomware*;

DIA 3: (i) No fim do dia foi instalado um servidor Web interno e o DNS da companhia foi configurado para direcionar as chamadas às URLs do "*kill switch*" para este servidor, criando-se então um *honey pot* (uma espécie de armadilha para coletar informações sobre a ameaça). Este último recurso, embora simples, foi extremamente eficiente no tratamento do *ransomware*, por duas razões: a. Interrompeu o ataque (da variente original); b. Criou um mecanismo para identificar a origem de *hosts* infectados, gerando insumo para posterior tratamento (atuou como um recurso de IDENTIFICAÇÃO).

4.2.4. Erradicação

A fase de erradicação envolve a implementação de soluções para prevenir novas ocorrências do incidente. As seguintes ações de erradicação foram realizadas:

DIA 4: (i) Foram bloqueadas as extensões do WannaCry nos servidores de arquivos, com o objetivo de coibir ações de criptografia do *ransomware* em arquivos de rede; (ii) Foi iniciada rotina para detecção de máquinas suspeitas/vulneráveis e tratamento (24x7); (iii) Equipes de suporte local foram instruídas a aplicar os procedimentos necessários nos casos de máquinas suspeitas antes que elas pudessem ser religadas.

Ações da rotina de tratamento: a. Aplicação das atualizações de Sistema Ope-

 © 2017 Sociedade Brasileira de Computação

racional; b. Atualização do antivírus c. Reinicialização forçada dos equipamentos; d. Execução do *scan* completo do antivírus. Critérios adotados no tratamento: **Estações administradas**: tratamento remoto; **Estações não administradas**: desconectadas da rede; **Servidores**: delegado tratamento emergencial para os respectivos responsáveis; **Servidores sem identificação ou responsável**: porta de rede desabilitada.

4.2.5. Recuperação

A fase de recuperação consiste em restaurar os sistemas afetados e retomar a operação normal do negócio. As seguintes ações de recuperação foram observadas:

DIA 5 em diante: (i) Foi estabelecido um fluxo padrão para autorização do desbloqueio de portas que exigia sempre autorização do grupo de resposta a incidentes; (ii) Redes de filiais foram reconectadas gradativamente, na medida em que seus administradores reportavam status seguro. Eventualmente, em casos onde não foi possível atuação, mas a reconexão se fazia necessária, para minimizar os riscos foram liberadas portas de maneira seletiva, apenas para garantir a operação dos serviços essenciais. Na medida em que foram religadas, as redes ganharam monitoração especial para identificação de possíveis riscos; (iii) No processo de avaliação das redes de filiais, foram detectados vários problemas de padronização e arquitetura. Este processo de avaliação deveria ocorrer frequentemente, independente do processo de resposta a incidentes.

4.2.6. Acompanhamento

O objetivo principal da fase de acompanhamento é estabelecer lições aprendidas, disseminá-las internamente na organização e, eventualmente, para agentes regulatórios externos. Relatórios técnicos e gerenciais textuais, com detalhes sobre o incidente, lições aprendidas e recomendações, foram gerados poucos dias após o ataque.

4.3. Oportunidades de melhoria e contribuições

Mesmo sendo considerado um processo de resposta bem sucedido, várias oportunidades de melhoria foram observadas.

Na fase de identificação, inicialmente, foi desafiador detectar os *hosts* suspeitos. Os critérios e regras de detecção, bem como as corretas configurações das ferramentas, aconteceram na medida em que o incidente era tratado. Embora a escolha correta de ferramentas e configuração de regras e correlações sejam pontos fundamentais para a detecção de eventos, que é o cerne da resposta a incidentes, é recorrente a dificuldade das organizações de operar e amadurecer este processo [Ab Rahman and Choo 2015]. Visibilidade do ambiente é de suma importância, portanto **há carência de um modelo que auxilie as organizações no processo de seleção, configuração e operação de ferramentas de monitoração, bem como metodologias de avaliação da sua eficiência e maturidade**.

Na fase de **contenção**, observou-se falta de colaboração e pesquisa com atores externos. Um analista dedicado à pesquisa sobre o *malware* ou informações assertivas dos fornecedores de soluções de segurança teriam permitido a implantação do *kill switch* com

© 2017 Sociedade Brasileira de Computação

maior antecedência, já que a informação estava disponível desde o **dia 2**. As distribuições de atualizações e pacotes de *software* foram lentas. É preciso investir em ferramentas e processos que tornem mais eficiente esta distribuição.

Na fase de **erradicação**, houve problemas com a correta identificação de responsáveis pelos servidores no Banco de Dados de Gestão de Configuração (BDGC), ou seja, uma não conformidade com políticas gerais e boas práticas de Segurança da Informação. A equipe também não possuía privilégios adequados em diversos *hosts* que precisavam de tratamento, outro problema que poderia ser resolvido com a utilização de ferramentas e processos adequados.

Na fase de **recuperação**, foram identificadas diversas vulnerabilidades e desvios de padrões durante o processo de reconexão das filiais, permitindo a sua documentação e indicação das correções. Este foi um saldo positivo desta fase, porém também é um indicativo de que o processo de revisão e auditoria das conexões deve ser implantado como uma rotina sistemática e periódica.

No contexto do **acompanhamento**, ao longo do tratamento, não havia uma pessoa dedicada à documentação dos processos, tampouco um modelo de dados ou sistema de apoio para o seu registro. De uma maneira geral, nota-se que há carência de um modelo estruturado para a documentação do incidente. Por esta razão, a criação dos relatórios técnicos e executivos foi um processo lento que exigiu a colaboração de todos os envolvidos e a consulta a informações de fontes diversas, como e-mails e rascunhos em papel. Incidentes menos relevantes provavelmente não serão documentados ou não terão o mesmo nível de detalhes e escrutínio. **Este é um campo com grande oportunidade de melhoria**, uma vez que a documentação sistemática dos incidentes permite: (i) ater-se a detalhes importantes; (ii) obter estatísticas e conhecimentos diversos da base de incidentes após a sua conclusão e (iii) possibilidade de compartilhamento de informações anonimizadas com órgãos de defesa e outros grupos de resposta a incidentes.

Em um estudo com outra empresa da Forbes 500, [Grispos 2016] identificou que pequenas melhorias na qualidade da documentação do incidente trouxe ganhos significativos para o processo de resposta a incidentes. Por esta razão, optou-se por contribuir na dissertação com um *framework* de resposta a incidentes que foque na melhoria sistemática da documentação e registro dos incidentes, baseado nos achados de [Grispos 2016] e na ontologia proposta por [VERIS 2017].

O trabalho de [Ab Rahman and Choo 2015] corrobora com as conclusões sobre as oportunidades no contexto do acompanhamento, pois conclui que "não há estudo abrangente sobre o tratamento de incidentes" na literatura.

5. Conclusões e próximos passos

Ainda que mudem as motivações e as técnicas, os sinais indicam que as ameaças cibernéticas continuarão avançando. Por mais que se invista em soluções e processos preventivos, incidentes serão inevitáveis e é preciso preparar-se para respondê-los de maneira eficiente e, por isso, a produção acadêmica neste campo é tão importante. Com esta breve análise podemos, confirmar a hipótese de que a crise poderia ter sido pior caso o *malware* tivesse uma ou mais das seguintes características: (i) uma vulnerabilidade de dia zero – aquela sem correção disponível pelo fabricante; (ii) não tivesse o recurso de *kill switch*;

 © 2017 Sociedade Brasileira de Computação

(iii) adotasse comportamento discreto na fase de disseminação, com ativação futura programada (do módulo *ransomware* ou qualquer outro).

Este estudo de caso é parte de um trabalho maior, de dissertação de mestrado, onde pretende-se construir um *framework* de apoio para o desenvolvimento de processos e práticas relacionados à resposta a incidentes de Segurança da Informação.

Referências

Ab Rahman, N. H. and Choo, K.-K. R. (2015). A survey of information security incident handling in the cloud. *Computers & Security*, 49:45–69.

AYRAPETOV, D. (2017). Practical defense for cyber attacks and lessons from 2017 sonicwall annual threat report. 29 ago. de 2017.

CERT.br (2017). Estatísticas dos incidentes reportados ao cert.br - 2017. 16 ago. de 2017.

Cobb, S. (2017). Rot: Ransomware of things. 29 ago. de 2017.

Economist.com (2017). Ransomware attacks were on the rise, even before the latest episode. 29 ago. de 2017.

Formby, D., Durbha, S., and Beyah, R. (2017). Out of control: Ransomware for industrial control systems. 29 ago. de 2017.

Franceschi-Bicchierai, L. (2016). Hackers make the first-ever ransomware for smart thermostats. 29 ago. de 2017.

Gibbs, S. (2016). Ransomware attack on San Francisco public transit gives everyone a free ride. 12 fev. de 2017.

Grispos, G. (2016). *On the enhancement of data quality in security incident response investigations*. PhD thesis, University of Glasgow.

Healey, J. (2016). Winning and losing in cyberspace. In *International Conference on Cyber Conflict (CyCon)*, pages 37–49. IEEE. 08 fev. de 2017.

Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., and Kirda, E. (2015). *Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks*, volume 9148, pages 3–24. Springer International Publishing, Milan.

O Globo (2015). Investimento em segurança da informação cresce mais no país - 2015. 08 fev. de 2017.

Reuters (2017). Analysis: Wannacry attack shows trend toward 'economic' cyber threats, rising regulatory risk. 28 ago. de 2017.

Smith, M. (2016). Kansas Heart Hospital hit with ransomware; attackers demand two ransoms. 12 fev. de 2017.

Symantec (2017). Ransom.wannacry technical details. 31 ago. de 2017.

Trend Micro (2016). Trend micro's definition of ransomware. 12 fev. de 2017.

VERIS (2017). Vocabulary for event recording and incident sharing. 27 mar. de 2017.

Winton, R. (2016). Hollywood hospital pays $17,000 in bitcoin to hackers; FBI investigating. 12 fev. de 2017.

Yin, R. K. (2015). *Estudo de Caso: Planejamento e Métodos*. Bookman editora.

© 2017 Sociedade Brasileira de Computação

Resumos Estendidos - Seção Técnica 2 - Intrusão - detecção, prevenção e resposta & Criptografia e Criptoanálise & Aplicações & Hardware

© 2017 Sociedade Brasileira de Computação

Keyblock: a software architecture to prevent keystroke injection attacks

C. David B. Borges, J. Rafael B. de Araujo,
Robson L. de Couto, A. Márcio A. Almeida

Computer Engineering – Universidade Federal do Ceará (UFC)
Campus Mucambinho – Sobral – CE – Brazil

david.borges@protonmail.com, rafael.acurcio@gmail.com
robsoncouto@alu.ufc.br, marcio.albu@alu.ufc.br

Abstract. *This work investigates a solution to mitigate the threat of keystroke injection attacks. Current defense mechanisms often require relatively expensive hardware and time consuming configuration. We describe and test the effectiveness of a software layer between USB input hardware and processes. Our software, Keyblock, intercepts events from newly connected devices and uses keystroke dynamics analysis to detect whether an attack is in course. By detecting and immediately disabling devices with anomalous typing patterns, Keyblock provides a software-only automatic solution to prevent keystroke injection.*

1. Introduction

The widespread use of USB technology has advantages and disadvantages. One advantage is that plug-and-play, a requirement in the USB specification, makes the attachment and setup of new USB devices a straightforward task. Nevertheless, automatic hardware configuration can be exploited, specially when concerning human interface devices (HID), such as keyboards. It is possible to disguise a microcontroller as a USB keyboard by providing the target computer with a fake HID descriptor. An attacker can exploit this security flaw by setting up a device to enter predefined keystrokes into a shell terminal running on a target computer [SRLabs 2014]. This attack is known as keystroke injection.

In [SRLabs 2014], flaws in USB plug-and-play were exploited with the use of a microcontroller and a storage device coupled together. The authors demonstrated attacks in which an adversary could gain administrative access to Windows and GNU/Linux machines. These attacks were able to install a keylogger in the target computer, gain root access and infect other USB devices. These vulnerabilities also acquired popularity in the hacker community. A tool that exploits USB plug-and-play by HID emulation, known as Rubber Ducky, is available online. The device, which includes a 60 MHz 32-bit CPU and Micro SD storage, is disguised as a regular flash drive and programmed to perform keystroke injection attacks when connected to a computer. In fact, these attacks can be performed using any USB microcontroller capable of USB HID emulation.

It is important do develop defensive measures against this threat. In this paper, we describe, implement and test the effectiveness of a software layer between USB input hardware and processes. Our program, Keyblock, intercepts events from newly connected keyboards and detects whether an attack is in action by performing keystroke dynamics analysis. Our goal is to classify and automatically disable devices that present suspicious typing patterns, a characteristic of keystroke injection attacks. The main contribution of

© 2017 Sociedade Brasileira de Computação

this paper is a software-based keystroke injection detector with 0% error rate on prevalent attacks and near 0% false positives for standard keyboards. Moreover, Keyblock does not interfere with plug-and-play and requires no user configuration.

2. Related Work

The USB flaws that allow unwanted HID emulation and keystroke injection have drawn attention from the security research community and some protection mechanisms have been proposed. One measure to mitigate this threat is to require user interaction before allowing an HID to generate events that target other applications. This measure has been proposed and implemented in software and hardware.

[GData 2014] distributes a program that blocks events from new keyboards until permission is granted when the user clicks a pop-up message. Another, more robust, software-based approach is GoodUSB, a mediation architecture in the USB stack, proposed by [Tian et al. 2015]. It prompts the user to define the capabilities of new USB devices and restricts access to drivers that perform any other unwanted functionality, effectively preventing HID emulation.

Another solution, USBCheckIn, is a hardware-based HID protection system, proposed by [Griscioli et al. 2016], that asks the user to solve a random challenge before forwarding events to the computer's USB ports. Hardware interfaces between a computer system and USB peripherals are proposed by [Kang and Saiedian 2015] and [Loe et al. 2016]. They implement USBWall and SandUSB, respectively, systems that enumerate newly connected devices and allow the user, via software, to verify whether the peripheral is trusted before allowing it to operate. SandUSB has the advantage of detecting some attacks using keystroke dynamics analysis.

Different algorithms and applications have been proposed for keystroke dynamics analysis. User authentication by analysis of typing patterns is proposed by [Trojahn and Ortmeier 2013]. They classify users with attributes such as key press timing and interval between different key presses. Their results show that it's possible to obtain error rates below 2% using a mixture of keystroke and handwriting recognition. Not limited to physical keyboards, their study presents data related to touch screen devices including features such as pressure and swipe distances. [Killourhy and Maxion 2009] and [El-Abed et al. 2014] perform benchmarks of the main algorithms for anomaly detection and user biometrics, respectively, based on keystroke patterns. The use of keystroke analysis for detection of HID attacks was first proposed by [Barbhuiya et al. 2012]. The authors propose one possible architecture, useful keystroke parameters and metrics.

The majority of the cited solutions to alleviate keystroke injection risks depend on relatively expensive hardware, such as [Griscioli et al. 2016], [Loe et al. 2016] and [Kang and Saiedian 2015]. Others, such as [GData 2014] and [Tian et al. 2015], require a set of configurations steps to be performed for every new device. One of the goals of the proposed software, Keyblock, is to allow software-based keystroke injection detection with no required configuration steps. The work by [Barbhuiya et al. 2012] is the closest to our proposed solution, as it primarily uses keystroke dynamics to perform decisions. However, Keyblock does not require additional steps or configuration such as capturing keystroke samples and creating a user profile. In addition, our software uses a different detection architecture and a simpler classification method.

 © 2017 Sociedade Brasileira de Computação

3. Principles of Designing Secure Systems

Any device that complies with the USB specification is vulnerable to HID emulation and keystroke injection attacks. Current anti-virus software is not able to protect against this threat [Tian et al. 2015]. But these plug-and-play exploits exist because current USB standards ignore certain security principles. When considering computer security, it is important to continuously reexamine all assumptions about threat agents and the environment. The principle of complete mediation states that every access and every object should be checked every time [Beznosov 2015]. A system should regard the environment where it operates as inherently hostile and be reluctant to trust [McGraw 2013]. In the case of USB HID authentication, the object is a new USB device that requests access to the USB bus as an HID. While configuring a new device, the only verification is related to device class. However, because class information is provided by the device itself, it cannot be considered reliable. After the drivers are loaded and the device is configured, there is no further verification concerning its behavior. Therefore, current USB protocols neglect the principles of questioning all assumptions, complete mediation and reluctance to trust, thereby giving rise to a security flaw. Keyblock fills this hole and allows a system to continuously verify if a device should be trusted or not.

4. Keyblock and monitor threads

Keyblock was written in C/C++ and, currently, only supports GNU/Linux systems. Its purpose is to be run as a daemon, requiring no interaction or configuration by the user. The main loop of Keyblock continuously read the list of available input devices located in /dev/input/ and starts a monitor thread for every new device.

A monitor thread is responsible for reading every input event from a given device and decide whether it should be blocked or continue to operate normally. The operation of a monitor thread is summarized in figure 1. As a security measure, no key event is allowed to reach applications running on the X system, or graphical user interface, without analysis from the monitor thread. Keyblock uses the ioctl system call in order to configure a mechanism for monitor threads to grab and consume all events from a device. This prevents a key press from directly reaching any X application. The thread registers key press timestamps with nanosecond precision and only keeps in memory a window containing the last N events.

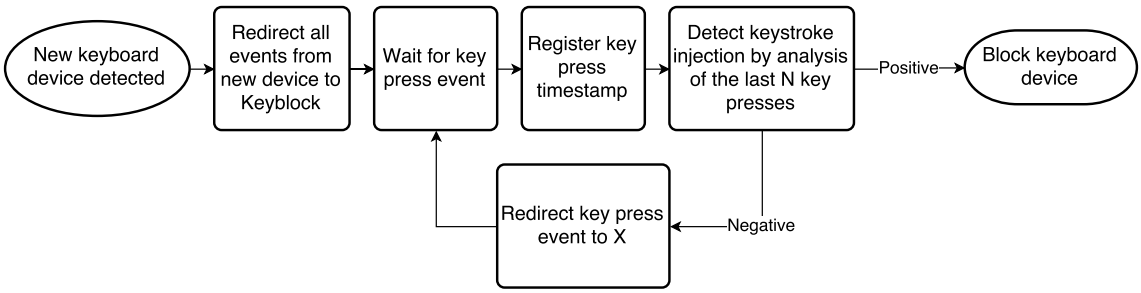


Figure 1. Monitor thread operation

When a new key press is registered, the monitor thread executes the detection phase. This phase is represented by a function that receives data about the last N key events and returns true, if it identifies a possible attack, or false, if there is no unusual

© 2017 Sociedade Brasileira de Computação

behavior. In the latter case, the key event is redirected to the X system. However, if unusual typing patterns are detected by a monitor thread, the corresponding device is automatically blocked until removal.

5. A simple detection model

The attack detection phase seen in figure 1 can be implemented in many ways. In principle, it is a function with a fixed length input vector that returns true or false. If true, the monitor thread will block the device, else, it will redirect the key press to the X system. This description allows great flexibility. One can use keystroke latencies or hold times as input and perform classification using any of the available methods in the literature. For our experiments, we developed two very simple classifiers, derived from empirical analysis of two keystroke latency datasets: one produced by humans while typing and another generated by our rogue microcontroller during keystroke injection tests.

The first dataset, $\mathcal{D}_1$, provides keystroke latencies from human writers and was collected by [Calot 2015]. It contains delays between combinations of successive key presses and releases, gathered during the period of one week. From this dataset, one is able to extract statistics of human keystroke latency and hold time. Following this approach, we generated a second dataset, $\mathcal{D}_2$, containing similar information to the first, in order to characterize keystroke injection attacks, using our test tool, an ATMEGA32U4.

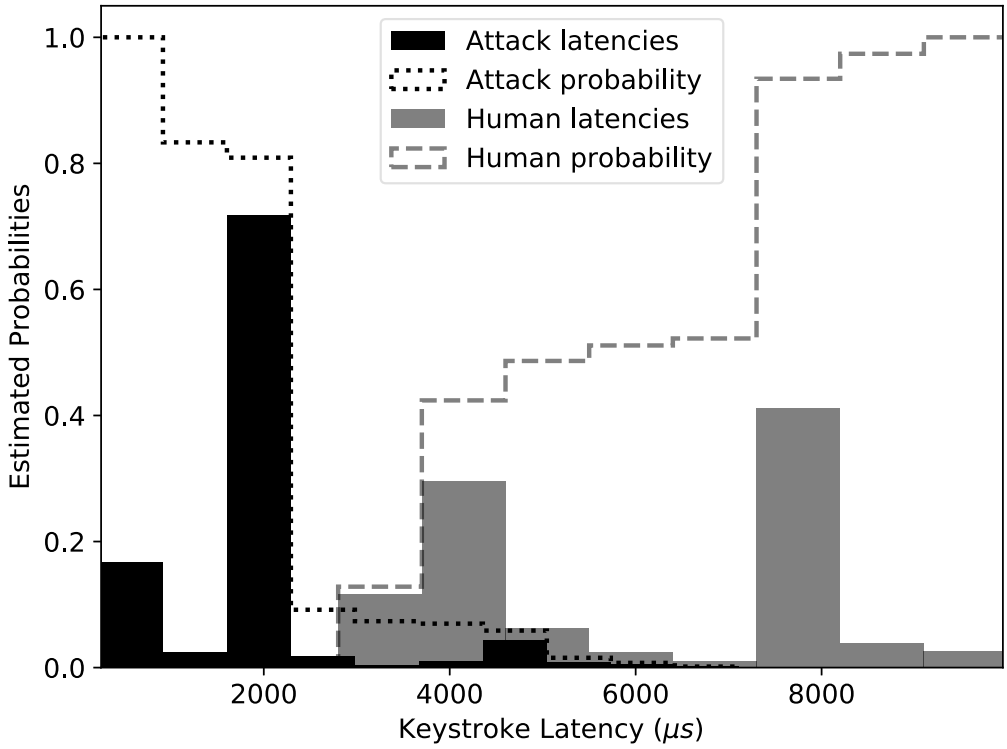


Figure 2. Estimated probabilities of keystroke injection attacks and human operators for latencies below 10 ms from $\mathcal{D}_1$ and $\mathcal{D}_2$

.

Our primary detection model derives from the observation that keystroke injection attacks usually have the purpose of introducing a payload as fast as possible, thus leaving

 © 2017 Sociedade Brasileira de Computação

a low latency footprint. Therefore, in principle, it's possible to discriminate between a human operator and an attack tool by defining a latency threshold. Figure 2 shows the lowest human typing latencies registered in $\mathcal{D}_1$, along with attack latencies in $\mathcal{D}_2$. The dashed line is a cumulative density function (CDF) matching the observed human latencies. The dotted line is a reverse CDF based on the latency distribution for keystroke injection attacks. From figure 2, according to latencies from $\mathcal{D}_1$ and $\mathcal{D}_2$, we see that there is an overlap between both distributions, specially in the interval from $3\ ms$ to $5\ ms$. For this reason, using latency alone, it is difficult to decide whether a set keystrokes are the product of an attack or a fast typing human.

If we fit a normal distribution, with mean μ and standard deviation σ, to the observed human latencies below $50\ ms$, the result is $\mu = 31.64\ ms$ and $\sigma = 9.63\ ms$. Our detection model takes into account how far the last N latency samples deviate from the expected human latency distribution. In this case, only left-side deviation from μ is considered, as arbitrary deviations to the right are common when a human operator is typing. Another characteristic of keystroke injection attacks is that keys are injected at an approximately steady rate. Our detector is improved by measuring the standard deviation of the last N keystrokes and comparing to the σ of an expected distribution.

Our detector is composed of two thresholds. The deviation threshold τ_d computes how far is the average of the last N latencies to μ; it is triggered if keystrokes are fed to the system with a much higher frequency than the average expected for a human. The stability threshold τ_s measures how stable are the last keystroke latencies; it is triggered if latencies have near constant values,producing a standard deviation much lower than the expected σ. The current Keyblock implementation defines $\tau_d = \mu - 3\sigma$ and $\tau_s = 0.1\sigma$.

6. Experiments

We tested how effectively Keyblock is able to detect keystroke injection attacks and how fast they are blocked using the model described in section 5. We used an ATMEGA32U4 to emulate a USB keyboard and inject a set of keystrokes in a target computer that runs Keyblock. Our simulated attacks mimic the behavior of standard keystroke injection tools, such as Rubber Ducky and Teensy. The ATMEGA32U4 was chosen as test platform, instead of the alternatives, Rubber Ducky and Teensy, because the former is cheaper and allows great programming flexibility. The ATMEGA32U4, which is USB 2.0 compliant, has one 8-bit processor core, can run up to 16 MIPS and has 32 kB of program flash memory [Atmel 2015].

We performed 100 keystroke injection experiments and registered whether each attack was successfully blocked and how many keystrokes were accepted before the offending device was disabled. Moreover, we tested if Keyblock could effectively identify and allow humans to operate normally with standard USB keyboards. The process of connecting a new keyboard and typing at different speeds was repeated 114 times. The resulting false negative and false positive rates are summarized in table 1. Our tests included two deviation threshold values, namely $\tau_d = \mu - 2\sigma$ and $\tau_d = \mu - 3\sigma$. A false negative happens when Keyblock incorrectly allows a keystroke injection attack to proceed. A false positive is registered if Keyblock erroneously disables a newly connected keyboard while a human is typing.

For $\tau_d = \mu - 2\sigma$, we performed 50 keystroke injection tests with an AT-

© 2017 Sociedade Brasileira de Computação

MEGA32U4 and 64 tests with typing humans. No attack was allowed to proceed, however we registered that 12 out of 64 human keyboard sessions were blocked, implying a 18.75% false positive rate. This means that almost 1 in 5 human sessions were erroneously closed. For $\tau_d = \mu - 3\sigma$, we carried out 50 keystroke injection tests and 50 human keyboard sessions. Again, there was a 0% false negative rate. However, the false positive rate was reduced to 2%, as only 1 human session was blocked. Additionally, in these experiments, not a single keystroke injected by an attack was incorrectly allowed to reach the X system.

Table 1. Results

	$\tau_d = \mu - 2\sigma$	$\tau_d = \mu - 3\sigma$
False negatives	0%	0%
False positives	18.75%	2%

7. Conclusion

The effectiveness of Keyblock's detection architecture (figure 1) was demonstrated in our experiments. This conclusion is drawn from the fact that Keyblock, using a relatively simple detection model, can acquire a 0% false negative rate while keeping false positive rates as low as 2%. The detection model used in the experiments was based on two thresholds, derived from latency distributions measured for standard keystroke injection attacks and human operators. Notwithstanding its simple classification method, it was able to block the most prevalent category of keystroke injection attack, in which key events are generated at speeds above that of a human, with a near constant frequency. More sophisticated attacks can be developed, such as imitating human typing patterns. Such an attack would hardly be blocked by a detection model based on latency only. However, our software architecture offers extensibility for implementation of new keystroke parameter measurements and classification techniques. The next step in this project is to develop and test other keystroke statistics, like key hold time, error rate and general typing habits, and more robust classification methods, such as linear discriminant analysis, support vector machines and neural networks. Although our current implementation is focused on GNU/Linux, the same architecture can be implemented as a tool for other operating systems, including Windows and Mac. [Keyblock 2017] is released as open-source software under the CC Attribution-NonCommercial 4.0 license.

References

Atmel (2015). Atmega32u4 datasheet. [Available] http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf [Access 30-07-2017].

Barbhuiya, F. A., Saikia, T., and Nandi, S. (2012). An anomaly based approach for hid attack detection using keystroke dynamics. In *Proceedings of the 4th International Conference on Cyberspace Safety and Security*, CSS'12, pages 139–152, Berlin, Heidelberg. Springer-Verlag.

Beznosov, K. (2015). Computer security: Principles of designing secure systems. [Available] http://courses.ece.ubc.ca/cpen442/sessions/08-design_principles.pdf [Access 30-07-2017].

 © 2017 Sociedade Brasileira de Computação

Calot, E. P. (2015). Keystroke dynamics keypress latency dataset. Database.

El-Abed, M., Dafer, M., and Khayat, R. E. (2014). Rhu keystroke: A mobile-based benchmark for keystroke dynamics systems. In *2014 International Carnahan Conference on Security Technology (ICCST)*, pages 1–4.

GData (2014). Usb keyboard guard: How to be sicher from usb attacks. [Available] https://www.gdatasoftware.com/en-usb-keyboard-guard [Access 30-07-2017].

Griscioli, F., Pizzonia, M., and Sacchetti, M. (2016). Usbcheckin: Preventing badusb attacks by forcing human-device interaction. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 493–496.

Kang, M. and Saiedian, H. (2015). Usbwall: A novel security mechanism to protect against maliciously reprogrammed usb devices. *Information Security Journal: A Global Perspective*.

Keyblock (2017). Keyblock. [Available] https://github.com/cosmonautd/Keyblock [Access 17-10-2017].

Killourhy, K. S. and Maxion, R. A. (2009). Comparing anomaly-detection algorithms for keystroke dynamics. In *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, pages 125–134.

Loe, E. L., Hsiao, H. C., Kim, T. H. J., Lee, S. C., and Cheng, S. M. (2016). Sandusb: An installation-free sandbox for usb peripherals. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 621–626.

McGraw, G. (2013). Thirteen principles to ensure enterprise system security. [Available] http://searchsecurity.techtarget.com/opinion/Thirteen-principles-to-ensure-enterprise-system-security [Access 30-07-2017].

SRLabs (2014). Badusb: On accessories that turn evil. [Availbale] https://srlabs.de/wp-content/uploads/2014/11/SRLabs-BadUSB-Pacsec-v2.pdf [Access 30-07-2017].

Tian, D. J., Bates, A., and Butler, K. (2015). Defending against malicious usb firmware with goodusb. In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, pages 261–270, New York, NY, USA. ACM.

Trojahn, M. and Ortmeier, F. (2013). Toward mobile authentication with keystroke dynamics on mobile phones and tablets. In *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 697–702.

© 2017 Sociedade Brasileira de Computação

A publicly verifiable protocol for random number generation

João Penna[1], **Jeroen van de Graaf**[1]

[1]Universidade Federal de Minas Gerais

{joaopenna, jvdg}@dcc.ufmg.br

Abstract. *Chance plays an essential role in many decision procedures such as lotteries, draws etc. As such procedures are moving on-line, several web services offering randomness have appeared over the last few years. NIST's randomness beacon, which publishes a sequence of 512 random bytes every minute, unfortunately lacks transparency: the beacon does not eliminate the possibility of an insider attack who knows the outcomes beforehand. We propose an improvement of NIST's beacon which is publicly verifiable and fully transparent: any outsider who did not witness the bit generation in person but has internet access can convince himself that the beacon acted honestly, provided he can be sure that fresh, independent random bits were contributed to the seed value. Our proposal is based on a novel cryptographic assumption: the existence of functions that are slow to compute even on the fastest supercomputers.*

1. Introduction

1.1. The NIST Randomness Beacon

Since 2011 there exists a public service of randomness in the United States provided by the National Institute of Standards and Technology (NIST) called the Randomness Beacon [Fischer et al. 2011]. The concept of a beacon as a randomness source was originally proposed by [Rabin 1983]. Every 60 seconds, NIST produces a sequence of 512 random bytes and publishes on the internet on their website. Each sequence also includes a timestamp, a digital signature, and the hash of the previous value. This way one can verify that it was generated by NIST, and it is also possible to detect tampering and retroactive changes on values.

But can one really trust NIST's randomness beacon? Perhaps the bits could be doctored through some insider attack. Or perhaps they were generated a year ago and are only being published now, meaning that an insider could know beforehand what values would come up at specific times. This could be just paranoia, but NIST has been infiltrated before by the NSA, as Edward Snowden revealed. And since there exist protocols which can eliminate these doubts, why not implement them?

1.2. Objective of this paper

The main objective of this paper is to present an alternative to NIST's randomness beacon that provides optimal verifiability. NIST's beacon uses a very sophisticated, quantum-based randomness source [NIST 2016], but it is not verifiable. It might appear to have high entropy, but it is not feasible to physically inspect it, meaning one has to trust the output values to be random and not being influenced on particular occasions. Furthermore, it is not possible to assert the freshness of the values generated. Insiders might have access to the random values long before they are published.

 © 2017 Sociedade Brasileira de Computação

We approach this issue in the following way. We imagine a service in which any person or entity can contribute in a verifiable way to the random seed value z, for instance by hashing all the contributions. The output of the beacon will be $F(z)$, where F is a function with special properties, to be discussed below.

If you have a sequence z of $N = 4096$ random bits and apply a permutation F over $\{0,1\}^N$ on z, then the result $F(z)$ still will be random. Usually in cryptography we require F to be easy to compute and hard to invert. But if the input z is composed of many contributions, the person who provides the last contribution could try to influence the final outcome by choosing a contribution resulting in an $F(z)$ with special properties.

To thwart this attack we impose a different requirement on F: that F is **slow** to compute. In particular, in the scenarios outlined below we assume it takes at least 1 minute to compute $F(x)$, even with the fastest supercomputers available today. However, it would be very inefficient if would take hours to compute $F(x)$ on an ordinary workstation with a fast processor.

Finding a suitable F is not an easy endeavor, but our efforts were greatly simplified by the realization that the function Sloth introduced by Lenstra and Wesolowsk [Lenstra and Wesolowski 2015] is perfectly suited for this task. Using their techniques it is possible to define a function which takes 5 minutes to compute on an ordinary computer, while taking at least 1 minute on the best supercomputers available.

The goal of their paper is different, however: they present a transparent and verifiable way to produce the parameters of elliptic curves to be used in standards, thus avoiding any suspicion of tampering. Our contribution is to extend their techniques in order to provide a service equivalent to the NIST randomness beacon, but which provides verifiability.

Another approach is taken in the recent paper of [Syta et al. 2017], which contains excellent references to related work. Their goal is to develop a beacon scalable to many hundreds of parties connected to the internet, and is available as an on-line service at https://pulsar.dedis.ch. Their cryptographic assumptions are different from ours.

This paper is structured as follows: the next section provides additional background on the need for verifiable public randomness. Section 3 discusses a two-party setting based on the coin-flipping by telephone problem, which is extended to a multi-party setting in Section 4. By then we can define the properties needed for the function $F()$, and in Section 5 we show that Sloth from [Lenstra and Wesolowski 2015] addresses these needs. In Section 6 we specify our protocol for a new randomness beacon equivalent to NISTs but providing verifiability, succeeded by a brief discussion about the protocol's security properties.

2. Background

Suppose you need to make a random choice. Perhaps you could use a die to decide. But the die could be loaded; how could you know if the roll had no bias at all? How do you know if the lottery tickets are really randomly drawn? How could you tell if a physical random bit generator worked correctly without there being a way of influencing the result on particular occasions? It may be difficult to answer these questions. Take for instance the 1980 Pennsylvania Lottery Scandal, in which balls with different

© 2017 Sociedade Brasileira de Computação

weights were used to greatly increase the chance that certain numbers would be chosen [Togyer 1999]. One other lottery had its draw corrupted in the form of bribed children and marked balls [BBC 1999]. Random numbers generated by computer can also be tampered with [Simmons 2015].

2.1. An example: how to implement a verifiable random draw

Fortunately, in the digital world, cryptography can offer solutions. Suppose you teach a crypto class, and you have to make a random list of the order in which the students will present their final project. One transparent way to do this is as follows: you make a list with the name of the students and you concatenate it with the same, randomly chosen number r; you compute SHA256(name+r); and sort the list by the hash value obtained. Draws are possible but very unlikely; in fact, a draw implies a collision in SHA256 which would imply a nice publication.

Now a student could argue that you manipulated the order by testing various random numbers and reveal only the one corresponding to the order you liked. To avoid this argument, you agree to use an externally generated r to be published at some exact moment in the future, like one obtained from NIST's randomness beacon. Now a very paranoid student can argue that these bits may be random, but that you have an insider working at NIST who manipulated those bits for you. This is the problem we are trying to solve here.

The case of the algorithm which distributes court cases to the Supreme Judges of the *Supremo Tribunal Federal* is not essentially different. It is a well-known theoretical result that a randomized algorithm $R(x)$ can be modeled as a deterministic algorithm $D(x, r)$, where x is the same input and r is randomly chosen bit string of appropriate size.

The random numbers produced by NIST are of length 2048 bits, which already is a lot. However, if this would be deemed insufficient, one could use this r as the seed of a pseudo-random bit generator, like Blum-Blum-Shub, to produce an unlimited supply of bits which no poly-time algorithm can distinguish from truly random bits.

So the question is, where do these initial random bits r come from? What we need is a randomness service that is publicly verifiable on the internet, meaning that it is verifiable even without being present in person to witness the generation process.

3. Coin Flipping by Telephone

To explain our idea let us first consider two parties A and B who want to produce a random sequence $c \in S$, where $S = \{0,1\}^{2048}$. Suppose for now we dispose of a function $F : S \to S$ that takes 5 minutes on A's computer and which cannot be parallelized; how we obtain such a function will be discussed in Section 5. Then A and B can execute the following protocol:

1. A chooses a random bit string $a_0 \in S$, sends it to B and start computing $F(a_0)$.
2. B chooses a $b_0 \in S$ at random and sends it to A
3. The outcome $c \in S$ is defined as $c = F(a_0) \oplus b_0$, where $\oplus$ denotes bitwise XOR.

So after 5 minutes they have the outcome. But what if B has a faster computer than A? Then he could also compute $F(a_0)$ quickly, and choose a value for b_0 to obtain

 © 2017 Sociedade Brasileira de Computação

a c that he likes. Well, B's computer may be faster, but by an appropriate choice of F and doing some hardware estimates (discussed in Section 5 and the paper cited), one can conclude that B cannot be more than five times faster. So A accepts Step 2 only if B sends it within 1 minute of her having sent the message in Step 1, otherwise she aborts. This time constraint guarantees that B cannot know $F(a_0)$ and therefore cannot bias the value of c.

Observe that in the traditional protocol of Coin-flipping by Telephone [Blum 1983], A will send a string commitment to a_0 in Step 1. At some later stage she unveils a_0 by opening the commitment; in that case the final result of the coin flip is defined as $a_0 \oplus b_0$. In the traditional case there are cryptographic restrictions on the commitment, which must be hiding and binding.

Note that in our protocol the function F could be invertible. *Our cryptographic assumption is that F takes at least one minute to compute even by the fastest supercomputers on the planet, while taking 5 minutes on an off-the-shelf PC with a fast processor.* Another difference is that in the traditional protocol A can wait as long as she wants to open the commitment, whereas is our protocol there is a time window in which she must accept or reject B's message in Step 2. Otherwise she is vulnerable to an attack of Bob.

The advantages of this approach may not be too clear in a two-party setting, but becomes more obvious when there are passive observers who have a stake in the outcome.

4. A multi-party setting with a beacon

We now change the setting: A and B rely on a trusted randomness beacon Z. As a very naive first approach, suppose that Z chooses an initial seed z and computes $H(z)$, where H is some very fast hash function like SHA256 modified to have a 2048-bit output.

If A is suspicious of Z secretly colluding with B to bias the outcome, this approach cannot remove this suspicion, because Z could try thousands of values for z, compute $H(z)$ for each and choose the value that suites B best. One way to allay A's suspicion is to allow her to contribute to the random seed with her value a. Say that the result is $H(z, a)$. The problem we now face is this: who is last, Z or A? Suppose that Z is last. Then, given a, he can still apply the same attack as before, searching for some suitable z. But if A is last to provide her value, she could mount this attack. Using a very fast function H, it is not possible to eliminate this possible attack by the party who is last.

However, substituting H for a function F which is deliberately slow in terms of wall clock time, the situation changes. Assume that $F(z, a)$ is a very slow function and enforce that any contribution must be submitted inside a small time frame before the computation starts. If at least one of z, a makes an honest contribution not known to the others previously, it would be impossible to compute F for many input values and choose another contribution that would generate a specific output.

5. Choosing a function that takes a long time to compute

Choosing the right function F is a subtle process but is possible. A simple solution is defining $F(n)$ as n consecutive iterations of a function G:

$$F(x) = \underbrace{G(G(G(G(x)...)))}_{n} = G^{(n)}(x)$$

© 2017 Sociedade Brasileira de Computação

Since this calculation is inherently sequential, n can be appropriately chosen to ensure it requires 5 minutes of wall clock time to compute.

One possible candidate for G would be Argon2 [Biryukov et al. 2016], a memory-hard function for password hashing. It has very interesting features, including that it is optimized for the x86 architecture, this way specialized hardware such as a FPGA board, an ASIC circuit or a GPU board do not gain a big advantage at runtime. Unfortunately, the downside is that the verification would take the same amount of time to compute. A more elegant approach is presented below, using a function for which the inverse is easy to compute: given $F(z)$ it is fast (a few seconds) to invert F and compute z.

5.1. Sloth

Sloth is a function proposed by [Lenstra and Wesolowski 2015] which takes a long time to compute, but its result can be verified very quickly. In other words, calculating G is slow, but calculating its inverse G^{-1} is quick.

Sloth is based on computing the square root of a number modulo a prime p. More specifically, for a prime p the finite group of p elements is denoted by Z_p, and its multiplicative group is denoted by Z_p^*, where we identify the elements with the integer between 1 and p. Let p be a large prime number congruent to 3 modulo 4. It follows that for any x in Z_p^* precisely one of x and $-x$ is a square, and a square root x can be computed by calculating $x^{\frac{p+1}{4}}$. It is generally believed that this computation cannot be done faster than using $log_2(p) - 2$ unparallelizable modular squarings, but computing the inverse, for verification, involves just a single modular squaring.

Recall that x has two square roots: if y is a root then so is $p - y$; consequently one of them is even and the other odd. To determine which of the two square roots will be used as the result of the computation, the following convention is used: If x is a quadratic residue, take its even root; if not take the odd root of $-x$, which is a square. Whether x is a quadratic residue can be checked using the Legendre symbol: if $x^{\frac{p-1}{2}} = 1$, then it is a quadratic residue, and if $x^{\frac{p-1}{2}} = -1$ it is not. Note that this turns $G(x)$ into a permutation on Z_p^*.

To avoid algebraic shortcuts in computing $G^{(n)}$, each iteration also contains a additional permutation. By using a block cipher such as AES (Advanced Encryption Standard), attacks that would explore the algebraic structure of the calculation are avoided. Note that when using this kind of permutation there is a possibility that it makes the number bigger than the prime p chosen. Depending on the choice of p, this chance is extremely small, but if it does happen then executing another permutation until the result generates a number smaller than p solves the issue.

The result of this calculation is an output that results in high quality random bits. The output is also used to calculate the inverse function, allowing one to check whether the result corresponds to the computation of the inputs given, thus verifying if the whole process was done correctly. The output is deterministic and will always yield the same result for the same inputs and parameters, but in our protocol the output cannot be predicted during the first minute.

 © 2017 Sociedade Brasileira de Computação

6. A Publicly Verifiable Randomness Beacon

In this section we present a novel construction of a randomness beacon with the following properties:

- The beacon produces 2048 bits every minute. These bits will be signed digitally.
- Any party who wishes can contribute a (random) string which is used as input to the initial value z_i for the process that starts at minute i.
- Every minute a process Z_i is started which computes $F(z_i)$, which takes 5 minutes to complete.
- To this end we have 5 parallel processors, T_1 to T_5, which work in pipelined fashion: at minute i processor $T_{(i \bmod 5)}$ yields the result of the process started 5 minutes earlier, $G(z_{i-5})$, and starts computing $G(z_i)$.

A diagram illustrating this scheme is shown in Figure 1. As previously explained, if a party is suspicious that the generation might be corrupted, submitting an honest contribution prevents other parties from forcing a specific result. While this makes the output unpredictable by the start of Z's computation, note that the output is deterministic and always the same if the same inputs are given, thus enabling anyone to verify afterwards if the computation was done correctly. Another positive side effect of an honest contribution is that one can be assured that the generated value is fresh. Of course, all of this relies on the fact that there exist a function that guaranteedly takes a long time to compute.

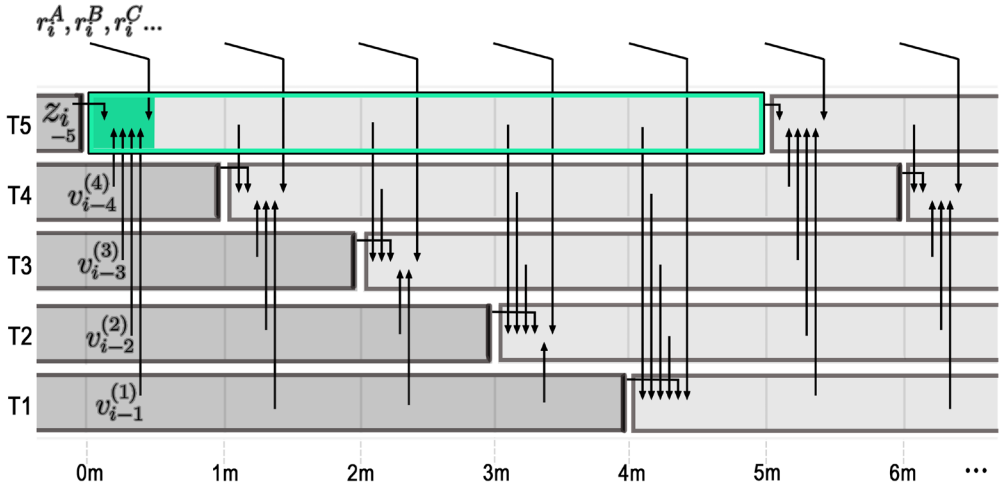


Figure 1. At the start of every minute, about one second is spent synchronizing the inputs from the values contributed from outsiders, intermediate values from the four parallel process, and the final value from the process that just ended

We now describe the actions of Z which implements a beacon with these properties, assuming that all data it publishes is authenticated. At the beginning of minute i:

1. Z collects all random external contributions $r_i^A, r_i^B, r_i^C, \ldots$ submitted by outsiders A, B, C, ... for minute i;
2. Z receives the intermediate values v of the four parallel processes run on the other cores, and the final value of the process that just ended running on the same core. Let v_i^j be the intermediate value of computing $F(z_i)$ after $j = 1, 2, 3, 4$ minutes.

© 2017 Sociedade Brasileira de Computação

Observe that superscripts represent indexes, not exponents. Then the values received by Z can be represented as $v_{i-1}^{(1)}, v_{i-2}^{(2)}, v_{i-3}^{(3)}, v_{i-4}^{(4)}, z_{i-5}$;
3. Z produces his own seed value s_i;
4. Z uses some publicly known hash function H to reduce all these values to 2048 bits: $z_i = H(r_i^A, r_i^B, r_i^C ..., v_{i-1}^{(1)}, v_{i-2}^{(2)}, v_{i-3}^{(3)}, v_{i-4}^{(4)}, z_{i-5}, s_i)$;
5. Z publishes $BC(z_i)$ where BC is some unconditionally hiding commitment scheme;
6. Z dispatches a new process to compute $F(z_i)$ om processor $T_{(i \bmod 5)}$;
7. After $j = 1, 2, 3, 4$ minutes Z publishes the intermediate values $v_i^1, v_i^2, v_i^3, v_i^4$, resp.
8. After 5 minutes Z publishes $F(z_i)$, opens $BC(z_i)$ and terminates.

7. Informal security analysis

Let us assume that the authority that runs the new service conspires with a three-letter agency which can run F up to 5 times faster. The security of the protocol, that is, the freshness of the output $F(z_i)$ depends on the presence of random external contributions $r_i^A, r_i^B, r_i^C, ...$ submitted by outsiders.

If no random external contribution is submitted, this protocol offers no advantage. Z can just compute all random values a week ahead in time, selectively reveal these values to insiders, and publish them exactly at the right moment. Nothing is gained.

If A is suspicious of Z secretly colluding with B to bias the outcome, what she could do is to provide some random contribution r_i^A to z_i, because then she will be sure that $F(z_i)$ is fresh and random. Even a malicious Z cannot impose any bias on $F(z_i)$ because $F(x)$ takes at least one minute to compute, and the bit commitment binds Z to z_i. A's timing is essential here: she must submit r_i^A less then one minute before the computation of $F(z_i)$ starts.

Note that, even though generating 4096 random bits for A is easy, the system would already be secure if r_i^A had an entropy of about a 100 bits. Because, for Z's attack to work it would have to compute $F(H(..., s_i))$ for 2^{100} different values s_i, and this is clearly infeasible.

In fact, the freshness propagates to $F(z_{i+1})$ too. In case Z disposes of a supercomputer, he can compute the intermediate value v_i^1 in $60/5 = 12$ seconds. Now, even supposing that no new contributions $r_{i+1}^A, r_{i+1}^B, r_{i+1}^C ...$ are submitted in the last 48 seconds before dispatching the computation of $F(z_{i+1})$, still Z will need at least a full 60 seconds to compute $F(z_{i+1})$, leading to a total of 72 seconds. But he is forced to dispatch the computation of $F(z_{i+1})$ after 60 seconds, and a 12 second delay will certainly be noticeable, even on the internet. The freshness does not propagate to $F(z_{i+2})$. A malicious Z can compute v_i^2 in 24 seconds, leaving it $60 + 36 = 96$ seconds to compute a suitable $z_{i+2} = H(..., s_{i+2})$, again assuming no fresh contributions $r_{i+2}^A, r_{i+2}^B, r_{i+2}^C ....$

For an outsider B, who did not contribute to z_i, the situation is more complicated. He can only be sure that Z is honest if verifiably fresh randomness has been submitted. In fact, what we see is that in this adversarial scenario, the freshness of the results $F(z_i)$ hinges on the freshness and independence of the external contributions $r_{i+1}^A, r_{i+1}^B, r_{i+1}^C ....$

We believe that this can be made practically feasible, as follows. It would be sufficient if neutral entities such as INMETRO, universities or other organizations (like

 © 2017 Sociedade Brasileira de Computação

the Electronic Frontier Foundation) submit every minute 2048 random bits to Z for B to believe the bits were really random. This belief would even be retro-actively, i.e. applying to bits generated in the past.

It would be even better if some of these trusted entities would run their own version of the beacon Z, cross-feeding each other with fresh randomness by broadcasting there outputs to all the others to be included to the seed for the next input.

8. Conclusion

This paper proposes a novel protocol which offers a randomness beacon on the internet which provides transparency and verifiability, even to persons not physically present to witness the generation. The idea of receiving public contributions to influence random number generation allows any person to verify that the beacon outputs are indeed fresh, and prevents insiders from unduly biasing these outputs. By pipelining parallel instances of the beacon we achieve the desired throughput, publishing fresh randomness every minute. Any person can audit the generator for correctness by verifying that the output published indeed is obtained from the initial seed.

References

BBC (1999). Italy hit by lottery scandal. `http://news.bbc.co.uk/1/hi/world/europe/256206.stm`.

Biryukov, A., Dinu, D., and Khovratovich, D. (2016). Argon2: New generation of memory-hard functions for password hashing and other applications. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 292–302.

Blum, M. (1983). Coin Flipping by Telephone a Protocol for Solving Impossible Problems. *SIGACT News*, 15(1):23–27.

Fischer, M. J., Iorga, M., and Peralta, R. (2011). A public randomness service. In *SECRYPT 2011*, pages 434–438.

Lenstra, A. K. and Wesolowski, B. (2015). A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive*, 2015:366.

NIST (2016). Truly random numbers – but not by chance. `https://www.nist.gov/news-events/news/2012/10/truly-random-numbers-not-chance`.

Rabin, M. O. (1983). Transaction protection by beacons. *J. Comput. Syst. Sci.*, 27(2):256–267.

Simmons, D. (2015). Us lottery security boss charged with fixing draw. `http://www.bbc.com/news/technology-32301117`.

Syta, E., Jovanovic, P., Kokoris-Kogias, E., Gailly, N., Gasser, L., Khoffi, I., Fischer, M. J., and Ford, B. (2017). Scalable bias-resistant distributed randomness. In *IEEE Symposium on Security and Privacy*, pages 444–460.

Togyer, J. (1999). Putting in the fix. `http://www.tubecityonline.com/history/perry.html`.

© 2017 Sociedade Brasileira de Computação

Robustness Testing of CoAP Server-side Implementations through Black-box Fuzzing Techniques

Bruno da S. Melo[1], **Paulo Lício de Geus**[1]

[1]Instituto de Computação (IC) – Universidade Estadual de Campinas (Unicamp)
Av. Albert Einstein, 1251 – CEP 13083-970 – Campinas – SP – Brazil

{brunom,paulo}@lasca.ic.unicamp.br

Abstract. *This paper presents the current status of our research on the robustness of CoAP server-side implementations. We discuss the importance of the CoAP protocol as an enabler of the Internet of Things (IoT) vision, and also the current state of CoAP implementations available out there. Then, we proceed to test those implementations using fuzzing techniques previously used in the literature in areas such as Web Service and Network Protocol security testing, namely Random, Mutational and Generational Fuzzing, both "dumb" and "smart". Finally, we provide preliminary results and analysis regarding i) how robust the CoAP implementations studied are and ii) how the different fuzzing techniques used compare to each other.*

1. Introduction and Motivation

The promise of the Internet of Things (IoT) is to increase the efficiency of our lives by providing new, valued-added services through the integration of several technologies and communications solutions, so our society could benefit from a wide range of applications in areas such as agriculture, manufacturing, city infrastructure (e.g. mobility & transportation, energy & water distribution, environment monitoring), retail, logistics, healthcare, home & building, and many others [1]. This vision leads to a very heterogeneous environment, in terms of both hardware and software [Atzori et al. 2010]. Also, considering the complex, heterogeneous scenarios IoT networks may lead us to, there are security and privacy concerns that need to be addressed before this new paradigm can be widely accepted by the public [Granjal et al. 2015].

Even so, the amount of connected devices is already growing, with current estimates ranging from 20.8 to 30.7 billions for 2020 [2]. Regardless of the exact figure and despite the fact that the "*real* IoT" vision—with uniquely identifiable edge nodes running IPv6, reachable through a Service-Oriented or Resource-Oriented Architecture (ROA or SOA)—is not realized yet, we can find reportings of vulnerable devices in the "ad-hoc, *fake* IoT"—with devices mostly running HTTP over IPv4 behind NAT or proprietary stacks—both found in a lab [Patton et al. 2014] as well as in the wild [3] [4], the biggest and most famous one probably being the Mirai botnet [5].

[1]http://www.libelium.com/libelium-smart-world-infographic-smart-cities-internet-of-things/

[2]http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated

[3]http://www.crn.com/slide-shows/internet-of-things/300084663/8-ddos-attacks-that-made-enterprises-rethink-iot-security.htm

[4]http://www.checkpoint.com/press/2014/media-alert-check-point-researchers-discover-isp-vulnerabilities-hackers-use-take-millions-consumer-internet-wi-fi-devices/

[5]https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/

© 2017 Sociedade Brasileira de Computação

Although a number of standards for IoT communications have been proposed in the recent years, and none of them have become a *de facto* standard yet, the protocol stack composed by IPv6 over Low-Power Wireless Personal Area Networks (**6LoWPAN**s), IPv6 Routing Protocol for Low-Power and Lossy Networks (**RPL**), and the Constrained Application Protocol (**CoAP**) [Shelby et al. 2014] running over **IEEE 802.15.4** has been thoroughly explored by the research community, due to its focus on enabling direct end-to-end integration of edge devices—and initially isolated Wireless Sensor Networks (WSNs)—to the Internet [Palattella et al. 2013, Granjal et al. 2015]. A parallel between this protocol stack and the current Internet stack can be seen in Figure 1.

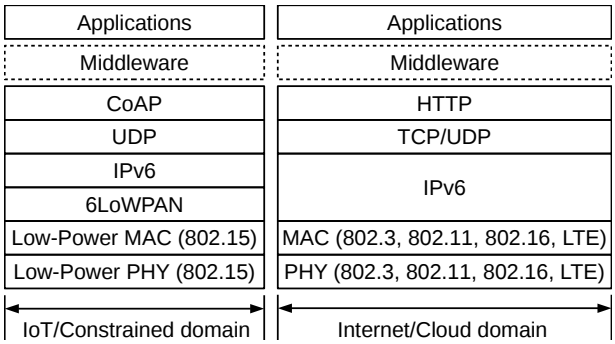


Figure 1. 6LoWPAN-based stack for IoT in comparison with the traditional protocol stack used in the Internet.

Running at the application layer, CoAP is a RESTful protocol featuring a well-defined mapping to HTTP and its methods (`GET`, `POST`, `PUT` and `DELETE`), and supporting multicast, Universal Resource Identifiers (URIs), content-type identification, resource discovery, response codes and simple subscription and caching mechanisms. Aiming for simplicity and low overhead, it requires a fixed 4-byte header. Since the messages are transported over UDP, CoAP implements simple reliability and message deduplication mechanisms as well. Additionally to the base specification, several extensions to the protocol were already standardized as well, the most popular ones being Observe [6], Block [7] and CoRE Link-Formats [8].

As IoT devices are exposed to the Internet to allow resource sharing and ubiquitous services provisioning, the CoAP protocol becomes a possible attack vector, which may be exploited by a myriad of vulnerabilities, including protocol parsing, URI processing, proxying and caching, risk of amplification, IP address spoofing, etc. System-wide, these threats include (D)DoS, corrupted nodes, fraudulent packet injection, and the possibility of a malicious entity compromising a node to perform lateral movements on a private network [Mohsen Nia and Jha 2016].

To the best of our knowledge, there is little to no work on the practical security of this "*real* IoT" vision, and most of them focus either on cryptographic approaches [Raza et al. 2013], or attack scenarios performed by previously compromised CoAP nodes in a constrained network [Martins et al. 2016]. Our work, on the other hand,

[6]RFC7641 - Observing Resources in the Constrained Application Protocol (CoAP)
[7]RFC7959 - Block-wise Transfers in the Constrained Application Protocol (CoAP)
[8]RFC6690 - Constrained RESTful Environments (CoRE) Link Format

© 2017 Sociedade Brasileira de Computação

focus on understanding and finding out the ways in which a CoAP node could be compromised in the first place. To that end, we leverage fuzzing techniques to discover vulnerabilities in available CoAP implementations.

2. Background and Related Work

As previously stated, we found no work on CoAP Robustness or Security Testing. There are, however, a few projects targeting the functional test of CoAP implementations, which we use as a basis for our testing system, especially as an input for the Mutation-based Fuzzer (see PCAP Conversation in Figure 2).

The European Telecommunications Standards Institute (**ETSI**) promotes Plugtest events, in which different organizations can test their own implementations of a given standard, focusing on the interoperability of products and services. Together with other industry members, ETSI has organized four IoT CoAP Plugtests so far, the latest one in March, 2014 [9]. The **F-Interop** project aims to develop and provide online testing tools to perform Conformance, Interoperability, and other non-functional testing, although it does not cover Robustness nor Security testing. A video demonstration of a Proof-of-Concept (PoC) CoAP interoperability testing tool is currently available at the project website [10]. The test specifications used are the ones from ETSI CoAP#4 Plugtest. We note this is an ongoing research project, so further development can be expected. **Peach Fuzzer** is a commercial fuzzer with CoAP support [11]. The tool has an open source Community Version available as well, but CoAP is not supported in that distribution. Codenomicon's **Defensics** is another commercial fuzzer with support for CoAP [12]. This tool has no open source or evaluation version. Table 1 shows a comparison between these testing solutions.

Table 1. Comparison of available CoAP Testing Solutions.

Project	CoRE Reference Specification				Tool Support / Automated Testing	Security Testing
	Base	Observe	Block	Link-Format		
ETSI Plugtests	RFC7252	draft-12	draft-14	RFC6690	- Offline pcap analyzer - Automatic stimuli	-
F-Interop	RFC7252	draft-12	draft-14	RFC6690	- Real-time pcap analyzer - Manual stimuli	-
PEACH Fuzzer	RFC7252	draft-16	draft-17	-	- Fully automated	Mutational Fuzzing
Defensics	RFC7252	-	-	-	- Fully automated	Generational Fuzzing

Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. Regarding fuzzing techniques in general, [Oehlert 2005] describes two ways of obtaining testing data for fuzzing—data generation and data mutation—; differentiates between intelligent fuzzers (those that leverage some knowledge of the target format) and unintelligent fuzzers (those that, for instance, just randomly changes bits); discuss common fuzzing problems, such as the care to be taken regarding target formats using hashes or

[9] http://www.etsi.org/news-events/events/741-plugtests-2014-coap4
[10] http://www.f-interop.eu/index.php/tools-experiments
[11] http://www.peachfuzzer.com/wp-content/uploads/CoAP.pdf
[12] http://www.codenomicon.com/products/defensics/datasheets/coap-server.html

 © 2017 Sociedade Brasileira de Computação

checksums; and the difficulties to check if a target application behaves correctly or not—suggesting the use of source code instrumentation and monitoring parameters from the System Under Test (SUT) execution, such as memory and CPU usage. The present study will provide a comparison between those techniques.

Finally, our efforts are guided by work on fuzzing techniques previously applied in Web Service Security testing, such as data perturbation for generating test cases, by [Offutt and Xu 2004], guidelines to build penetration testing tools by [Antunes and Vieira 2016] and work on using Boundary Value Analysis and dictionaries of peculiar strings to test Network Protocols as done by the PROTOS project [13].

3. Experimental Setup

The architecture of our testing system can be seen in Figure 2. Fuzzing engines (responsible for generating fuzzed data or scrambling—mutating—valid data into fuzzed data) and the workload executor (responsible for sending and receiving packets) were based on the Scapy [14] framework. The steps of an experiment instance are as follows:

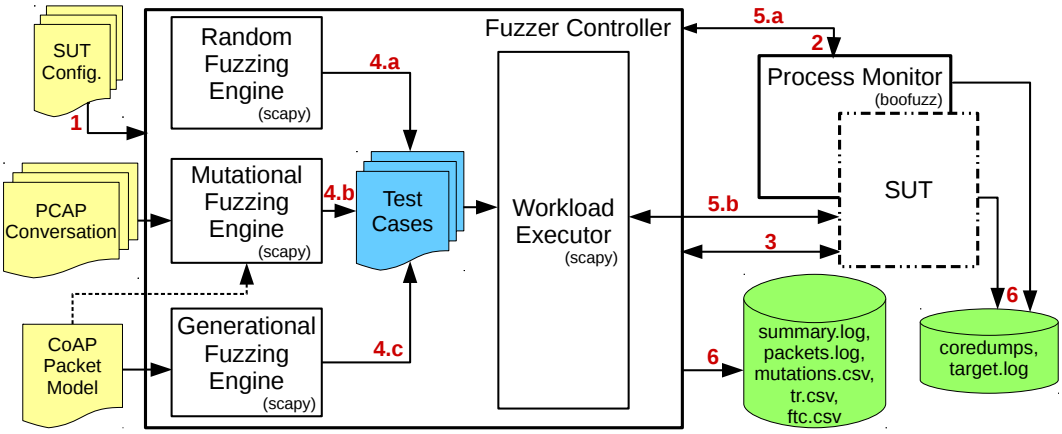


Figure 2. Architecture of the testing system.

1 We fill a configuration file for the given System Under Test (SUT), holding an unique name, the command line to start and stop it, any required environment variables it may need, and parameters regarding how long we should wait for the SUT to (re)start, as well as a target path to be used as heartbeat. Then, we can start our Fuzzer.

2 Our Fuzzer communicates with the Process Monitor [15] (which runs at the target machine—which, in turn, could be the same as the fuzzer machine—listening on a given TCP port), through Remote Procedure Calls (RPC), sending the parameters from the SUT configuration file to it, so the Process Monitor is able to properly start the SUT.

3 Now that the SUT is already running, the first step of our Fuzzer is to send a `GET .well-known/core` CoAP request to the SUT, in order to obtain a list of available links (or paths) the SUT exposes, as well as other relevant information about those links, e.g. if a given link supports the observe feature; the resource type it returns; etc.

[13] https://www.ee.oulu.fi/research/ouspg/Protos

[14] https://github.com/secdev/scapy

[15] This entity was adapted from the Boofuzz tool, available at https://github.com/jtpereyda/boofuzz

© 2017 Sociedade Brasileira de Computação

4.a If we are experimenting with **Random Fuzzer**, a set of completely random Test Cases (TCs) is generated. We also run experiments with a slightly modified version that we call **Informed Random Fuzzer**, which adds `Uri-Path` options—obtained from **step 3**—to the TCs generated, providing some guidance to the testing process.

4.b Else, if we are running a **Mutational Fuzzer**, a set of TCs is generated from mutations performed on previously captured (through `tcpdump` or Wireshark) CoAP conversations. We also run experiments with a slightly modified version that we call **Smart Mutational Fuzzer**, which uses information from the CoAP Packet Model to perform smarter and/or more complex mutations, as well as the target links from **step 3**. Complex mutations requiring knowledge of the CoAP Packet Model includes option duplication, option removal or any mutation to specific protocol fields.

4.c Otherwise, if we are running a **Generational Fuzzer**, we use techniques such as Boundary Value Analysis, Equivalence Partitioning and dictionaries of carefully-crafted strings to fill out CoAP packet templates based on the CoAP Packet Model, generating smarter TCs to be sent to the SUT. In this mode, we are free to explore any possible CoAP packet, given the inherent knowledge of protocol structure.

5 Workload Execution, sending the generated Test Cases to the SUT. Before actually sending it (**5.b**), the Fuzzer informs the Process Monitor of a new incoming TC using the `pre_send()` method through RPC (**5.a**). After the fuzzed packet is sent, and a possible response packet is received (**5.b**), the Fuzzer performs some checks. If the *packet is not answered*, it is appended to a list, together with its TC number; if this list reaches a threshold (we currently use 5), the Fuzzer will check the SUT's health both through an RPC polling method (`post_send()`, which will contact the Process Monitor to check for process's signals or coredump files related to the SUT), as well as through a heartbeat, which is basically a request known to always work on that SUT (the most common one is just a `GET .well-known/core` request). If the fuzzer establishes that the SUT has crashed, relevant information (coredump file, exit status and signal, last five packets sent, etc.) is saved (see **step 6**) and the SUT is restarted, so the process can continue from **step 5**; else, it just continues from **step 5** again. If, however, the *packet is answered*, the Fuzzer can just continue from **step 5**.

6 Relevant information is always saved throughout the fuzzing process. This includes a summary and a report for each packet template (Generational Fuzzer), mutated value per TC, execution times, etc. Especially useful information relevant to the offline crash analysis described in Section 5 includes the SUT log and coredump files. Additionally, the list of "last sent packets" related to each failed TC can be used to reproduce the failures in a more controlled way, assisting in manual debug for root cause analysis.

4. Target Implementations

Since CoAP is a rather new protocol, we have faced some difficulties trying to find relevant samples to be used as SUTs. We ran an extensive and systematic search against public source code and/or software repositories as well as general search engines and other sources, from which results can be seen on Table 2, together with the number of SUT candidates found and their current status regarding availability to our research. The definition for each status is described in the following paragraphs. We note that these numbers are roughly preliminary, since this study about the current use, status and availability of CoAP implementations is expected to constitute one of our research's contributions.

 © 2017 Sociedade Brasileira de Computação

Table 2. Number of SUT candidates found per Code/Software Repository and/or Search Engines. Only the keyword `'coap'` applies, unless noted.

Repository/Search Engine	SUT Candidates				
	Irrelevant	Hard	Pending	OK	Total
https://github.com[a]	1099	33	122	16	1270
https://bitbucket.org	16	2			18
https://gitlab.com	61	1	3		65
http://coap.technology	24		2		26
https://en.wikipedia.org/wiki/Constrained_Application_Protocol	26		2		28
https://code.google.com/archive	6		2		8
https://sourceforge.net	6	2			8
https://www.openhub.net	13				13
http://www.krugle.org	2				2
https://www.codeplex.com	6				6
http://www.grepcode.com	8				8
https://google.com[b]		2	3		N/A

[a] Includes keywords `'lwm2m'` and `'om2m'` (two Device Management protocols based on CoAP).
[b] Mostly ongoing. For each of the relevant keywords, we intend to check the first 5 pages, i.e. 50 results.

After we find an SUT candidate, we perform a preliminary manual inspection of any available source code and documentation to assess the suitability of that candidate towards our research goals. This assessment includes, but is not limited to i) look for any executable which will be listening on an UDP port expecting CoAP packets (including example applications that ships together with programming libraries for CoAP, demo applications ranging from personal projects to industry or academia prototypes, etc.); ii) check if we have available infrastructure to run such an executable (hardware architecture, supported operating system, etc.); and iii) build, install and run the candidate to test for basic CoAP communication.

Table 3. Short list of SUT candidates.

SUT Candidate	Status	CoAP Library	Lang.	SUT Candidate	Status	CoAP Library	Lang.
aiocoap-rd	OK	aiocoap	Python	mbed-client	Hard	mbed SDK	C++
californium-pt	OK	californium	Java	microcoap-server	OK	microcoap	C
californium-proxy	OK	californium	Java	molex-transcend-gw[a]	Hard	?	?
go-coap-server	OK	go-coap	Go	riot-gcoap-server	OK	gcoap + nanocoap	C
ikea-tradfri-gw[a]	Hard	?	?	rtos-wot-server	Hard	Erbium	C
iotivity-fridge-server	Pending	libcoap	C, C++	swiftcoap-server	Hard	SwiftCoAP	Objective-C

[a] As far as we know, these are the only two commercial products using CoAP out there. IKEA Tradfri[16]is a smart lighting + gateway ecosystem and Molex Transcend[17]is a Power over Ethernet (PoE) gateway.

From that assessment, the SUT candidate is classified into one of the following four statuses, due to—but not limited to—the following reasons: a) Irrelevant - duplicate or search result already found in a previous repository, very minimal (or old) support to CoAP RFCs, no actual executable to listen for packets; b) Hard - missing infrastructure, not possible to follow documentation in order to run it due to a language barrier; c) Pending - looks like a good candidate, only pending the build-install-run step, which can take some time since we may have to deal with different programming languages, building

[16]http://www.ikea.com/gb/en/products/lighting/smart-lighting/
[17]http://www.transcendled.com/webfoo/wp-content/uploads/AI-POE-Gateway.pdf

© 2017 Sociedade Brasileira de Computação

platforms or other unpredicted adversity; and d) OK - we are able to run it, and the SUT is not a candidate anymore, being ready to be tested. Table 3 shows a few of our relevant SUT candidates so far, including their statuses, CoAP libraries they use and programming languages they are made in. The application environment diversity should be noted, as it is one of our biggest obstacles.

5. Preliminary Experimental Results

In order to find details about the crashes reported by the Fuzzer, we need to investigate the output artifacts generated between steps 5 and 6 from the process described in Section 3.

For TC-related and time-related metrics, we have files holding TC/sec, TC/crash and execution times, to name a few, which can be drilled down by packet template (Generational Fuzzer only) or CoAP option name, supporting different possible types of analysis, e.g. which options cause more crashes throughout all SUTs, etc. We did not perform this kind of analysis yet.

For crash-related information, we can use both the logfile produced by the SUT itself, which is properly marked as part of the process described in Section 3, so we can relate a given piece of output to a known TC number; as well as the coredump files generated. This task is currently automated, but there is still room for improvements, as it is very dependent on the SUT itself, since our parser for the SUT logfile has to deal with information provided by the specific SUT or specific SUT's programming language, for instance. Nevertheless, we show preliminary crash-related results in Table 4.

Table 4. Experimental results for some SUTs.

SUT	Generational Fuzzing						Random Fuzzing[a]				Informed Random Fuzzing[a]			
	Target Paths	Gen. TCs	% TC Exec.	Exec. Time	Crashes Total	Crashes Uniq.	% TC Exec.	Exec. Time[b]	Crashes Total	Crashes Uniq.	% TC Exec.	Exec. Time[b]	Crashes Total	Crashes Uniq.
californium-pt	29	3086634	100%	90h	9	0	100%	30	0	0	100%	34	0	0
coapthon-server	13	1255058	20,6%	22h	20725	7	5,2%	12	100	4	83,7%	35	50	3
ibm-coap-proxy	1	103442	95,2%	3h	209	1	56%	16	100	1	8,1%	6	200	1
jcoap-pt	4	385262	46,4%	7h	4217	14	100%	29	3	1	100%	41	50	3
libcoap-server	3	299746	90,8%	4h30	693	1	100%	30	0	0	100%	33	0	0
libnyoci-pt	7	670358	99,7%	12h30	308	6	100%	31	0	0	100%	37	0	0

[a] 40000 TCs were generated for Random and Informed Random Fuzzing.
[b] Execution Time in **minutes** for Random and Informed Random Fuzzing.

We can see from Table 4 that, although the number of generated TCs is bigger for Generational Fuzzing than for both types of Random Fuzzing—it is proportional to the number of packet templates used, which in turn is proportional to the number of paths exposed by the SUT—, incurring in larger execution times, this technique is also able to detect more crashes, for most cases. We also observe the number of recurrent crashes. We are currently using a simple heuristics in which if a given parameter, such as an specific packet template or CoAP option causes a threshold number of crashes, we skip to the next template/option, trying not to lose much time on potential duplicated crashes; this is what causes the round numbers in total crashes for both types of Random Fuzzing.

6. Final Considerations

Preliminary results shows the ability of fuzzing techniques in uncovering crashes during CoAP servers' execution and testing, which in turn represents potential security vulnerabilities in those servers (observing that a crash already threatens the availability property

 © 2017 Sociedade Brasileira de Computação

of security anyway). It also shows that a more informed, elaborate technique, such as generational fuzzing, is able to expose crashes not exposed by random fuzzing. Even so, our research currently have limitations regarding the relevance and number of our samples (the SUTs), due to no widespread use of CoAP so far. Additionally, although the Mutational Fuzzing Engine module is not ready yet, it is a work in progress, and the efficiency of this technique will be compared to the ones presented here in Section 5 as well. Finally, we expect to obtain more relevant and detailed results, such as performing root cause analysis for discovered crashes, as well as to apply these results, as our research progresses.

References

Antunes, N. and Vieira, M. (2016). Designing vulnerability testing tools for web services: approach, components, and tools. *International Journal of Information Security*, pages 1–23.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15):2787–2805.

Granjal, J., Monteiro, E., and Sa Silva, J. (2015). Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials*, 17(3):1294–1312.

Martins, R. d. J., Schaurich, V. G., Knob, L. A. D., Wickboldt, J. A., Filho, A. S., Granville, L. Z., and Pias, M. (2016). Performance analysis of 6lowpan and coap for secure communications in smart homes. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 1027–1034.

Mohsen Nia, A. and Jha, N. K. (2016). A comprehensive study of security of internet-of-things. *IEEE Transactions on Emerging Topics in Computing*, 6750(c):1–1.

Oehlert, P. (2005). Violating assumptions with fuzzing. *IEEE Security and Privacy Magazine*, 3(2):58–62.

Offutt, J. and Xu, W. (2004). Generating test cases for web services using data perturbation. *ACM SIGSOFT Software Engineering Notes*, 29(5):1.

Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., and Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. *IEEE Communications Surveys & Tutorials*, 15(3):1389–1406.

Patton, M., Gross, E., Chinn, R., Forbis, S., Walker, L., and Chen, H. (2014). Uninvited connections: A study of vulnerable devices on the internet of things (iot). *2014 IEEE Joint Intelligence and Security Informatics Conference*, pages 232–235.

Raza, S., Shafagh, H., Hewage, K., Hummen, R., and Voigt, T. (2013). Lithe: Lightweight secure coap for the internet of things. *IEEE Sensors Journal*, 13(10):3711–3720.

Shelby, Z., Hartke, K., and Bormann, C. (2014). Rfc7252 - the constrained application protocol (coap).

© 2017 Sociedade Brasileira de Computação

Detecção e autorreparo de anomalias em redes definidas por software*

Fernando Luiz Moro, Alexandre Amaral, Ana Paula Amaral, Rodrigo Nogueira

Instituto Federal de Educação, Ciência e Tecnologia Catarinense – Campus Camboriú
Caixa Postal 2016 – 88.340-055 – Camboriú – SC – Brasil

fernandoluizmoro@gmail.com,
{alexandre.amaral,ana.amaral,rodrigo.nogueira}@ifc.edu.br

Abstract. *The accelerated technological advances have resulted in the increased occurrence of network anomalies, such as DoS attacks. The complexity and heterogeneity of the current networks has been the main obstacle to detect and especially block the attacks without human intervention. Thus, this work presents a solution capable of detecting and blocking anomalies automatically in the context of software defined networks (SDN). The proposed solution consists of three main modules, responsible for collecting of data, detecting anomalies and applying a corrective action to solve the identified problem. To validate the solution a case study was performed using a real network attack.*

Resumo. *Os avanços tecnológicos acelerados resultaram no aumento da ocorrência de anomalias de redes, como os ataques DoS. A complexidade e heterogeneidade das atuais redes têm sido os principais óbices para a realização da tarefa de detectar e principalmente bloquear os ataques sem a intervenção humana. Assim, neste trabalho é apresentada uma solução capaz de detectar e bloquear anomalias de forma automática no contexto das redes definidas por software (SDN). A solução proposta é constituída de três módulos principais, responsáveis pela coleta dos dados, detecção de anomalias e aplicação de uma ação corretiva para solucionar o problema identificado. Para validar a solução, um estudo de caso foi realizado utilizando um ataque de rede real.*

1. Introdução

Bilhões de dispositivos como *smartphones*, *notebooks*, *smart tvs*, geladeiras, micro-ondas e celulares estão conectados à Internet, denominada agora de Internet de Todas as Coisas (*IoE – Internet of Everthing*) [Conti *et al.* 2017]. Porém, o número de ataques também tem crescido em larga escala nos últimos anos. Segundo o balanço divulgado pelo Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br) em 2016, houve um aumento de 138% de ataques DoS (*Denial of Service*) originados por equipamentos de Internet das Coisas infectados que faziam parte de botnets [Abranet 2017]. Uma previsão realizada pela Forrester estimou que 500.000 dispositivos de IoT seriam comprometidos em 2017 [Francis 2017]. Uma das principais

*Artigo apoiado pelo IFC - Campus Camboriú, edital nº 007/GDG/IFC-CAM/2017

 © 2017 Sociedade Brasileira de Computação

causas para o aumento dos ataques está nas vulnerabilidades presentes nas atuais tecnologias. Assim, com o objetivo de explorar estes pontos de falhas, cibercriminosos disparam uma série de ataques, causando alterações no comportamento da rede, também conhecidos como anomalias de redes [Amaral *et al.* 2017].

Mecanismos para detectar e bloquear as anomalias de redes se fazem necessários. Todavia, as atuais redes de computadores se tornaram complexas e heterogêneas, contendo dispositivos e *softwares* de inúmeros fabricantes denominados "caixas pretas" por [Costa 2013], ou seja, implementações integradas baseadas em *software* e *hardware* proprietário. Isto traz uma complexidade para detectar e principalmente para aplicar ações automatizadas capazes de bloquear a ocorrência da anomalia na rede.

Neste contexto, surgiram as redes definidas por software (SDN – *Software Definied Network*) que podem ser implementadas através da tecnologia OpenFlow[2]. Dentre as vantagens advindas deste paradigma está a simplicidade para a realização do gerenciamento e a execução das tarefas de detecção e bloqueio dos ataques de rede [Ahmad *et al.* 2015].

Na literatura um dos principais pontos abordados sobre as redes definidas por *software,* no que diz respeito aos aspectos de segurança, tem sido a centralidade do controlador. Embora o controle central seja a principal vantagem do SDN, também é um único ponto de falha e alvo de ataques [Dacier *et. al* 2017]. Em uma rede SDN convencional (controlador único), se um controlador falhar, a disponibilidade da rede pode ser perdida completamente [Yang 2015].

Mediante a problemática encontrada, [Mousavi 2014] apresenta um mecanismo de detecção antecipada de um ataque DDoS (*Distribute Denial of Service*) utilizando entropia. O mecanismo é implementado no controlador de uma rede SDN com o objetivo de ser uma solução leve. A proposta apresentada pelo autor é efetiva, detectando os ataques nos primeiros 250-500 novos pacotes, evitando a perda do controlador. Contudo, a tarefa de bloqueio do ataque não é apresentada pelo autor, recomendo-a como trabalho futuro.

Embora a centralidade do controlador seja enxergada como um ponto de falha, a visão holística da rede oferecida por ele, simplifica a representação dos problemas e decisões, bem como a execução das ações de reparo [Macedo *et al.* 2015]. Com isso, a tarefa de detecção e bloqueio de anomalias em uma SDN é facilitada, sem a necessidade de *hardware* especializado [Porras *et al.* 2015].

Este artigo tem como principal objetivo propor uma solução para detectar e bloquear as anomalias de rede, em uma rede definida por *software.* Diferente dos trabalhos atuais, o enfoque da proposta deste trabalho não está na detecção de ataques realizados especificamente no controlador, e sim, como explorar as características de uma SDN para detectar e principalmente aplicar ações de autorreparo em ataques lançados a qualquer dispositivo na rede. A solução proposta é composta por três módulos principais, *Flow Collector* (FC), *Network Attack Detector* (NAD) e *Action Executor* (AE). O primeiro módulo é responsável pela coleta e exportação dos fluxos IP

[2] http://archive.openflow.org/wp/learnmore/

© 2017 Sociedade Brasileira de Computação

originados dos dispositivos de redes. O segundo módulo faz a detecção de anomalias de redes através da análise dos fluxos coletados e por último, o terceiro módulo possui a função de reparar a ocorrência da anomalia com a mínima intervenção humana.

2. Fundamentação teórica

2.1. Anomalias de rede e fluxos IP

Anomalias de redes são consideradas eventos anormais, como ataques (*e.g.*, DDoS) que podem trazer inúmeros prejuízos à rede [Ahmed *et al.* 2016]. Assim, é imprescindível o desenvolvimento de mecanismos capazes de detectar e bloquear as anomalias de redes de forma automatizada, para que a integridade das informações e dispositivos envolvidos sejam preservados.

Uma das fontes de dados para a detecção de anomalias é o fluxo IP. Em uma rede de computadores, um fluxo IP é gerado pela troca de dados entre dois dispositivos de redes, sendo uma sequência de pacotes unidirecional com propriedades comuns, como endereços de origem e de destino, porta de origem e destino e protocolo observados em um dado intervalo de tempo [Amaral *et al.* 2017]. Através da análise estatística obtidas através dos fluxos é possível detectar os eventos anômalos na rede.

2.2. Redes definidas por software

A proposta de novas soluções de redes para atender a demanda das novas tecnologias é dificultada pela complexidade das redes atuais. Solucionar a ocorrência de anomalia de rede se tornou uma tarefa árdua e improfícua de ser realizado manualmente pelo administrador de rede, devido ao aumento do número de dispositivos na rede e por eles possuírem suas próprias configurações. A necessidade de criar novas soluções para as atuais redes e a falta de homogeneidade levou a criação das redes definidas por *software*.

Este novo paradigma de rede promove um padrão mais simplificado e livre das limitações encontradas nas arquiteturas de redes existentes, possibilitando um melhor controle sobre os recursos da rede [Mousavi 2014]. Atualmente os dispositivos de redes possuem implementado em seu interior o plano de dados e de controle. O plano de dados é responsável pelo encaminhamento de pacotes por meio de protocolos e regras definidas pelo plano de controle. Ambos os planos são exclusivamente definidos pelos fabricantes, dificultando a criação de novos serviços e a gerência da rede. Diferentemente, nas redes SDN há uma separação do plano de dados do plano de controle, resultando em elementos de comutação (*switches* e roteadores) com uma interface simplificada e que podem ser controlados por um elemento externo via *software*. Para a implementação das redes definidas por software foi desenvolvido pela Universidade de Stanford o OpenFlow [Costa 2013 apud McKeown *et al.* 2008].

O OpenFlow é um padrão aberto para o paradigma de redes SDN. Ele consiste de uma interface de programação, que permite ao desenvolvedor controlar diretamente os elementos de encaminhamento de pacotes presentes no dispositivo de rede [Costa 2013]. A estrutura desta nova tecnologia é apresentada na Figura 1.

O controlador é o elemento externo responsável por definir o comportamento da rede através do gerenciamento das tabelas de fluxos. Este componente realiza a

 © 2017 Sociedade Brasileira de Computação

abstração dos elementos de comutação, permitindo através de uma linguagem de programação de alto nível a criação de aplicações para agirem sobre a rede. Todos os switches OpenFlow em uma rede se comunicam através do protocolo OpenFlow com o controlador que age de forma centralizado sobre eles, embora, podendo ser implementado de forma distribuída.

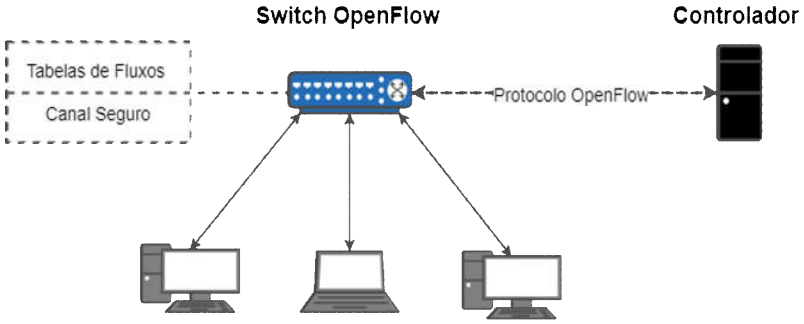


Figura 1. Estrutura OpenFlow [Costa 2013 apud Stanford University].

O *switch* OpenFlow é composto por tabelas de fluxos e um canal seguro. O protocolo OpenFlow não garante a segurança das mensagens, por conta disto, é estabelecido um canal seguro através do protocolo SSL **(***Secure Socket Layer*) e TCP (*Transmission Control Protocol*). As tabelas de fluxos são responsáveis pela verificação de cada pacote trafegado na rede em relação as regras inseridas pelo controlador. Cada regra define como um fluxo de pacotes deve se comportar. Segundo [Costa 2013] uma regra é formada por uma tupla de doze elementos que reúne características dos protocolos da camada de enlace, de rede, e de transporte segundo o modelo TCP/IP. Além disso, toda entrada de fluxo está associada a um conjunto de instruções, contadores, prioridade, intervalo de tempo e *cookies*. Em um *switch* que não possui a tecnologia OpenFlow as regras são definidas apenas pelo endereçamento MAC (*Media Access Control*) e IP (*Internet Protocol*).

Todo pacote é comparado com as entradas de fluxos presentes nas tabelas. Caso haja uma regra compatível com o cabeçalho do pacote, uma ação será aplicada pelo conjunto de instruções. Caso não haja, o pacote é enviado para o controlador determinar o que será realizado. O pacote pode ser enviado como um todo para o controlador ou o *switch* pode armazenar a carga útil e enviar apenas o cabeçalho [Mousavi 2014]. O controlador pode inserir uma nova entrada de fluxo para que o pacote possa prosseguir para o seu destino ou optar por outra ação. Cada entrada em uma tabela pode possuir um intervalo de tempo predeterminado e para cada análise de pacote o campo de contadores é atualizado para a geração de estatísticas dos fluxos. Nota-se, que o controlador é o responsável por determinar o comportamento de cada fluxo trafegado em uma rede definida por *software*. Com isso, ao detectar uma anomalia de rede, pode-se enviar imediatamente uma regra para rejeitar todos os pacotes responsáveis pela anormalidade, evitando o esgotamento dos recursos.

© 2017 Sociedade Brasileira de Computação

3. Implementação

A solução proposta neste trabalho é constituída de três módulos principais representados na Figura 2. O primeiro módulo proposto é o *Flow Collector* que coleta os fluxos IP dos *switches* OpenFlow. A coleta dos fluxos IP é obtida via protocolo NetFlow[3] v.9 e a exportação é realizada para um computador através da ferramenta Softflowd[4]. A captura dos fluxos exportados foi feita através da ferramenta Nfcapd[5]. A ferramenta utilizada para o processamento dos fluxos foi a Nfdump, gerando fluxos em um formato acessível para o módulo *Network Attack Detector*.

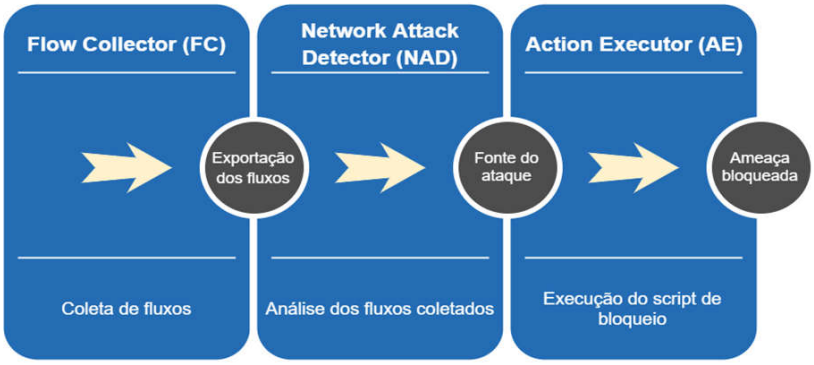


Figura 2. Módulos da solução proposta.

Na etapa seguinte, todos os fluxos IPs coletados são analisados pelo detector desenvolvido por [Amaral *et al.* 2017] com o objetivo de detectar um ataque de rede. Ao detectar um ataque é gerado como entrada para o módulo *Action Executor,* o IP de origem, IP de destino e o tipo de ataque detectado.

O *Action Executor* realiza o bloqueio automático do ataque, utilizando um *shell script* desenvolvido com base no módulo *Static Entry Pusher* presente no controlador (*e.g.*, controlador floodlight). Este módulo é acessado via REST API (*Representational State Transfer Application Programming Interface*) e possui como função principal a consulta, a inserção e a exclusão das entradas de fluxos nas tabelas presentes nos *switches* OpenFlow [Izard 2017].

4. Resultados e discussão

A simulação de uma rede definida por *software* foi realizada utilizando a ferramenta Mininet[6] através de sua interface gráfica Miniedit. Esta aplicação permite a criação e a execução de uma rede SDN de forma simples e rápida. A Figura 3 apresenta a topologia utilizada para a realização dos experimentos. O Mininet utiliza o Open

[3] Tecnologia que permite obter informações sobre o fluxo IP como: data, hora, duração, protocolo, IP de origem, IP de destino, número de pacotes, número de bytes e número de fluxos.
[4] http://www.mindrot.org/projects/softflowd/.
[5] http://nfdump.sourceforge.net/.
[6] http://mininet.org/.

© 2017 Sociedade Brasileira de Computação

vSwitch[7] para a criação dos *switches* virtuais com a tecnologia OpenFlow. Para o controle dos dispositivos de rede foi escolhido o controlador OpenFlow Floodlight[8].

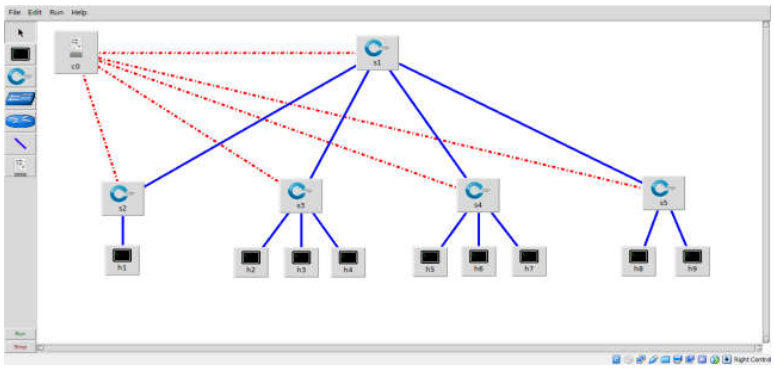


Figura 3. Topologia de rede criada.

A fim de avaliar o funcionamento da solução foi realizado um ataque DoS (*Denial of Service*) do tipo SYN *flooding*, utilizando a ferramenta Hping3[9]. Esta ferramenta foi configurada para enviar `100` pacotes de `64` bytes por segundo para a porta `80` do dispositivo alvo. Esse ataque envolveu dois dispositivos de rede. O primeiro foi o atacante com IP `192.168.0.5` e o segundo o dispositivo alvo com IP `192.168.0.1`.

Todos os fluxos encaminhados pelas máquinas foram exportados corretamente pelo módulo *FC*. Ao ser acionado, o módulo *NAD* detectou o ataque, apresentando uma visão holística da rede e destacando em vermelho os dispositivos envolvidos, como pode ser observado na Figura 4.

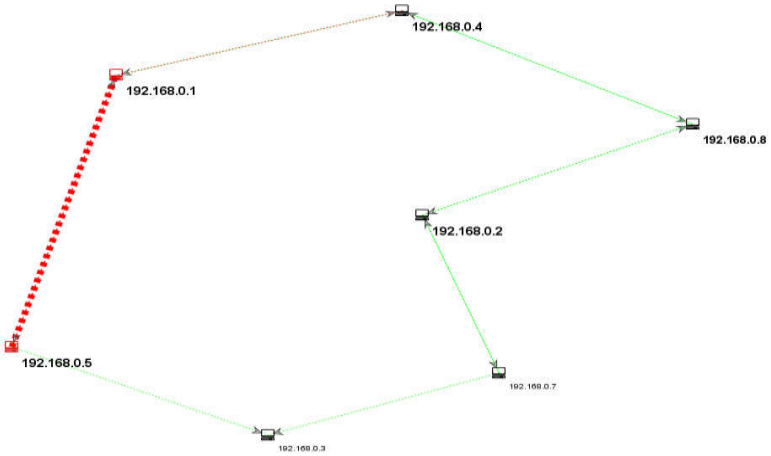


Figura 4. Visão global da rede gerada pelo módulo *NAD*.

[7] http://docs.openvswitch.org/en/latest/.
[8] https://floodlight.atlassian.net/wiki/display/floodlightcontroller/User+Documentation.
[9] http://www.hping.org/.

© 2017 Sociedade Brasileira de Computação

Após a detecção do ataque DoS, o módulo *AE* executou um *shell script* de bloqueio automatizado, conforme apresentado na Figura 5 (a). O bloqueio foi realizado através do envio de uma entrada de fluxo na tabela do *switch* vinculado ao IP atacante. Esta entrada possui a ação de rejeitar todos os pacotes enviados do IP `192.168.0.5` para o IP de destino `192.168.0.1`, impedindo que a continuidade do ataque culminasse no esgotamento dos recursos do dispositivo alvo.

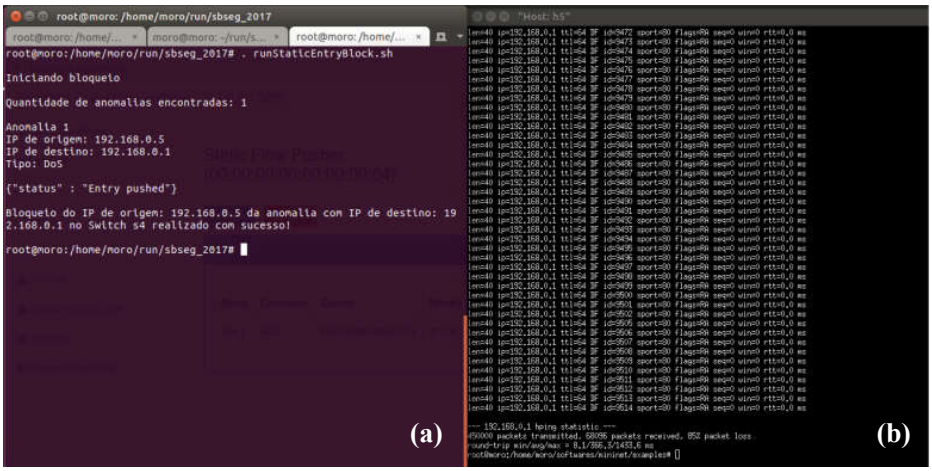


Figura 5. (a) Shell script de bloqueio (b) Ataque DoS bloqueado.

É possível visualizar na Figura 5 (b) que, neste ataque, `450.000` pacotes foram enviados para o alvo. Porém, deste montante, apenas `68.096` pacotes chegaram ao destino, ou seja, `15%`. Um total de `381.904 (85%)` dos pacotes não chegaram ao alvo, mostrando que a solução proposta neste trabalho consegue bloquear o ataque automaticamente, sem a intervenção humana. Os `15%` dos pacotes que obtiveram sucesso ocorreram devido ao tempo demandado na coleta e na exportação dos dados da rede (fluxos IP) para serem analisados pelo detector.

5. Considerações Finais

Este trabalho apresentou uma solução capaz de detectar e aplicar ações de autorreparo de anomalias em uma rede definida por *software*. Através de um estudo de caso inicial utilizando um ataque DoS foi possível validar a solução, destacando a sua capacidade para detectar e interromper a ocorrência da anomalia sem a intervenção humana. A ação de bloqueio executada pelo módulo *AE* mostrou como as ações de reparo automáticas podem ser aplicadas em uma rede SDN utilizando a tecnologia OpenFlow. Com isto, é possível salientar que a segurança e o gerenciamento de rede, neste novo paradigma, são simplificados, reduzindo a complexidade imposta pelas redes atuais em função da heterogeneidade dos equipamentos. Como trabalho futuro outros testes com diferentes ataques serão realizados e a autonomicidade da solução, bem como a sua capacidade de inferir ataques, através do aprendizado de máquina, serão explorados.

 © 2017 Sociedade Brasileira de Computação

Referências

Abranet. (2017) “Internet das Coisas faz ataques DDoS crescerem 138% no Brasil”, http://www.abranet.org.br/Noticias/Internet-das-Coisas-faz-ataques-DDoS-crescerem-138%25-no-Brasil-1531.html?UserActiveTemplate=site#.WabyFciGPIV, Agosto.

Ahmad, I., Namal, S., Ylianttila, M. and Gurtov, A. (2015) “Security in Software Defined Networks: A Survey”, In: IEEE Communications Surveys & Tutorials, vol. 17, no. 4, p. 2317-2346.

Ahmed, M., Mahmood, A. N. and Hu, J. (2016) “A survey of network anomaly detection techniques”, In: Journal of Network and Computer Applications 60, p. 19-31, Austrália.

Amaral A. A. et al. (2017), “Deep IP flow inspection to detect beyond network anomalies”, Computer Communications, vol. 98, p. 80-96.

Conti, M., Dehghantanha, A., Franke, K. and Watson, S. (2017) “Internet of Things Security and Forensics: Challenges and Opportunities”, In: Future Generation Computer Systems, ISSN 0167-739X.

Costa, L. R. (2013) “OpenFlow e o Paradigma de Redes Definidas por Software”, In: Universidade de Brasília, Monografia (Licenciatura em Ciência da Computação), ix, 143 f., Brasília.

Dacier M. C. *et al.* (2017), "Security Challenges and Opportunities of Software-Defined Networking," in IEEE Security & Privacy, vol. 15, no. 2, p. 96-100.

Francis, R. (2017) “Data breaches through wearables put target squarely on IoT in 2017”, http://www.csoonline.com/article/3150881/internet-of-things/data-breaches-through-wearables-put-target-squarely-on-iot-in-2017.html, Agosto.

Izard, R. (2017) “How to add a REST API to a Module”, https://floodlight.atlassian.net/wiki/display/floodlightcontroller/How+to+add+a+REST+API+to+a+Module, Julho.

Macedo, D. F., Guedes, D., Vieira, L. F. M., Vieira, M. A. M. and Nogueira Michele (2015) “Programmable Networks – From Software-Defined Radio to Software-Defined Networking”, In: IEEE Communications Surveys & Tutorials, vol. 17, no. 2, p. 1102-1125.

Mousavi, S. M. (2014) “Early Detection of DDoS Attacks in Software Defined Networks Controller”, In: Thesis (master), Carleton University, Ottawa, Ontario.

Porras, P., Cheung, S., Fong, M., Skinner, K. and Yegneswaran, V. (2015) “Securing the Software-Defined Network Control Layer”, In: Network and Distributed System Security Symposium (NDSS), Sand Diego, California.

Yan, Q., Yu, F. R., Senior member, IEEE, Gong, Q. and Li, J. (2015) “Software-defined networking (SDN) and distributed denial of servisse (DDoS) attacks ind cloud computing environments: a survey, some research issues, and challenges”, In: IEEE Communications Surveys & Tutorials, vol. 18, no. 1, p. 602-622.

© 2017 Sociedade Brasileira de Computação

Análise de segurança da distribuição de raízes na ICP-Brasil

Bruno C. Dias Ribeiro, Edson Floriano S. Junior[1], Diego F. Aranha[2]

[1] Departamento de Ciência da Computação – Universidade de Brasília (UnB)

[2]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)

Abstract. *This work presents a security analysis of the distribution of root certificates in the Brazilian Public Key Infrastructure (ICP-Brasil). A simple proof of concept is provided to replace a root certificate downloaded from the official repository with a fake certificate, compromising the trusted store of the client machine even when the suggested verification procedure is strictly followed. Finally, some recommendations are offered to strengthen the certificate distribution processin hope of contributing to make ICP-Brasil more robust.*

Resumo. *Este trabalho analisa a distribuição de certificados raízes na ICP-Brasil do ponto de vista de segurança. Uma prova de conceito simples é apresentada para substituir o certificado raiz distribuído no repositório oficial por um certificado falso, comprometendo a base de certificados confiados da máquina cliente, mesmo quando o procedimento sugerido de verificação é respeitado. Finalmente, algumas recomendações são apresentadas para fortalecer a distribuição de certificados, na esperança de contribuir para tornar a ICP-Brasil mais robusta.*

1. Introdução

A certificação digital se propõe a resolver o problema de obtenção de chaves públicas autênticas, atuando como uma "terceira parte confiável" que atesta a titularidade de uma chave pública [Kohnfelder 1978]. Este papel é desempenhado pelas Autoridades Certificadoras (AC) e Autoridades de Registro (AR), cuja hierarquia caracteriza uma Infraestrutura de Chaves Públicas (ICP) [Kuhn et al. 2001].

As ACs são portanto elemento chave do modelo e também sua maior fraqueza [AccessNow 2011], já que o comprometimento de uma AC pode implicar comprometimento de toda estrutura abaixo dela. Falhas de segurança em sua operação podem levar a verdadeiros desastres, como o ocorrido no caso DigiNotar em 2011 [Prins 2011], com o comprometimento de várias ACs. Um problema central para a segurança de ICPs é a instalação de certificados raízes de ACs, dado que sua manipulação maliciosa permite ataques posteriores difíceis de detectar. Sob controle da chave privada correspondente a um certificado maliciosamente instalado, um atacante pode emitir certificados reconhecidos como válidos para personificar qualquer domínio na Internet.

Neste trabalho, foi realizada uma análise de segurança do processo de distribuição de certificados raízes na ICP-Brasil, com o objetivo de colaborar com a robustez da infraestrutura contra atividade maliciosa. É apresentada uma prova de conceito que demonstra a substituição do certificado raiz legítimo por um fraudulento, mesmo quando os procedimentos recomendados de verificação são seguidos. Esse problema não é exatamente novo para a comunidade científica brasileira, mas parece ser a primeira vez que aparece discutido de forma sistemática, com recomendações para mitigação e aprimoramento.

 © 2017 Sociedade Brasileira de Computação

2. Trabalhos Relacionados

O caso de comprometimento da DigiNotar [Prins 2011] é um exemplo notório de desastre na área da certificação digital. A empresa teve cerca de 30 ACs de diversos níveis (inclusive raízes) comprometidas, sendo que em pelo menos 6 delas foram encontradas evidências de abuso por parte dos atacantes. Todas faziam parte do mesmo domínio *Active Directory* (AD), com isso, uma vez obtidos privilégios de administrador no domínio, o atacante conseguiu invadir quantas ACs julgou necessário.

Falhas como esta possibilitam ataques "*man-in-the-middle*" (MITM), como demonstrado em [Huang et al. 2014], em que um atacante fabrica certificados para domínios arbitrários de seu interesse, tornando-se capaz de controlar todo o tráfego entre cliente e servidor. Como estratégia de mitigação, os autores propõem um método para detectar certificados falsos através do uso de um pequeno *plugin* nos navegadores de clientes. Através da análise de características como tamanho da cadeia de certificados, tamanho das chaves, domínio no campo *Common Name* e emissor do certificado, o *applet* conclui, com grande probabilidade de acerto, se um certificado é verdadeiro ou não.

Estes problemas colocam em evidência a necessidade de um rápido e eficiente tratamento de incidentes nas ICPs. Como mostrado na literatura [Cooper 1998, Li and Feigenbaum 2001], os ataques podem inclusive atuar na tentativa de burlar o processo de revogação de certificados. Se o processo de revogação não for feito de maneira eficiente, o atacante pode tanto impedir que o certificado seja revogado, quanto continuar a se passar pela vítima mesmo após a revogação do mesmo. O modelo de confiança em ACs também é questionado na literatura [Leavitt 2011, Roosa and Schultze 2013]. Isto porque qualquer AC dita "confiável" pode emitir um certificado atestando a titularidade de uma determinada chave pública e este será automaticamente aceito. Com isso, muitas vezes a verificação de identidade efetuada pelas autoridades de registro é a única defesa contra a obtenção de certificados fraudulentos.

Cientes do risco, navegadores modernos estabelecem níveis rigorosos de segurança e transparência para admitir certificados raízes em sua cadeia de certificados confiados. Por exemplo, a Mozilla instituiu o *CA Certificate Program*, um processo rigoroso de avaliação que determina implantação de boas práticas e requer auditorias independentes regulares [WebTrust 2011, WebTrust 2017]. Os requisitos costumam ser ainda mais rigorosos para ICPs governamentais, dado o interesse evidente de alguns governos em personificar serviços para promover censura e vigilância [Mozilla 2015]. Além da admissão, a operação de autoridades certificadoras é cuidadosamente acompanhada por esforços como o *Certificate Transparency* [Laurie et al. 2013], que mantém um registro temporal de certificados emitidos e observados, já utilizado anteriormente para detectar emissão de certificados fraudulentos com sucesso [Kerner 2015].

3. ICP-Brasil

A Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil) é responsável por garantir autenticidade, integridade e validade jurídica aos documentos eletrônicos com certificação digital no Brasil[1]. A ICP-Brasil conta atualmente com 14 ACs de primeiro nível credenciadas, responsáveis pela emissão, distribuição, gerenciamento e revogação de certificados

[1] `https://www.iti.gov.br/icp-brasil`

© 2017 Sociedade Brasileira de Computação

para ACs de próximo nível ou diretamente ao usuário final. Embora a ICP-Brasil seja instituída de modo governamental, tanto entidades públicas quanto privadas podem requerer credenciamento para atuarem como ACs e ARs. Desta forma, é possível a atuação de empresas no ramo da certificação digital como parte integrante da hierarquia, comercializando certificados digitais a pessoas físicas ou jurídicas.

Representada pelo Instituto Nacional de Tecnologia da Informação (ITI), a ICP-Brasil já emitiu um total de 6 certificados auto-assinados (raízes). Com validade inicial de 10 anos, a primeira raiz foi emitida em 2001 utilizando o algoritmo RSA com chaves de 2048 *bits* e função de resumo SHA-1. Os parâmetros foram mantidos na segunda raiz (v1), quando o prazo de validade foi estendido para 13 anos. A terceira raiz (v2) foi emitida em 2010 e também utilizou RSA com o dobro de tamanho de chaves, mas adotou a função de resumo SHA-512, mantida desde então. A quarta raiz (v3) deu início à utilização de assinaturas ECDSA sobre curvas elípticas, mas terminou revogada após suspeitas sobre as curvas padronizadas pelo *National Institute of Standards and Technology* (NIST) [Bernstein et al. 2015]. Raízes posteriores utilizaram curvas Brainpool [Merkle and Lochter 2010] (v4) e RSA com 4096 *bits* (v5).

4. Distribuição de certificados

Um problema desafiador para qualquer ICP é promover a distribuição de suas raízes, que se torna ainda mais complexo no caso de infraestruturas governamentais, ao ser reconhecido o risco adicional [Mozilla 2015]. Após negociação com a empresa Microsoft, o ITI conseguiu introduzir a raiz v2 do certificado raiz em versões do Windows a partir do XP[2]. Aparentemente, não houve o mesmo sucesso para raízes subsequentes, que continuam ausentes no sistema operacional, mesmo havendo certificados já emitidos para serviços populares sob essas raízes. Um exemplo de serviço é o Expresso[3], utilizado para comunicação institucional em órgãos do governo federal. Além disso, sistemas operacionais alternativos, como GNU/Linux ou *Android*, ou ainda navegadores que não confiam incondicionalmente na base de certificados fornecida pelo sistema operacional, como o *Mozilla Firefox*, continuam apresentando alerta de segurança indicando a ausência do certificado raiz para encerrar a verificação de um certificado final.

Para resolver esse problema, a ICP-Brasil utiliza o procedimento de instalação manual, documentado em páginas específicas para cada navegador[4]. Há severas limitações nessa abordagem. Em primeiro lugar, o repositório raiz de certificados é fornecido utilizando o protocolo HTTP[5]. Apesar das informações lá contidas serem de caráter público, não exigindo a propriedade de confidencialidade fornecida pelo transporte sobre SSL/TLS, perde-se também a propriedade vital de autenticação. Para esse novo problema, o ITI serve páginas com valores de *hash* para verificação de integridade dos pacotes de certificado, também servidos via HTTP[6]; ou HTTPS sob certificado da própria ICP-Brasil, reduzindo o problema a si próprio. Não sendo possível a autenticação da página que oferece os certificados para instalação manual, o usuário não consegue determinar se o con-

[2] https://www.microsoft.com/brasil/setorpublico/temas/icp-brasil.mspx
[3] https://expressobr.serpro.gov.br/
[4] http://www.iti.gov.br/navegadores
[5] http://acraiz.icpbrasil.gov.br
[6] http://iti.gov.br/repositorio/84-repositorio/489-certificados-das-acs-da-icp-brasil-arquivo-unico-compactado

 © 2017 Sociedade Brasileira de Computação

junto correto de certificados foi obtido e será instalado, até porque tanto os certificados quanto informações para verificação de integridade podem ser sutilmente substituídos por um adversário com controle da rede, sem produzir qualquer alerta visível. A prova de conceito descrita na Seção 5 apresenta em detalhes um ataque dessa natureza. Não obstante, por vezes os valores de *hash* publicados pelo ITI estavam simplesmente incorretos, conforme pode ser verificado na versão de Junho de 2016 da página arquivada no *Internet Archive Wayback Machine* que possui nome de arquivo e valor de *hash* incorretos[7].

A solução encontrada por outras ACs lançadas recentemente, como a *Let's Encrypt*, foi realizar assinatura cruzada de seus certificados intermediários, permitindo a verificação com sucesso de certificados em sua árvore até que a raiz tenha sido propagada[8]. Entende-se que essa abordagem não é de aplicação trivial, dado que é problemático confiar a soberania da ICP-Brasil a uma AC comercial já devidamente instalada em navegadores modernos, por isso outras abordagens serão sugeridas na Seção 6.

A solução ideal para distribuição das raízes ICP-Brasil é sua inclusão nos repositórios de certificados confiados por navegadores modernos. O ITI procura há 9 anos satisfazer requisitos impostos pela Mozilla, mas sem sucesso. Há uma longa discussão no fórum da organização[9], com participação de funcionários do ITI e ocasional ruído, que demonstra as dificuldades. Primeiramente, fica evidente que a falta de documentação em Inglês das normas e procedimentos dificulta o entendimento sobre a organização da iniciativa, agravado pela natureza peculiar da ICP-Brasil, que possui mais de uma dezena de ACs subordinadas comerciais e governamentais operadas de maneira independente. O termo utilizado para raízes desse tipo é *Super-CA*, com tratamento compatível ao aplicado às ICPs governamentais da Venezuela e Índia (comentário #122 na discussão).

A determinação da equipe Mozilla para o caso é que cada uma das ACs subordinadas da ICP-Brasil participe do processo de credenciamento em caráter individual, para apenas após a finalização do procedimento, as raízes poderem ser inseridas com sucesso. Isso não parece viável a curto prazo para o grande número de ACs da ICP-Brasil. Mais recentemente, há também questionamentos sobre escolha da empresa de auditoria de práticas na ICP-Brasil (comentário #155), que já esteve envolvida em auditorias deficientes em outros países. A leitura de toda a discussão é fascinante e fortemente recomendada para pesquisadores e profissionais da indústria envolvidos com certificação digital.

5. Prova de conceito de ataque

Baseando-se nos problemas de distribuição dos certificados raiz da ICP-Brasil, foi desenvolvida uma prova de conceito para exercitar um ataque em ambiente de testes.

5.1. Objetivo

A prova de conceito tem o objetivo de executar um ataque MITM contra um usuário que deseja ter acesso a qualquer serviço ou funcionalidade dependente da ICP-Brasil. O usuário simulado deseja proteger o conteúdo de todas suas informações de tráfego e

[7] https://web.archive.org/web/20160609022631/http://www.iti.gov.br/icp-brasil/repositorio/144-icp-brasil/repositorio/3886-repositorio-de-certificados-arquivo-unico-compactado

[8] https://letsencrypt.org/2015/06/04/isrg-ca-certs.html

[9] https://bugzilla.mozilla.org/show_bug.cgi?id=438825

© 2017 Sociedade Brasileira de Computação

navegação com o protocolo HTTPS, mas o ataque não levanta qualquer suspeita ou alerta do navegador ou do sistema operacional utilizado.

5.2. Detalhamento

O ataque MITM é uma interceptação maliciosa da comunicação entre duas partes de modo que essas partes julgam estar se comunicando diretamente entre si. Essa interceptação pode ser passiva, apenas observar o conteúdo das informações em trânsito, como também ativa, alterando as informações.

No cenário da prova, essa ação é executada por um atacante contra uma vítima conectada à mesma rede local. Antes da vítima acessar um serviço na Internet, o atacante convence mutuamente o computador atacado de que sua máquina representa o roteador da rede e vice-versa. Assim, o atacante passa a ser capaz de interceptar qualquer comunicação feita pela vítima, um ataque chamado *ARP spoofing*. Caso a vítima, no entanto, acesse serviços HTTPS, em que há autenticação do usuário e dados privados, existe uma camada criptográfica que o atacante não seria, em princípio, capaz de derrotar. O serviço em HTTPS possui um certificado digital válido atrelado ao seu endereço emitido por uma AC confiada por navegadores e sistemas operacionais que não poderia ter sido forjado por um atacante.

Uma técnica conhecida para contornar o uso de HTTPS pela vítima no ataque MITM é a *ssl-strip* ou *protocol downgrade* na qual as requisições HTTPS são reduzidas para HTTP pelo atacante, forçando, assim, uma conexão sem proteção criptográfica do usuário com o serviço. A adversidade é que, atualmente, os navegadores exibem um alerta visual quando não há a conexão HTTPS, podendo criar a suspeita pelo próprio usuário. Não somente, também há um parâmetro na resposta de requisições chamado *HTTP Strict Transport Security* (HSTS) que permite que servidores *Web* instruam os navegadores a interagir estritamente via protocolo HTTPS. Se ativa, esta política é guardada pelo navegador após a primeira comunicação com o servidor, portanto um ataque dessa natureza seria ineficaz caso a vítima tenha previamente acessado o serviço correto.

A prova de conceito para o ataque consiste exatamente em explorar as fragilidades na distribuição das ACs raízes da ICP-Brasil de modo a enviar ao usuário uma substituição maliciosa da raiz desejada, gerada pelo atacante. Assim, o sistema da vítima passa a confiar em certificados emitidos por essa raiz maliciosa, tornando possível também a substituição posterior dos certificados SSL de serviços HTTPS por certificados efêmeros emitidos pelo atacante. Isso torna possível o ataque MITM inclusive sobre o protocolo HTTPS, já que a entidade maliciosa se torna capaz de decifrar e recifrar o tráfego de informação, tendo acesso a todo seu conteúdo, cenário com maior impacto.

5.3. Metodologia

A implementação do ataque foi realizada através de uma instalação Kali Linux[10], uma distribuição GNU/Linux voltada para testes de penetração e análises de segurança. Esta distribuição facilita o uso de várias ferramentas, entre elas a *arpspoof* e *mitmproxy*, que serão usadas para interceptar o tráfego entre a vítima e o roteador; e para tratar as requisições e respostas tendo acesso ao conteúdo dos pacotes, respectivamente.

[10] `https://www.kali.org/`

 © 2017 Sociedade Brasileira de Computação

O primeiro passo é criar um certificado de chave pública auto-assinado que mimetize a raiz da ICP-Brasil. Essa tarefa pode ser executada através da interface de linha de comando do *OpenSSL*[11]. Ao definir o *Common Name* do certificado, adicionamos o prefixo *ROGUE* ao nome da AC raiz brasileira para evidenciar de que este é o certificado malicioso. Para filtrar as comunicações e permitir a manipulação das informações trafegadas, foi criado um programa em *Python*, a ser acoplado à execução da ferramenta *mitmproxy*. O filtro é responsável pelas seguintes adulterações:

1. Aplicar ataque de *downgrade*: o programa irá substituir todos os links HTTPS por HTTP enquanto ainda não temos a raiz maliciosa confiada pela vítima. Esse ataque demonstra como é inócuo criar *links* HTTPS a partir de páginas HTTP, pois esta já pode estar sob ataque, removendo os recursos de proteção adicionados pelo transporte SSL/TLS.
2. Identificar o *download* de um certificado de autoridade certificadora através do *header Content-Type* da resposta do servidor e fazer a substituição pelo certificado raiz malicioso.
3. Substituir *hash* de verificação do certificado original pelo *hash* do malicioso.

Definido o comportamento do filtro na máquina com o sistema Kali Linux, foi habilitado o modo *IP forwarding* e em seguida aplicado o ataque *ARP Spoofing* para interceptar o tráfego de rede entre roteador e vítima, fazendo com que a comunicação entre eles passe pela máquina do atacante. Resta modificar o conjunto de regras da tabela de IPs da máquina atacante com o intuito de desviar o tráfego HTTP e HTTPS para passar pelo *proxy* da aplicação *mitmproxy* e, por fim, sua execução com o filtro e a autoridade certificadora maliciosa criada anteriormente.

5.4. Ataque

O ataque montando foi executado em ambiente de teste controlado, com todas as partes cientes e acompanhando o processo. No início, a vítima, em computador pessoal com sistema operacional Windows e navegador Firefox, acessa seu suposto *webmail* provido por `https://expressobr.serpro.gov.br/`. O acesso é imediatamente interrompido pela mensagem de erro da Figura 1, pois o certificado de SSL do serviço não foi emitido por uma autoridade certificadora de confiança do navegador.

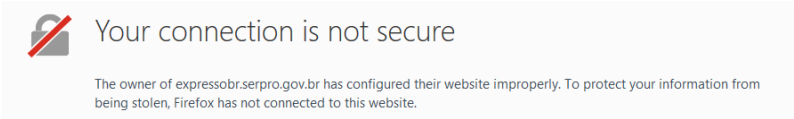


Figura 1. Erro de conexão desprotegida exibido pelo navegador.

A vítima, agora sob ataque, acessa a página do ITI para seguir os passos de configuração para o seu navegador, disponível em `http://iti.gov.br/navegadores/2-uncategorised/106-mozilla-firefox`. Ao clicar no *link* da cadeia v5, cadeia sob a qual está o certificado SSL do serviço desejado, o navegador irá sugerir à vítima a configuração de confiança para a raiz maliciosa, identificada pelo prefixo *ROGUE*, adulteração que passaria completamente despercebida pela vítima caso o desejado fosse gerar certificado com idêntico *Subject Name* da raiz brasileira v5.

[11] `https://www.openssl.org`

© 2017 Sociedade Brasileira de Computação

A página com os passos de configuração oferece um *link* para o valor do *hash* SHA-512 do arquivo da cadeia obtido, caso o usuário deseje verificar sua integridade, uma medida de segurança que é ineficaz contra o ataque proposto, que também permite adulterar o valor para concordar com o certificado malicioso. A Figura 2 exibe a página com o resumo criptográfico do arquivo sob ataque, evidenciado novamente pelo prefixo *ROGUE* adicionado.

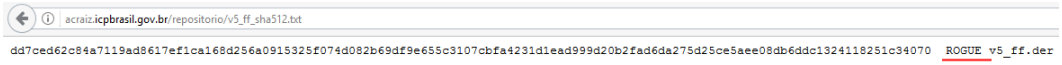


Figura 2. Página com resumo criptográfico do certificado raiz sob ataque.

Concluindo a configuração da confiança do certificado pelo navegador, a vítima retorna à página do *webmail*, agora com acesso positivo, validado pelo navegador, sem qualquer alerta de segurança (Figura 3). Por detrás da navegação usual, o atacante passa a ser capaz de examinar ou modificar o conteúdo de todo o tráfego da vítima, não somente no serviço de *webmail*, mas de qualquer outro *site* ou recurso acessado, uma vez que a aplicação do atacante é capaz de utilizar a raiz maliciosa para emitir dinamicamente certificados com endereço de qualquer domínio sendo acessado.

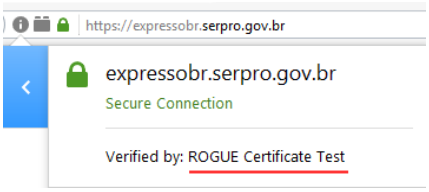


Figura 3. Página acessada pela vítima com certificado malicioso validado.

6. Recomendações

A tarefa de se projetar, implementar e manter uma infraestrutura de certificação digital é reconhecidamente desafiadora. Além dos aspectos técnicos e marco legal para validação jurídica de assinaturas digitais, há ainda dezenas de procedimentos para emissão de certificados, manutenção dos algoritmos criptográficos suportados, produção de carimbos de tempo e revogação de certificados cujas chaves privadas foram comprometidas. Esses desafios são ainda maiores para ICPs governamentais, que precisam aderir a conjuntos de práticas mais rigorosos, pelo risco inerente envolvido.

Desta forma, este trabalho apresentou uma análise de segurança das práticas vigentes na ICP-Brasil para distribuição de certificados raízes. A curto prazo, recomenda-se aprimoramento dos canais de distribuição de raízes para utilizar autenticação fornecida pelo protocolo HTTPS. Uma sugestão é utilizar as ACs de primeiro nível para descentralizar a distribuição, utilizando a solução adotada pela certificadora *Let's Encrypt*. Ou seja, permitir a certificação cruzada de certificados intermediários para também possibilitar que as ACs tenham páginas via SSL/TLS para distribuição de certificados e/ou verificação de integridade. Páginas com origem autenticada podem então ser utilizadas para distribuição de cadeias inteiras de certificados. O principal argumento de suporte à essa estratégia é que as ACs da ICP-Brasil já são confiadas do ponto de vista operacional, e podem também ser confiadas para distribuição das cadeias de certificados para instalação manual. A inclusão posterior de alguma AC em repositórios confiados facilitará a tarefa de distribuição de certificados raízes e fornecerá uma solução definitiva para o problema estudado aqui.

© 2017 Sociedade Brasileira de Computação

Referências

AccessNow (2011). The weakest link in the chain: vulnerabilities in the SSL certificate authority system and what should be done about them. https://www.accessnow.org/cms/assets/uploads/archive/docs/Weakest_Link_in_the_Chain.pdf.

Bernstein, D. J., Chou, T., Chuengsatiansup, C., Hülsing, A., Lambooij, E., Lange, T., Niederhagen, R., and van Vredendaal, C. (2015). How to manipulate curve standards: A white paper for the black hat. In *SSR*, volume 9497 of *LNCS*, pages 109–139. Springer. http://bada55.cr.yp.to.

Cooper, D. A. (1998). A Closer Look at Revocation and Key Compromise in Public Key Infrastructures. In *Proceedings of the 21st National Information Systems Security Conference*. NIST.

Huang, L. S., Rice, A., Ellingsen, E., and Jackson, C. (2014). Analyzing Forged SSL Certificates in the Wild. In *IEEE Symposium on Security and Privacy*, pages 83–97.

Kerner, S. M. (2015). Symantec Issues Fraudulent Google SSL Cert. http://www.esecurityplanet.com/network-security/symantecissues-fraudulent-google-ssl-cert.html.

Kohnfelder, L. M. (1978). Towards a practical public-key cryptosystem. B.S. Thesis, supervised by L. Adleman.

Kuhn, D. R., Hu, V., Polk, W. T., and Chang, S.-j. H. (2001). SP 800-32. Introduction to Public Key Technology and the Federal PKI Infrastructure. Technical report, Gaithersburg, MD, United States.

Laurie, B., Langley, A., and Kasper, E. (2013). Certificate transparency. RFC 6962: https://www.rfc-editor.org/rfc/rfc6962.txt.

Leavitt, N. (2011). Internet Security under Attack: The Undermining of Digital Certificates. *Computer*, 44(12):17–20.

Li, N. and Feigenbaum, J. (2001). Nonmonotonicity, user interfaces, and risk assessment in certificate revocation. In *Financial Cryptography*, volume 2339 of *LNCS*, pages 157–168. Springer.

Merkle, J. and Lochter, M. (2010). Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. RFC 5639: https://www.rfc-editor.org/rfc/rfc5639.txt.

Mozilla (2015). The MCS Incident and Its Consequences for CNNIC. https://blog.mozilla.org/security/files/2015/04/CNNIC-MCS.pdf.

Prins, J. (2011). DigiNotar Certificate Authority breach: "Operation Black Tulip". Interim report, Cybercrime Business Unit.

Roosa, S. B. and Schultze, S. (2013). Trust Darknet: Control and Compromise in the Internet's Certificate Authority Model. *IEEE Internet Computing*, 17(3):18–25.

WebTrust (2011). Trust Service Principles and Criteria for Certification Authorities v2.0. http://www.webtrust.org/homepage-documents/item54279.pdf.

WebTrust (2017). Principles and criteria for certification authorities: Ssl baseline with network security v2.2. http://www.webtrust.org/principles-and-criteria/docs/item83987.pdf.

© 2017 Sociedade Brasileira de Computação

Evaluating the Randomness of the RNG in a Commercial Smart Card

Wellinton Costa Ribeiro[1], Marcus Tadeu Pinheiro Silva[1]

[1]Departamento de Eletrônica e Biomédica – Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Campus II – 30510-000 – Belo Horizonte – MG – Brazil

wcostaee@gmail.com, tadeu@cefetmg.br

Abstract. *This paper brings results concerning the quality evaluation for the pseudo-random number generator (PRNG) in a commercial smart card. The RNG is a fundamental part for the cryptography carried out in several applications. We have acquired a huge quantity of random numbers from three samples of a commercial smart card. These data were evaluated using the statistical computation package developed by National Institute of Standards and Technology. In order to be used as gold benchmark and to validate our methodology, we have also tested the true random number generator (TRNG) included in a commercial integrated circuit. Our results show that the card PRNG owns quality too inferior than the TRNG. Due to card vendor confidentiality policy is not possible state the tested PRNG is base for the device cryptography. However, if this occurs, results lead us to conclude the tested PRNG is not adequate to provide the required security in the systems that adopt the evaluated smart card.*

1. Introduction

Typically, smart cards include a cryptographic unit, because the applications of this type of device require high security. However, eventual attackers will has total access to the smart card, what will allow to them act without restrictions existing in other applications where cryptography is used to guarantee security. Smart cards are also more vulnerable due to limitations imposed to them, including small dimensions, low cost and reduced power consumption [Akram 2012]. These factors lead to a security level lower than desirable, inasmuch as the cryptographic unit design can not to adopt the better solutions for some aspects. For example, the designer is forced to use a pseudo-random number generator (PRNG), while the ideal would be use a true random number generator (TRNG) instead.

In recent years, some researchers have reported failures in the cryptographic unit of commercial smart cards. The majority of these works has focused in the power consumption analysis methods in order to determine the private keys from the devices, what indicates necessity of new countermeasures from the vendors to avoid this vulnerability [Balasch 2012]. Other part of these works showed that the private keys can be determined when the PRNG in the smart card presents a weak randomness [Bernstein 2013].

 © 2017 Sociedade Brasileira de Computação

In this paper we present results about the quality assessment for the PRNG in a commercial smart card. In order to be a reference for our methodology we have also applied it to the TRNG included in a commercial integrated circuit (IC). Although the tested smart card has resource to provide direct access to the number generated in the PRNG, the device documentation makes none assertion about to be exclusively this PRNG the base for the cryptography.

2. Qualifying an RNG

When the internal structure of an RNG is not exactly known, assessment of its quality can be carried out only by statistical methods. So, a myriad of statistical tests has been developed to qualify the level of randomness for the sequences produced by an RNG. However, even the best RNG will eventually produce sequences that fail in one or other test. Consequently, the quality judgment for an RNG is a cumbersome process, involving generation of huge sequences and the assessment these sequences using many diverse tests. Several researchers have developed software tools to make this process easier [L'Ecuyer 2007][Marsaglia 2002][Rukhin 2010]. For the analysis in this paper we chose the Statistical Test Suite (STS) developed by National Institute of Standards and Technology (NIST) [Rukhin 2010].

3. Data Acquisition Setup

In this work three identical smart cards were tested. These cards includes only a contactless interface, which is compliant with the radio-frequency identification (RFID) standard ISO 14443B, being devices focused in access control applications, as stated in vendor documentation. As usual in RFID standards, the communication data rate is very limited. Furthermore, in the command to obtain data from the PRNG, only one byte is made available after each access. So, with the tested smart card the maximum data throughput achieved was 150 bps. An RFID reader was developed specifically for the work. This reader includes a microcontroller, a text LCD, a real time clock (RTC) with battery, a serial interface for communication with the host personal computer (PC) and a 1 MByte flash memory, the Intel i82802 IC. In order to validate the data acquisition setup and provide a gold benchmark for the results, we also carried out the generation of random numbers from the TRNG included in the i82802 [Taylor 2011]. Figure 1 shows the system block diagram and Figure 2 the corresponding assembled prototype. The firmware for the reader microcontroller was developed using C language, while for the PC was developed a data acquisition program written in Java. Concerning code execution, the smart card used is not user programmable, owning a proprietary operating system (OS) written in non-volatile memory during the manufacture. Further information about this OS is not available, in reason of the vendor confidentiality policy.

4. Methodology

For the three smart cards and i82802, we acquired huge bit streams, which were stored in files on a PC that controlled the data acquisition experiments. For the cards these bits streams were acquired in two modes. One these modes, called *on-off*, consisted of cycles on/off for the card, i.e., for each new number generated the card was power-off, from the interruption of the reader RF field, and some time after it was power-on again to gene-

© 2017 Sociedade Brasileira de Computação

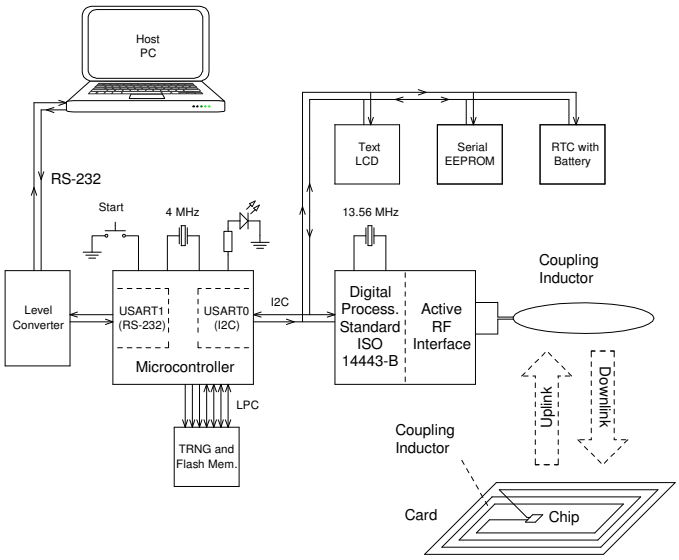


Figure 1. Block diagram of the system developed for this work.

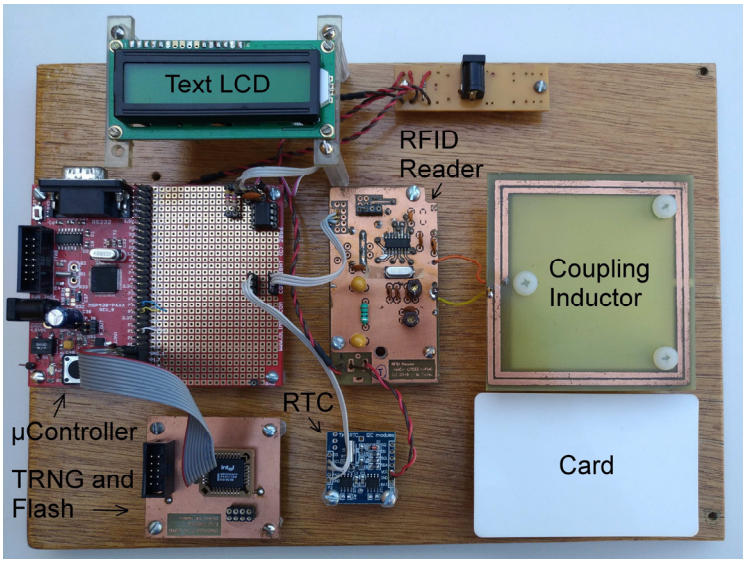


Figure 2. Prototype corresponding to figure 1.

rate the next number. In the other mode, called *continuous*, the card was continuously powered during the whole bit stream acquisition. The bit streams were processed by STS in blocks of n bits. When testing a RNG, in each statistical test is indispensable a high value for n, typically equal or upper to 10^5. Besides that, only is possible to qualify the generator if were acquired m sequences of n bits, since it is the rate of sequences validated as random that will establish the generator judgment in each test. The STS assumes an approval rating equal or upper 96% of the m sequences will qualify the generator as satisfactory. Our experiments were carried out with n equal to 10^5, m equal to 10^2, in compliance with values commonly reported in the literature [Boorghany

 © 2017 Sociedade Brasileira de Computação

Table 1. Rate of passing sequences on the NIST-STS. Tests for mode continuous. A generator is approved for a rate better or equal to 96%, except for the Universal and FFT tests, where the threshold is 90%. Not approval condition is indicated by an underlined value.

	i82802 (%)	Card 1 (%)	Card 2 (%)	Card 3 (%)
Frequency	99	93	81	73
Block Frequency	100	100	94	62
Cumul. Sums (fwd.)	98	90	81	74
Cumul. Sums (rev.)	99	94	81	73
Runs	99	94	75	88
Longest Run	100	71	80	20
Rank	100	99	99	29
FFT	100	100	70	0
Overlapping Templates	99	98	97	39
Approximate Entropy	100	6	12	0
Random Excursions	100	100	100	100
Random Exc. Variant	100	100	100	100
Serial	98	94	68	0
Linear Complexity	97	97	99	87
Universal	100	90	70	60

2014]. FFT and Universal tests of the STS require $n \geq 10^6$. So, for these specific tests, we have utilized n=10^6, m=10 and adopted an approval threshold equal to 90%. Due to RF interface low data rate, acquisition of 10 Mbits from each card was a time consuming task, with an approximate duration of 18.5 hours and 7 days, for the continuous and on/off modes, respectively. Concerning the i82802, the acquisition time was equal to 59 minutes and for this generator only the continuous mode was used.

5. Results

Tables 1 and 2 show the results in the continuous mode. Table 1 gives the results for 15 tests of NIST-STS. The i82802 IC has passed all tests. By contrast, the PRNG into cards 1, 2 and 3 has failed in 7, 10, and 13 tests, respectively. In addition to the tests in Table 1, NIST-STS also includes the Non-Overlapping Templates (NOT) test, whose results are presented in Table 2. NOT test validates each one of the m sequences over 148 different 9 bits templates; each template owns a non-periodicity character in its bit pattern. For example, Table 2 shows the i82802 failed only for one (1) in 148 templates. Specifically, for the template "101010000", where 5% of sequences failed; until 4% qualifies the generator as "passed in test". In the case of the smart cards the worst performance was for the number 3, where all templates leads to a not approval result and in the worst case 72% of sequences were rejected. If we consider each one of the 148 processed templates as a test itself, we will obtain a set of 163 tests. The results for

© 2017 Sociedade Brasileira de Computação

Table 2. Results for the NIST-STS Non-Overlapping Templates (NOT) test, mode continuous.

	i82802	Card 1	Card 2	Card 3
number failed templates	1	43	23	148
worst rate of not passed	5%	20%	13%	72%

the i82802 are compatible with the expected for a TRNG, since only one test had a not approval result and, even in this case, the rejection was by a minimum margin. Meanwhile, the card 2, that with better evaluation, failed in 33 tests. However, in some tests even this better card had a high rejection result, reaching 88% in the Approximate Entropy test.

Table 3 presents the results for the cards in on-off mode. It is immediately seen that the card 3 had a performance yet worst than in continuous mode. Between the cards 1 and 2, again the card 2 showed a better result. However, in general all cards had weak performance in comparison with the continuous mode. The on-off mode reported in Table 3 makes possible to examine the aspect of seed for the PRNG, although indirectly [Rank 2003]. It can be stated that the seed generation in the tested cards is very far from

Table 3. Rate of passing sequences on the NIST-STS. Tests for mode on-off. NC means "not carried out". Not approval condition is indicated by an underlined value

	Card 1 (%)	Card 2 (%)	Card 3 (%)
Frequency	15	21	12
Block Frequency	24	33	36
Cumul. Sums (fwd)	2	14	6
Cumul. Sums (rev)	4	15	6
Runs	5	7	4
Longest Run	32	32	22
Rank	85	88	75
FFT	0	0	0
Overlapping Templates	42	58	34
Approximate Entropy	11	10	14
Random Excursions	NC	NC	NC
Random Exc. Variant	NC	NC	NC
Serial	82	85	68
Linear Complexity	99	100	99
Universal	0	0	1

 © 2017 Sociedade Brasileira de Computação

an ideal entropy source, leading to weak randomness shown.

From the results, we can note the linear complexity test was that where the cards obtained the best approval level. In the Table 1, only the card 3 was not approved in this test, but by a not significant margin. In Table 3 this was the only test where the cards were approved. The focus of the linear complexity test is the length of a linear feedback shift register (LFSR) [Rukhin 2010]. So, in general way, the results allows to infer that the tested smart card includes an RNG based on LFSR, as commonly happens in the commercial designs [Akram 2012].

6. Conclusion

In this paper we have presented results concerning the quality evaluation for the RNG in a commercial smart card. Although it is not possible to claim that the evaluated RNG is effectively used in the card cryptographic unit, if that happen, the obtained results shows the smart card will represent significant weakness in the system security. In particular, one of the tested cards has presented so weak results that we believe in some manufacturing problem. If the proportion of these defective cards in a production batch becomes significant, it will arise an additional concern about the system security. Also, we have shown the convenience in adopt a TRNG device as gold benchmark, since it allow to make sure that is valid the used methodology.

Acknowledgment

The authors express your thanks to the Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) by support given to this work through its research scholarship program managed by the Departamento de Pesquisa e Pós-Graduação (DPPG).

References

Akram, R.N. et al. (2012). Pseudorandom number generation in smart cards: an implementation, performance and randomness analysis. In *5th IEEE International Conference on New Technologies, Mobility and Security (NTMS)*, May 7-10, Istanbul, Turkey.

Balasch, J. et al. (2012). Power analysis of Atmel CryptoMemory - recovering keys from secure EEPROMs. In *Topics in Cryptology - CT-RSA 2012, The Cryptographers' Track at the RSA Conference*, Lecture Notes in Computer Science 7178, O. Dunkelman (ed.), Springer-Verlag, pages 19-34.

Bernstein, D.J. et al. (2013). Factoring RSA keys from certified smart cards: Coppersmith in the wild. In *International Conference on the Theory and Application of Cryptology and Information Security*, Dec. 1-5, Bangalore, India, pages 341-360.

Boorghany, A. et al. (2014). Random data key generation evaluation of some commercial tokens and smart cards. In *Proc. of 11th International ISC Conference on Information Security and Cryptology*, Sept. 3-4, Tehran, Iran, pages 49-54.

L'Ecuyer, P., and Simard, R. (2007). TestU01: A C library for empirical testing of random number generators. In *ACM Transactions on Mathematical Software*, 33, (4), article 22.

© 2017 Sociedade Brasileira de Computação

Marsaglia, G., and Tsang, W. W. (2002). Some difficult-to-pass tests of randomness. In *Journal of Statistical Software*, 7,(3), pages 1-9.

Rankl, W. and Effing, W. (2003). Smart Card Handbook. 3rd edition, Wiley, Chichester, England, pages 210-213.

Rukhin, A. et. al. (2010). A statistical test suite for random and pseudo-random number generators for cryptographic applications. *NIST special publication 800-22*, National Institute of Standards and Technology, USA.

Taylor, G. and Cox, G. (2011). Digital randomness. In *IEEE Spectrum*, vol. 48, no. 9, pages 32-58.

 © 2017 Sociedade Brasileira de Computação

Artigos dos Workshops do SBSeg 2017

© 2017 Sociedade Brasileira de Computação

WFC – VI Workshop de Forense Computacional

© 2017 Sociedade Brasileira de Computação

Fast Similarity Digest Search: A new strategy for performing queries efficiently with approximate matching

Vitor Hugo Galhardo Moia , Marco Aurélio Amaral Henriques

[1]Department of Computer Engineering and Industrial Automation (DCA)
School of Electrical and Computer Engineering (FEEC)
University of Campinas (Unicamp)
13083-852 – Campinas, SP, Brasil

[vhgmoia,marco]@dca.fee.unicamp.br

Abstract. *Approximate matching functions can identify similar objects based on their digests, a useful resource for digital forensics investigations. However, comparing sets of approximate matching digests can be too time-consuming when dealing with large data sets since the usual comparison process is done by brute force (all-against-all). In this paper, we present Fast Similarity Digest Search (FSDS), a new strategy to perform digests searches in an efficient manner using TLSH approximate matching. Our results showed significant improvements over the brute force strategy for different scenarios.*

1. Introduction

The vast amount of data nowadays put intense pressure on forensics examiners for the development of new methods to analyze the seized devices in investigations, which usually contain terabytes of data that comes in a variety of sources: desktop computers, notebooks, smartphones, etc. The analysis of such data needs to be accomplished in a relatively short time, a problem given the current scenario.

One way of handling this amount of data efficiently is using the Known File Filtering (KFF) method. In a triage process, relevant information can be separated from irrelevant one by the use of reference data lists. Forensics examiners can use white lists to eliminate data from analysis (operating system, known software, and other inoffensive objects) or black lists to separate interest content (illegal or suspicious objects) [Breitinger and Baier 2012].

KFF is usually performed using hash functions, such as SHA-1 and SHA-2. However, they are inefficient due to limitations in the identification of objects which suffered small changes. It is well known that with these hash functions the change of a single bit in an object produces an entirely different output in comparison to the original version. Attackers can fool hash functions by making small changes on objects to bypass detection. Also, operating systems and software are regularly updated, and keeping track of each change is very hard.

Approximate matching functions appear as a suitable solution to mitigate the limitations of hashes. They are capable of detecting similarity between objects by creating a small and compact representation for data (digest) in a way that similar objects produce similar digests. However, this sort of function is very costly compared to the hashes, especially the comparison process involving large digest sets. Usually, this process is

 © 2017 Sociedade Brasileira de Computação

performed by brute force (all-against-all), taking days, weeks, or even months to be accomplished depending on the size of the sets, an impossible task for current investigations.

In this work, we present Fast Similarity Digest Search (FSDS), a new strategy to perform digest searches efficiently based on the Trend Micro Locality Sensitive Hash (TLSH) approximate matching tool. We show how our proposal works and make a comparison with the naive TLSH brute force approach to measure its improvements. We evaluated FSDS according to memory consumption, lookup and time efficiency. Our results show impressive improvements over the brute force strategy.

2. Background

In this section, we present basic concepts about approximate matching functions, followed by a brief description of TLSH, the tool which our strategy is based on.

2.1. Approximate matching

NIST defines approximate matching functions as *"Promising technology designed to identify similarities between two digital artifacts. It is used to find objects that resemble each other or find objects that are contained in another object"* [Breitinger et al. 2014b]. Also, it classifies such functions according to the operational level, such as Bytewise, Syntactic, or Semantic. Our work is restricted to only those functions operating in the Bytewise level, which does not try to interpret the object or consider its data structures. Thus, these functions are more time efficient and format independent.

Approximate matching functions follow the same concept of hashes: They create a small and compact representation of the object (digest), with fixed or variable size (depending on the tool). Comparing two digests to measure the objects similarity produces results usually on a scale of 0 (dissimilar) to 100 (very similar).

The applications for approximate matching is vast. They can be useful for forensics investigations in the identification of new versions of documents and software, detect embedded objects (e.g., an image inside a word document), objects in network packets, locate malware families, among others [Roussev 2011] [Harichandran et al. 2016].

There are lots of tools that perform the similarity identification by the use of digests in the bytewise level. Among them, we can mention the most popular ones: ssdeep [Kornblum 2006], sdhash [Roussev 2010], mrsh-v2 [Breitinger and Baier 2013], and TLSH [Oliver et al. 2013]. However, the focus of this work is on the TLSH tool. We aim at building an effective strategy to compare large sets of digests to avoid the expensive brute force approach. Two reasons drove our choice of this particular tool: there is no search strategy developed for TLSH so far (from our knowledge) and because TLSH present interesting characteristics, such as robustness to random changes and adversarial manipulations and space and time efficiency [Oliver et al. 2014].

2.2. The TLSH approximate matching tool

Proposed by Oliver, J. et al. [Oliver et al. 2013], this tool is based on the locality-sensitive hashing (LSH) scheme, producing a fixed size digest used to find similarities among objects. Given an object as input, TLSH starts processing it using a 5-bytes sliding window that moves byte-by-byte, extracts six trigrams in each step (combination of the window

 © 2017 Sociedade Brasileira de Computação

characters), and populates a 128-bit array of counter buckets using a mapping function (Pearson hash). Next, three quartile points are calculated based on the array. The final digest is composed of a header and a body, and its size is 35 bytes. The first part has 3 bytes corresponding to the quartile points, the object size, and a checksum, where one byte is reserved for each of these parameters. The second one, encompassing 32 bytes, is constructed by comparing each position in the counter buckets array to the quartile points, and the result is a bit pair defined according to the quartile the value ranges on.

The comparison function of TLSH is divided into two, each comparing one part of the digest. The first one, *Distance Header*, produces an output based on the object size and quartile points. The second one, *Distance Body*, calculates an approximation of the hamming distance between the digests. The sum of both functions is the final result, scoring 0 (zero) for identical (or nearly identical) objects or more for different ones.

3. Related work: Similarity Digest Search strategies

The process of comparing two digests data sets to find similar objects is referred to similarity digest search. When performing an investigation using the KFF, every digest from the target system under analysis is compared to every digest in a reference list. Results above a threshold are select for further and deeper analysis. This naive process, called brute force, becomes impractical when dealing with large data sets since it is too time-consuming. However, it is the usual strategy for most approximate matching tools.

We highlight that this problem is different from finding exact matches, which can be solved efficiently with ordinary databases. Our problem is harder to solve since we want to find items that are similar to each other by comparing only their digests.

Strategies to avoid the brute force method were proposed in the literature, such as DHTnil [Zhang et al. 2008], iCTPH [Jianzhong et al. 2010], F2S2 [Winter et al. 2013], MRSH-NET [Breitinger et al. 2014a], BF-based Tree [Breitinger et al. 2014c], and MRSH-CF [Gupta and Breitinger 2015]. These strategies present two basic phases: Preparation and Operational. In the first one, digests are created for each object of the reference list. Next, all digests are organized somehow according to the strategy to improve the search procedure. In the latter phase, digests are created for each object in the target system under analysis and then, each digest is compared to those compiled in the preparation phase.

The strategies mentioned earlier work with different approximate matching tools: Nilsimsa, ssdeep, mrsh-v2, and sdhash. However, newer developed tools can not work with current strategies due to their characteristics. TLSH is an example, even though it presents attractive features for forensics investigations as robustness to detect small changes in objects, small digests, and low latency to create and compare digests. With no better strategy than brute force, TLSH use becomes restricted. In this work, we developed a strategy to work with TLSH and thus reduce the required time for investigations.

4. Fast Similarity Digest Search strategy (FSDS)

In this section, we present the proposed strategy to perform similarity digests search efficiently, called FSDS. Our strategy was inspired on the F2S2 approach and uses a hash table to store digests based on their distance from a reference point. When performing a

 © 2017 Sociedade Brasileira de Computação

search in the structure to find similar objects, one needs to calculate a TLSH digest for the given object and compute the distance of such digest to a reference point. The result value points to a position in the hash table where similar objects will be located. Using TLSH comparison function with all digests in the given position and with others stored around it, we can figure out whether or not similar items reside in the structure. In the following subsection, we describe the details of our proposal along with our design choices. The implementation of our approach as well as extra material to explain it (figures and algorithm) can be found in our GitHub page [Moia and Henriques 2017].

4.1. Structure

FSDS uses a particular kind of hash table to store digests, consisted of a central array (*main table*) and buckets. Each position in the main table, referred as bucket, can keep multiple entries composed by one field only, the *ID*, a link to the corresponding digest.

4.2. Distance function

The distance function maps digests to positions in the main table according to their distance to a reference point. We choose the TLSH comparison function itself to perform such task since it estimates the hamming distance between two digests. This way, similar objects will produce similar digests with small differences between each other and mapped to the same position in the main table or near. For instance, let's suppose we have digests *A*, *B*, *C*, and *R*, where the later is our reference point. Digests *A* and *B* are similar (with distance $d_{AB} \sim 0$), while *A* and *C*, and *B* and *C*, are very different ($d_{AC} \sim d_{BC} >> 0$). When we compare *B* and *R*, we get a distance of d_{BR}, which is expected to be close to d_{AR} since *A* and *B* are similar. On the other hand, comparing C to R is expected to result in a value larger than d_{AR} and d_{BR}. It is important to mention that the distance is modular, i.e. this function is commutative (changing the order of the digests will not modify the result).

However, our distance function may store very different digests in the same bucket since this function is like a modulo operation. For instance, the character A is four positions distant from character E, but at the same time, I is also four positions distant from E. Taking E as the reference point, our function will store A and I in the same bucket since they present the same distance to E, even though the two characters are very different (8 position distant). There is also a problem with the header calculus in our distance function [Oliver et al. 2013], which may store distant digests in the same bucket.

Given the limitations of our distance function, we need to take an extra step when performing queries. We first need to calculate the distance from the input digest to the reference point. Then, all digests stored in the main table at the given position and also the ones around it are selected. The extra step is to perform a TLSH comparison of the queried digest with all selected candidates to confirm similarity. We emphasize that our strategy is meant only to reduce the number of comparisons one should make, which restrict the search to a small digest subset instead of the whole set.

4.3. Reference point

The reference point is used in conjunction with the distance function to map digests in the main table. Our choice is to use a random TLSH digest selected from our data set for such task. Section 5.1 discusses the impact of our decision based on some experiments.

 © 2017 Sociedade Brasileira de Computação

4.4. Main table size

The maximum size of the main table is restricted to the maximum difference between two digests since our distance function maps digests in to buckets based on their distance. We calculated this maximum distance based on TLSH comparison function (Header and Body distance) [Oliver et al. 2013], getting a result of 2089. This way, the main table will have at most 2089 buckets. On the other hand, the minimal size can be chosen by the examiner. However, the consequence of using a small table is a concentration of different digests in the same buckets, requiring more comparisons than usual.

4.5. Preparation phase: Inserting digests in the structure

The preparation phase of FSDS strategy consists in creating the main table and fill it up with our data set objects. First, we need to compute the TLSH digest for the objects from our set and assign an ID for each one. Then, we store the ID of each digest in the main table position related to the distance from the corresponding digest and the reference point, according to our distance function. Equation 1 summarizes this process.

$$T_{FSDSpp} = T_{creation} + r \cdot (T_{calc} + T_{comp} + T_{insertion}) \quad (1)$$

Here $T_{creation}$ is the time to create the main table, r the number of digests in the reference list, T_{calc} and T_{comp} the time to compute and compare a TLSH digest, respectively, and $T_{insertion}$ the time to insert the digest ID in the table.

4.6. Operational phase: Performing similarity digest queries with FSDS

After the preparation phase, FSDS will be available for handling queries. The object input has its TLSH digest created and then processed by the distance function to point out to the main table position where possible similar candidates will reside. All digests in that position along with their neighbors, those stored in near buckets, will be selected. Then, all candidates are compared with the queried digest to determine which ones are indeed similar. Eq. 7 estimates the time for FSDS operational phase.

$$T_{FSDSop} = T_{load} + q \cdot (T_{cal} + ((L + 1) \cdot T_{comp})) \quad (2)$$

where T_{load} is the time to load the structure in memory and q the number of target system digests. The parameter L denotes the number of similarity candidates, estimated by the average number of digests per bucket multiplied by the number of near buckets visited.

4.7. Number of visited buckets

A starting point to determine the number of visited buckets near the position indicated by the distance function is the TLSH threshold value (cuff off point). For instance, if we configure the threshold as 30, every comparison performed by TLSH which result is less than this value will be accepted as similar. Then, all digests in the 30 buckets below and above a given position will require inspection for similarity. However, we could increase/decrease this value according to the level of similarity demanded.

 © 2017 Sociedade Brasileira de Computação

4.8. Using FSDS

FSDS is implemented in C language and requires the pre-installation of the TLSH tool. When initialized, the forensics examiner needs to provide the hash table size (number of buckets) and choose one of the following options:

- **Reference digest list insertion:** The examiner provides a list of digests to be inserted in the FSDS structure.
- **Searching procedure:** The examiner chooses one of the following options:
  - **All objects:** All objects with similarity equal or less than a pre-defined threshold *t* with the queried object are returned.
  - **First object:** The search looks for one object with a similarity of at least *t* with the queried item, stopping the search after finding it.
  - **Closest object:** The object with smaller distance in comparison with the queried object is returned. No threshold is established, only a maximum number of buckets to visit.
- **Main table information:** Show information about the structure, such as the number of filled and empty buckets and distribution of the digests in the main table.

5. Assessment

The tests performed in this paper used the following machine: Linux Ubuntu 16.10 64-bit, Quad core (3425MHz) processor and 4 GB of memory. We measured the times using the `time` command (sys + user times) available on Linux distributions. We repeated all executions 20 times and took the average, with exception of TLSH brute force operational phase due to its large processing time. We took care of clearing the cache each time to prevent previous results influencing new ones.

Our experiments were conducted using *t5-corpus* as data set [Roussev 2011]. Its size correspond to 1.78 GiB of data, with 4457 items in different formats (HTML, pdf, text, doc, ppt, jpg, xls, gif). The average object size is 419 KiB.

5.1. Impact of the reference point

We performed several experiments in the chosen data set to find the best reference point. Our goal is to locate the one that better spread the digests in the FSDS structure to reduce the number of comparison in the final step and hence to avoid false positives. We considered different elements in our experiments, such as object type (xls, doc, ppt, pdf, text, jpg), size (ranging from 4 KB to 17 MB), entropy, and digest entropy.

An analysis of several experiments taking into consideration the four elements aforementioned was performed. After each test varying the parameters, we counted the number of empty buckets in the structure. Our results indicated no pattern regarding our choices. In general, the type, size, and entropy of an object caused a small impact in the digest spread in the main table. All tests presented similar results: $\approx 44\%$ of buckets filled up with digests on average. We chose a structure of 2^{11} buckets. It is important to mention that all tested objects have medium or high entropy, as well as their digests.

Since we could not identify any pattern for the choice of a good reference point, we picked a random one to continue our assessments. Future studies encompass the analysis of multiple reference points to create clusters of digests, where only the digests of a few clusters would be considered candidates for similarity.

 © 2017 Sociedade Brasileira de Computação

5.2. Memory requirement

Before creating the FSDS structure, we need to create TLSH digests for each object in our data set. The results must be stored in a database, XML file, or even in ordinary files to allow quick access to the digests for later identification. Next, we need to assign IDs to each digest, create the FSDS main table and insert the data into the structure. In the end, the amount of memory required by FSDS is given by Eq. 3:

$$M = r \cdot (s_{DIG} + f_n + s_{ID}) \tag{3}$$

Here r is the number of digests of our data set (reference list), s_{DIG} the size of a TLSH digest, f_n the size of the object name, and s_{ID} the size of the ID variable in the structure.

In our experiments, we have adopted the following values for these parameters: $r = 4457$, $s_{DIG} = 70\ bytes$, $f_n = 12\ bytes$, and $s_{ID} = 4\ bytes$. These choices led us to a memory requirement of $\approx 375\ KiB$. The TLSH brute force approach required $\approx 357\ KiB$, a small difference compared to FSDS strategy using the same *t5-corpus*.

5.3. Lookup efficiency

We also evaluated the effectiveness of our strategy in finding similar objects. We compared FSDS with the naive brute force concerning the three searching modes implemented in our approach: *All objects*, *First object*, and *Closest object*. Brute force requires an all-against-all comparison for the modes *All objects* and *Closest object*. The *First object* mode depends on the objects in the data set and their similarity with the queried one. Supposing that we have at least one object with similarity below the threshold with respect to the queried item, we expect to find it with half of all comparisons on average. Considering the *t5-corpus*, we hope to find a similar item with at least 2228 comparisons.

Performing the same search procedure using the FSDS strategy is more efficient. Table 1 shows our results for performing 20 lookups with random objects extracted from *t5-corpus* using the three searching modes with FSDS. We established a threshold of *t*=30 since it presents a low false positive rate [Oliver et al. 2013]. However, we are not interested in measuring false positives rates, but the potential of FSDS strategy in locating similar items. To this end, we evaluated *t5* using TLSH brute force to know the ground truth about the objects similarity. Our next experiment aims to find the similar objects pointed by TLSH but using FSDS instead. The maximum number of visited buckets chosen in our analysis is 20 since this value had the best balance between the number of digest comparisons and efficiency in finding similar items.

The results summarized in the table presents the object queried followed by the score of the most similar one in the set and the results for the three modes. The *All objects* mode encompass the number of comparisons necessary until the search is complete, the number of similar items found in relation to the expected one, and the score of the most similar object found. The *First object* and *Closest object* modes present the number of comparisons necessary in the search and the score of the similar object found.

According to the results, there is a significant reduction in the number of comparisons using FSDS in the three modes in relation to TLSH brute force. Considering the All objects and Closest object modes, only 3891 comparisons are necessary in contrast to the 88540 comparisons of brute force (4.39 % of the comparisons). Even though we could not

 © 2017 Sociedade Brasileira de Computação

Table 1. FSDS-TLSH: Lookup efficiency

Random objects (*t5-corpus*)	Minimum score	Searching modes: ALL objects Comparisons	ALL objects Found / Total	ALL objects Min. score	First object Comparisons	First object Score	Closest object Comparisons	Closest object Min. score
000649.text	5	203	6 / 6	5	5	11	203	5
001816.text	5	193	5 / 5	5	9	9	193	5
000047.xls	22	97	1 / 1	22	20	22	97	22
000073.html	19	200	1 / 1	19	179	19	200	19
000836.text	23	138	1 / 1	23	17	23	138	23
001434.text	8	212	5 / 5	8	67	10	212	8
001443.text	12	235	2 / 2	12	72	12	235	12
003056.html	9	245	5 / 9	9	3	14	245	9
001519.html	8	235	8 / 9	8	1	16	235	8
001531.html	15	189	3 / 3	15	11	15	189	15
001708.html	10	228	5 / 9	10	30	10	228	10
002138.html	0	229	2 / 2	0	4	0	229	0
004347.html	30	134	1 / 1	30	50	30	134	30
002038.html	0	286	6 / 6	0	3	0	286	0
000659.text	7	213	6 / 8	7	22	11	213	7
000935.text	18	221	3 / 5	18	86	18	221	18
001435.text	10	200	10 / 10	10	12	15	200	10
001619.doc	27	219	1 / 1	27	69	27	219	27
001723.html	25	167	1 / 1	25	96	25	167	25
002161.ppt	27	47	1 / 1	27	6	27	47	27

find all similar objects in some cases, such as the search for the object 003056.html which we found 5 objects from a total of 9, we were able to locate at least the most similar one as indicated by the minimum score in the table. When comparing the *First object* mode of the two strategies, we were able to identify a similar item in all cases. In 35% of cases we could not find the most similar one. However, the number of comparisons dropped from the 44270 of brute force to only 762 using FSDS (1.72%), showing the efficiency of this mode compared to the others.

5.4. Time efficiency

We measured the time for a single TLSH digest generation and comparison over the average object size to estimate the preparation (Eq. 4) and operational (Eq. 6) phases using TLSH brute force and also using FSDS strategy (Eq. 1 and 7). Table 2 summarizes our results. The following parameters were adopted in the experiments: $T_{calc} = 1.16 \cdot 10^{-1}$ (sec.), $T_{comp} = 1.29 \cdot 10^{-3}$ (sec.), $T_{create} = 1 \cdot 10^{-6}$ (sec.), $T_{insert} = 1 \cdot 10^{-6}$ (sec.), $T_{load} = 3.4 \cdot 10^{-4}$ (sec.), $L = 170.65$ and 20 as the maximum number of visited buckets.

We also applied brute force over *t5* to measure the necessary time to accomplish both phases to corroborate the previous results got from the formulas. The preparation phase was completed in 487.88 (sec.) while the operational one in 24,730.25 (sec.). The difference is a small but acceptable different which we believe is due to the `time` command restriction in detecting tiny differences of time and other low costs operations not accounted in the estimation. The difference is around 6.35% for both preparation and operational cases (on average).

Table 2. FSDS vs. Brute Force: Preparation and Operational costs

Strategy	Preparation phase (sec.)	Operational phase (sec.)
Brute force	517.01	26,102.94
FSDS (All objects mode)	522.76 (+1.11%)	1,502.41 (-94.24%)

Tab. 2 shows that brute force took for an all-against-all comparison about 7 hour and 15 minutes of processing time while FSDS, only 25 minutes (5.76%). The preparation phases of both strategies were similar, showing another benefit of FSDS since the

 © 2017 Sociedade Brasileira de Computação

extra cost for using it is not significant (see Sec 5.5). The reduction of the number of comparison (discussed in the previous section) is significant, even though an analysis of other data set sizes is required to measure the improvements regardless size of FSDS. However, the hours took by brute force in the whole process were reduced to minutes with FSDS.

The asymptotic complexity of FSDS is $O(q \cdot L)$. Since L is a fraction of r, the complexity of this approach becomes the same as brute force. However, in practice, the time to perform with FSDS is $1/x$ of brute force, where x is the division of the parameter r by L. This statement is corroborate by the experiments presented before. More details about the complexity, see documentation provided in the GitHub.

5.5. Break-even point

The FSDS preparation phase requires time to create the main table, calculate the TLSH digests of all objects in the reference list, and insert them into the structure (Eq. 1). On the other hand, the brute force approach requires only the time to calculate the digests of the reference list objects, given by Eq. 4.

$$T_{BFpp} = r \cdot T_{calc} \tag{4}$$

To amortize the extra cost of FSDS to brute force in term of queries:

$$T_{FSDSpp} + T_{FSDSop} = T_{BFpp} + T_{BFop} \tag{5}$$

where T_{BFop} is the operational phase of TLSH brute force, given by Eq. 6.

$$T_{BFop} = q \cdot (T_{calc} + (r \cdot T_{comp})) \tag{6}$$

We can ignore the costs of creating and loading the hash table (T_{create} and T_{load}) and inserting elements in the structure (T_{insert}) which can be very low compared to the other ones, as well as reading the digests from any database in the brute force approach. Then, we can estimate the break-even point by:

$$q' = \frac{r \cdot T_{comp}}{(r \cdot T_{comp}) - ((L+1) \cdot T_{comp})} \tag{7}$$

where q' represents the number of queries required to amortize the costs of FSDS.

Using the parameters presented in section 5.3, we can calculate q'. Our results yield that 1.04 queries are necessary, which means that performing two queries only are enough for amortizing the extra cost of FSDS over brute force. The reason for the low cost is due to the generation digest process being larger than the comparison one.

6. Conclusions and future work

Digital forensics investigations using approximate matching tools present a bottleneck when it comes to comparing large digests data sets. In this work, we proposed FSDS, a new strategy to efficiently analyze large digests data sets based on the TLSH tool. We performed experiments showing the efficiency of our approach in comparison to the simple brute force method in three different scenarios, where a reduction about 95% of the time was observed. Future work encompasses measuring the impact of using several reference points instead of only one and analyzing different data set sizes. We also plan to develop and test other distance functions to reduce even more the number of comparisons and hence the required time for forensics investigations. Another important future work is the comparison between FSDS and other strategies based on different tools.

 © 2017 Sociedade Brasileira de Computação

7. Acknowledgment

This work is partially supported by CAPES FORTE Project (23038.007604/2014-69).

References

Breitinger, F. and Baier, H. (2012). *Performance Issues About Context-Triggered Piecewise Hashing*, pages 141–155. Springer, Berlin, Heidelberg.

Breitinger, F. and Baier, H. (2013). *Similarity Preserving Hashing: Eligible Properties and a New Algorithm MRSH-v2*, pages 167–182. Springer, Berlin, Heidelberg.

Breitinger, F., Baier, H., and White, D. (2014a). On the database lookup problem of approximate matching. *Digital Investigation*, 11:S1–S9.

Breitinger, F., Guttman, B., McCarrin, M., Roussev, V., and White, D. (2014b). Approximate matching: definition and terminology. *NIST Special Publication*, 800:168.

Breitinger, F., Rathgeb, C., and Baier, H. (2014c). An efficient similarity digests database lookup-a logarithmic divide & conquer approach. *JDFSL*, 9(2):155.

Gupta, V. and Breitinger, F. (2015). How cuckoo filter can improve existing approximate matching techniques. In *International Conference on Digital Forensics and Cyber Crime*, pages 39–52. Springer.

Harichandran, V. S., Breitinger, F., and Baggili, I. (2016). Bytewise approximate matching: The good, the bad, and the unknown. *The Journal of Digital Forensics, Security and Law: JDFSL*, 11(2):59.

Jianzhong, Z., Kai, P., Yuntao, Y., and Jingdong, X. (2010). *iCTPH: An Approach to Publish and Lookup CTPH Digests in Chord*, pages 244–253. Springer, Berlin, Heidelberg.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3:91–97.

Moia, V. H. G. and Henriques, M. A. A. (2017). Fast similarity digest search. `https://github.com/regras/fsds`. Accessed 2017 Oct 20.

Oliver, J., Cheng, C., and Chen, Y. (2013). TLSH–a locality sensitive hash. In *Cybercrime and Trustworthy Computing Workshop (CTC), 2013 Fourth*, pages 7–13. IEEE.

Oliver, J., Forman, S., and Cheng, C. (2014). Using randomization to attack similarity digests. In *International Conference on Applications and Techniques in Information Security*, pages 199–210. Springer.

Roussev, V. (2010). Data fingerprinting with similarity digests. In *IFIP International Conf. on Digital Forensics*, pages 207–226. Springer.

Roussev, V. (2011). An evaluation of forensic similarity hashes. *Digital investigation*, 8:34–41.

Winter, C., Schneider, M., and Yannikos, Y. (2013). F2S2: Fast forensic similarity search through indexing piecewise hash signatures. *Digital Investigation*, 10(4):361–371.

Zhang, J., Lu, H., Lan, X., and Dong, D. (2008). DHTnil: An approach to publish and lookup nilsimsa digests in dht. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 213–218. IEEE.

 © 2017 Sociedade Brasileira de Computação

Avaliação da Eficácia de Classificadores de Malware ao Longo do Tempo

Fabrício Ceschin[1], **André Grégio**[1], **David Menotti**[1], **Marcos Castilho**[1]

[1] Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

{fjoceschin, marcos, gregio, menotti}@inf.ufpr.br

Abstract. *New malware samples are created to overcome defense mechanisms, demanding continuous improvement. Usually, the infection's behavior is not modified, but rather the way it is performed, thus invalidating signatures and detection heuristics. In this paper, we classify programs into malicious or benign using static features from a labeled dataset of over 38 thousand samples collected in the Brazilian cyberspace between 2013 and 2017. Then, we evaluate how these classifiers handle the number of changes faced in the malware scenario. We also release the dataset, attributes, classification algorithms and their parameters. Our results shed a light on the validity of classifiers and expose the need of constant updating security mechanisms.*

Resumo. *Novos programas maliciosos são criados para sobrepujar mecanismos de segurança, demandando contínua atualização. Em geral, o comportamento de infecção não é modificado, mas a maneira de executa-la, invalidando assim assinaturas e heurísticas de detecção. Neste artigo, propõe-se classificar estaticamente programas em maliciosos ou benignos usando um conjunto de dados com mais de 38 mil programas coletados no ciberespaço nacional entre 2013 e 2017. Avaliou-se então como os classificadores se comportam com as mudanças frequentes no cenário de malware. Todos os dados para reprodução são disponibilizados. Os resultados expõem a validade dos classificadores e a necessidade constante de atualização de mecanismos de segurança.*

1. Introdução

O problema dos ataques por *malware* torna-se mais grave com o passar dos anos, fazendo com que a dificuldade em se encontrar uma solução adequada aumente a distância entre a efetividade dos mecanismos de defesa e a complexidade, sofisticação, capacidade de evasão das ferramentas de proteção e disseminação por diversos meios dos exemplares atuantes. Aliado a isto, a quantidade massiva de *malware* moderno lançado na Internet diariamente traz a necessidade premente de criação de detectores automatizados. Em geral, a detecção de *malware* por meio de técnicas de aprendizado de máquina baseia-se na extração de características dos programas a fim de que se possa classificá-los em maliciosos ou benignos, ou mesmo apenas agrupar os maliciosos em famílias distintas, a depender dos atributos e conjuntos de dados utilizados.

Há, na literatura, inúmeras abordagens para classificação, agrupamento e detecção de *malware* cuja premissa é o uso de aprendizado de máquina [Gandotra et al. 2014]. Boa

 © 2017 Sociedade Brasileira de Computação

parte dos artigos publicados na área apresentam problemas de metodologia e/ou reprodutibilidade, seja por não disponibilizarem os dados, *scripts* e programas desenvolvidos para classificação dos exemplares e análise dos resultados, seja pela falta de representatividade/transparência na seleção de amostras. Ademais, o foco observado na literatura é na obtenção de taxas de acurácia próximas a 100%, não se notando a preocupação com o tempo de expiração das soluções de aprendizado de máquina produzidas. Logo, não se pode garantir que os resultados gerados são completamente isentos, uma vez que não há discussão sobre a eficácia dos classificadores ao longo do tempo, tampouco a base de dados para que outros pesquisadores façam novos testes.

Neste artigo, faz-se uma comparação entre alguns algoritmos clássicos de aprendizado de máquina, como *Support Vector Machines* (SVM), *Multi-layer Perceptron* (MLP), *Random Forest* e k-*Nearest Neighbors* (KNN), voltados ao problema da classificação de programas em maliciosos ou benignos em um conjunto de dados representativo de exemplares coletados no espaço cibernético nacional de 2013 à 2017. Principalmente, provê-se uma análise crítica do tempo de expiração de um classificador (como é o caso do antivírus): identifica-se, através do treinamento (aprendizado) e teste (avaliação) dos classificadores gerados ano a ano, quando estes perdem a eficácia de detecção devido à evolução dos exemplares de *malware* em atuação. Além disso, supre-se uma lacuna na área de aprendizado de máquina aplicado à segurança, disponibilizando-se publicamente os vetores de atributos rotulados de 38.490 de programas únicos (19.979 maliciosos e 18.511 benignos), bem como as meta-informações que os geraram e os *scripts* (em *Python*) produzidos para extração, processamento e análise dos dados e resultados. Como contribuição mais significativa, além da validação da hipótese da expiração de detectores de *malware* ao longo do tempo, este trabalho também destaca a necessidade da frequência atualização dos mecanismos de segurança para manutenção da acurácia na detecção de ameaças. Embora esta máxima seja óbvia e a indústria já aplique atualizações constantes em seus produtos, a literatura acadêmica da área nunca se preocupou com uma análise mais embasada dos testes feitos usando classificadores gerados por algoritmos de aprendizado de máquina. São comuns as publicações que aplicam determinado algoritmo da moda em um conjunto restrito de amostras buscando máxima precisão, sem se avaliar como seria o comportamento do classificador quando aplicado em amostras que evoluem ao longo do tempo, tal qual acontece na vida real.

O restante deste artigo está organizado da seguinte forma: na Seção 2, apresenta-se trabalhos relacionados com detecção de *malware* usando aprendizado de máquina; a base de dados utilizada (os vetores de descritores estão disponíveis ao público em [Ceschin et al. 2016]) neste trabalho é descrita na Seção 3; o processo de representação para aprendizado de máquina dos atributos extraídos dos programas é detalhado na Seção 4; os experimentos realizados, bem como a análise da expiração dos classificadores de *malware* ao longo do tempo são apresentados na Seção 5. Finalmente, na Seção 6, apresenta-se as considerações finais do trabalho.

2. Trabalhos Relacionados

A literatura de detecção, classificação e/ou agrupamento de programas maliciosos usando aprendizado de máquina é vasta [Gandotra et al. 2014], podendo ser dividida em três tipos de acordo com as características extraídas: baseados em atributos estáticos provenientes do binário executável, em atributos dinâmicos provenientes da execução em

 © 2017 Sociedade Brasileira de Computação

ambientes controlados e, por fim, abordagens híbridas. Além disso, pode-se dividir as soluções entre as que consideram o componente de detecção (usam programas maliciosos e benignos para treinamento e teste do algoritmo de classificação) e aquelas que tentam classificar programas desconhecidos em grupos pré-existentes de programas sabidamente maliciosos. Entretanto, não há um *dataset* padrão que possa ser utilizado como *ground truth* para avaliação dos classificadores gerados, bem como raramente há disponibilização dos dados. A seguir, são mencionados alguns artigos que abordam a classificação de *malware* por meio da aplicação de técnicas de aprendizado de máquina, os quais apresentam os trabalhos relacionados de suas respectivas épocas.

O primeiro artigo ([Schultz et al. 2001]) sobre classificação de programas (desenvolvidos para o sistema operacional *Windows*) em duas classes (*malware* e *goodware)* surgiu em 2001 e foi o primeiro a introduzir o conceito de *data mining* para a detecção de *malware*, contendo 1.000 exemplares de ambas as classes e usando a lista de *DLLs*, chamadas de sistema, número de diferentes chamadas de sistemas usadas em cada *DLL*, *strings* e sequência de bytes não sobrepostas como *features* no classificador *Naive Bayes*. Três anos depois, [Kolter and Maloof 2004] usou as mesmas *features*, porém usando sequências sobrepostas de quatro bytes em uma lista mais extensa de classificadores, tais como *Support Vector Machine* e *Decision Tree*. Anos depois, em 2011, [Nataraj et al. 2011] extraiu *features* usando técnicas de processamento de imagem para visualizar os binários malignos como imagens em escala de cinza, classificando seus 9.581 exemplares (9.458 *malware* e 123 *goodware*) usando o *K-nearest neighbors* e obteve $98,08\%$ de acurácia. No mesmo ano, [Anderson et al. 2011] foi um dos primeiros trabalhos a usar a técnica de análise dinâmica. A mesma utiliza uma matriz de similaridade resultante de grafos que representam cadeias de Markov. O trabalho utilizou essas *features* em um *SVM* e obteve uma acurácia de $96,41\%$. No ano seguinte, em 2012, o mesmo autor ([Anderson et al. 2012]), usando *features* estáticas e dinâmicas, construiu, usando um grafo de fluxo de controle, rastreamento dinâmico de instruções e chamadas de sistema e as informações dos arquivos, obteve uma acurácia de $98,07\%$, um resultado bem próximo ao obtido em [Nataraj et al. 2011]. Entretanto, apesar desses resultados, a maioria dos trabalhos citados não possui um conjunto representativo exemplos. [Nataraj et al. 2011], por exemplo, usa apenas 9.581 exemplares. Por outro lado, [Kantchelian et al. 2013] usa um *dataset* que contém 188.704 exemplares, em que 94.352 são *goodware* e 123.845 são *malware*, e os classifica usando um *ensemble* de classificadores, um para cada família de *malware* (eles não reportam a acurácia, apenas o *false negative rate* – 55% – e *false positive rate* – 1%). No geral, é claro de perceber o quão difícil é comparar os trabalhos nessa área, uma vez que cada um utiliza uma massa de dados diferentes e não disponibilizam os mesmos, o que impossibilita a reprodutibilidade dos mesmos.

Além dos trabalhos citados no parágrafo anterior, existem questionamentos envolvendo abordagens baseadas em aprendizado de máquina: a maioria das avaliações não consideram o tempo, um fator importante já que os exemplares de *malware* estão sempre evoluindo (eles alteram seus conceitos, um problema chamado de *concept drift*) em resposta a pressões externas (novas tecnologias e detectores) e internas (novas funcionalidades) [Singh et al. 2012]. Tentando resolver este problema, [Roberto Jordaney and Cavallaro 2016] apresenta o *conformal evaluator*, um *framework* de avaliação que usa métricas estatísticas para medir a qualidade dos resultados produ-

 © 2017 Sociedade Brasileira de Computação

Tabela 1. Distribuição dos exemplares de *malware* e *goodware* entre 2013 e 2017.

Ano	<=2013	=2014	=2015	=2016	>=2017	Total
Malware	10.075	8.688	6.461	728	848	26.800
Goodware	11.930	512	1.057	5.009	3	18.511

zidos. [Roy et al. 2015] conclui que *AUPRC* (*Area Under the Precision Recall Curve* é uma métrica melhor para comparar resultados de diferentes abordagens em detectores para Android baseados em aprendizado de máquina.[Allix et al. 2015] mostra que utilizar um conjunto aleatório de *malware* conhecido para treinar um detector, como é feito em vários experimentos na literatura, produz resultados significativamente tendenciosos, ao passo que [Jordaney et al. 2017] introduz um *framework* chamado *Transcend*, capaz de identificar o envelhecimento de um modelo baseado em aprendizado de máquina, antes mesmo de sua performance começar a cair, através de comparações estatísticas, que possibilita identificar *concept drift* em *malware* dos sistemas *Android* e *Windows* – a análise feita em cima dos exemplares de *Windows*, entretanto, não utiliza o problema de classificação em duas classes abordada neste trabalho, mas sim a classificação em famílias.

3. Conjunto de Dados Maliciosos e Benignos

O estudo realizado neste trabalho utiliza uma massa de dados de ≈180 GB em arquivos executáveis que foram coletados (Seção 3.1) no espaço cibernéticos nacional de 2013 à 2017 como ilustra a Tabela 1. Para cada arquivo coletado—rotulados como *goodware* ou *malware*, dependendo da sua origem—foram extraídos atributos para sua representação (Seção 3.2). A base de vetores de atributos e seus rótulos serão utilizados para aprendizado supervisionado de classificadores [Bishop 2006] capazes de predizer a classe de um arquivo desconhecido.

3.1. Coleta

Para a construção do *dataset*, foi necessário obter duas classes de *software*: *goodware* e *malware*. Os exemplares de *goodware* foram coletados utilizando um *web crawler* em três sites de *download*: *Sourceforge* [Slashdot Media 2016], *Softonic* [Softonic Internacional S.A. 2016] e *CNET Download* [CNET 2016]. No total, ≈130 GB de arquivos foram baixados, contabilizando 18.511 amostras únicas de *goodware*. Os exemplares de *malware* foram obtidos de uma instituição financeira nacional que prefere se manter anônima, advindos de mecanismos de segurança instalados em seus clientes ou coletados de *phishing*. Até o momento deste trabalho, foram coletados ≈50 GB de binários de *malware*, totalizando 26.800 exemplares (19.979 únicos).

3.2. Extração de Atributos

Esta etapa tem como objetivo extrair os atributos dos executáveis obtidos, de forma que fiquem disponíveis publicamente para futuros trabalhos. Para isso, a biblioteca *pefile* [Carrera 2016], que permite acessar atributos de arquivos do tipo *PE (Portable Executable)*, foi utilizada. Os atributos extraídos de cada executável através dessa biblioteca, além de atributos gerais como nome, tamanho, *hash* (MD5 e SHA1) e a entropia média, são os seguintes: *Machine*, *Number Of Sections*, *Time Date Stamp*, *Pointer to Symbol Table*, *Number of Symbols*, *Size of Optional Header*, *Characteristics*, *Magic*, *Size*

 © 2017 Sociedade Brasileira de Computação

of Code, *Size of Initialized Data*, *Base of Data*, *Image Base*, *Size of Image*, *Size of Headers*, DLLs e funções importadas e lista de nomes de empacotadores, compiladores ou ferramentas usadas. Mais informações sobre cada atributo podem ser encontradas em [Pietrek 1994, Yonts 2010]. Os atributos coletados foram, então, escritos em dois arquivos de texto no formato *csv*, um para cada classe de *software* (*goodware* e *malware*).

4. Representação

A análise dos dados coletados permitiu dividir os atributos em três categorias: atributos únicos (únicos para cada executável), numéricos (números inteiros ou reais) e textuais (conjunto de palavras). As seções a seguir tratam do pré-processamento dos atributos textuais (Subseção 4.1) e da normalização de todos os atributos (Subseção 4.2).

4.1. Pré-Processamento

O pré-processamento dos atributos textuais tem como objetivo agrupar os textos (documentos) parecidos. Desta forma, programas que utilizam as mesmas bibliotecas, funções e compiladores tendem a ficar próximos no espaço de características. Para que isso seja possível, cada conjunto de palavras é transformado em um documento (no contexto de recuperação de informação), em que cada palavra é separada por um espaço. Assim, três "documentos" são gerados por arquivo, um para cada atributo textual. Em seguida, é feito o *case-folding* e a normalização de todos os textos: todos eles são mantidos em caixa baixa e qualquer caractere especial é removido, além de acentos e números. Com isso, é possível obter medidas estatísticas sobre cada documento, com base em suas palavras. Neste trabalho, utilizamos o modo de representação *Vector Space Model (VSM)*, o qual representa um documento por meio da importância relativa de suas palavras [Turney and Pantel 2010, Manning et al. 2008].

4.1.1. Vector Space Model

Dado um vocabulário de um conjunto de documentos, isto é, todas as palavras que aparecem neste conjunto, cada documento *i* é representado por um vetor $\vec{d_i} = (w_{i,1}, w_{i,2}, ..., w_{i,t})$, em que $w_{i,j}$ representa o *TF-IDF* (do inglês *Term Frequency—Inverse Document Frequency*) da *j-ésima* palavra do vocabulário. O *TF-IDF* é uma medida estatística utilizada para avaliar o quão importante uma palavra é para um documento em relação à uma coleção de documentos [Manning et al. 2008]. Esta medida é obtida através da multiplicação de dois termos: o *Term Frequency (TF)*, que mede com que frequência um termo *t* ocorre em um texto, mostrado na Equação 1, e o *Inverse Document Frequency (IDF)*, que mede o quão importante é um termo *t*, apresentado na Equação 2. Em resumo, cada texto é representado por um vetor esparso que contém suas medidas de *TF-IDF* para cada palavra do vocabulário, que pode ser reduzido a um número *V* de palavras (as mais presentes nos textos).

$$TF(t) = \frac{\text{Número de vezes que } t \text{ aparece no documento}}{\text{Número total de palavras do documento}} \tag{1}$$

$$IDF(t) = \log_e \left(\frac{\text{Número total de documentos}}{\text{Número de documentos com a palavra } t} \right) \tag{2}$$

 © 2017 Sociedade Brasileira de Computação

4.2. Normalização

Após o pré-processamento dos atributos textuais, o *Vector Space Model* de cada um é concatenado aos demais atributos de um arquivo, gerando um vetor de tamanho $22 + 3 \times V$, em que 22 é o número de atributos numéricos e *V*, o tamanho do vocabulário criado dos atributos textuais. Entretanto, ainda há uma diferença de escala entre as características extraídas, sendo necessário normaliza-las. A normalização feita sobre as características extraídas foi a *MinMax*, que escala todas as *features* (características) em um intervalo entre zero e um. A partir disto, todos os arquivos estão normalizados e prontos para a classificação.

5. Experimentos Realizados

Com as representações (conjunto de *features*) extraídas conforme a seção anterior, foram inicialmente realizados experimentos com quatro classificadores: *Multilayer Perceptron* (*MLP*), o tipo mais frequente de rede neuronal em classificação; *Support Vector Machines* (*SVM*); *K-Nearest Neighbors* (*KNN*); *Random Forest* (um *ensemble* de árvores de decisão). O *Multilayer Perceptron* utilizado é implementado pela *TFLearn*, uma *API* de alto nível construída sobre a *TensorFlow*, biblioteca de inteligência artificial do *Google* [Damien et al. 2016]. Para os experimentos, o *MLP* consistiu de duas camadas escondidas—a primeira constituída de *V*/2 neurônios e a segunda de *V*/3, em que *V* é o tamanho do vocabulário utilizado pelo *Vector Space Model*. Os demais classificadores utilizados são implementados pela biblioteca *Scikit Learn* [Pedregosa et al. 2011], sendo o *SVM* (implementação *SVC*), utilizado com *kernel* linear (*default* $C = 1$ – experimentos foram feitos com *kernel* RBF e *grid-search*, mas não foram promissores), e o *KNN*, com o valor de *K* fixado em 5. Valores diversos foram experimentados para os parâmetros e os mencionados foram os que obtiveram os melhores resultados em geral. A Subseção 5.1 apresenta os resultados gerais obtidos, seguido pela análise da expiração de classificadores (Subseção 5.2).

5.1. Resultados Gerais

Para validação dos experimentos, 50% dos exemplares de *goodware* e *malware* foram utilizados no conjunto de teste, enquanto que o restante foi para o conjunto de treino. Este procedimento foi realizado dez vezes pelo esquema de validação cruzada com dez partições (*folds*). O tamanho do vocabulário *V* adotado foi de 100 palavras. A Tabela 2 apresenta os valores médios obtidos desse processo para as medidas de acurácia, *Recall*, *Precision* e *F1-score* – medidas clássicas para problemas de classificação binário (*malware*/*goodware*) em que há desbalanceamento de classes – para cada classificador utilizado. É possível observar que o classificador *Random Forest* obteve os melhores resultados, seguido pelo *KNN*, *MLP* e *SVM*, que ficou quase três pontos percentuais abaixo em praticamente todas as métricas. Entretanto, ainda há o questionamento se o método de validação cruzada é válido para testar estes modelos, uma vez que este método mistura aleatoriamente os exemplares, fazendo com que o classificador reconheça exemplares de diversas épocas, sem levar em conta o problema de mudança de conceito ao longo do tempo.

5.2. Análise da Expiração de Classificadores

Dois experimentos representando cenários reais foram conduzidos para se avaliar a expiração de classificadores que obtiveram os melhores resultados no experimento

 © 2017 Sociedade Brasileira de Computação

Tabela 2. Valores médios (10 execuções) obtidos nos classificadores testados.

Classificador	Acurácia	F1-score	Recall	Precision
SVM	95.45%	95.56%	93.11%	98.15%
MLP	97.57%	97.70%	97.69%	97.71%
KNN	97.65%	97.76%	97.51%	98.02%
Random Forest	**98.26%**	**98.34%**	**97.81%**	**98.87%**

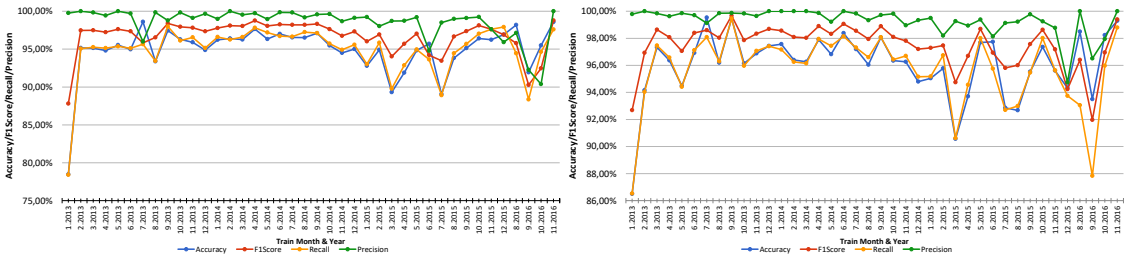
anterior, i.e., *Random Forest* e *KNN*.

5.2.1. Experimento #1

Neste experimento, os exemplares foram divididos em dois grupos: um contendo exemplos criados até determinado mês de um ano[1], usados para treino, e o outro com exemplos criados um mês após o mês escolhido, usado para validação. Por exemplo, se o mês é Fevereiro e o ano, 2015, o conjunto de treino terá exemplos que foram criados até Fevereiro de 2015 (conjunto cumulativo), enquanto que o conjunto de validação terá exemplos criados apenas em Março de 2015 (um mês após Fevereiro de 2015). Este experimento simula uma situação do mundo real, já que nós treinamos com dados que já foram vistos, i.e., *malware* e *goodware* que foram criados antes de determinado mês, e testado com dados do presente (do mês selecionado). A Figura 1(a) mostra os resultados para o *KNN*, enquanto a Figura 1(b) apresenta os resultados para o *Random Forest*, com acurácia, *f1score*, *recall* e *precision*. Em ambos os casos, existe uma grande variância nas métricas conforme o tempo passa. No caso do KNN, a melhor acurácia foi obtida em Julho/2013, com 98,61%, e o pior, em Janeiro/2013, com 87,81%. O *f1score* também é pior em Janeiro/2013, com 87,81%, e o melhor, em, Novembro/2016, com 98,78%. Olhando para o *recall*, o pior resultado também foi em Janeiro/2013, com 78,41%, enquanto que o melhor foi em Setembro/2013, com 97,99%. Por outro lado, a melhor *precision* foi obtida em Outubro/2016, com 90,38%, e a melhor, em Fevereiro/2013, Maio/2013, Fevereiro/2014 e Novembro/2016, com 100%. Estes resultados mostram que o *KNN* é um bom classificador para evitar falsos positivos, já que obteve 100% de precisão em 4 meses, e tem limitações para evitar falsos negativos, neste caso, já que o *recall* nunca atinge 100%. No caso do *Random Forest*, a pior acurácia foi em Janeiro/2013, com 86,50%, e o melhor, em Julho/2013, com 99,54%, enquanto que o pior *f1score* foi 91,97% em Setembro/2016 e o melhor, em Setembro/2013, com 99,66%. Olhando para o *recall*, o pior obtido foi em Janeiro/2013, com 86,53%, enquanto que o melhor foi obtido em Setembro/2013, com 99,46%. A pior *precision* foi obtida em Dezembro/2015, com 94,74%, e o melhor, em Fevereiro/2013, Dezembro/2013, Janeiro/2014, Fevereiro/2014, Março/2014, Junho/2014, Agosto/2016 e Novembro/2016, com 1200% em todos eles. É possível concluir que o *Random Forest* é um melhor classificador em comparação ao *KNN*, já que todas as métricas foram melhores na maioria dos casos. A queda significativa e a recuperação de todas as métricas com o passar do tempo em ambos os classificadores evidencia a existência de um *concept drift* neste *dataset*.

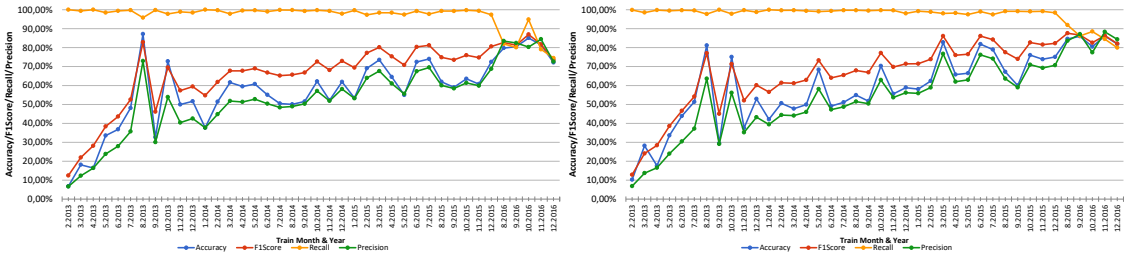
[1]Nos casos dos *malware*, o ano corresponde ao que o exemplo foi coletado, i.e., quando ele foi recebido por nosso parceiro; No caso dos *goodware*, foi utilizada a data de compilação.

 © 2017 Sociedade Brasileira de Computação



(a) KNN (b) Random Forest

Figura 1. Resultados obtidos no Experimento #1.



(a) KNN (b) Random Forest

Figura 2. Resultados obtidos no Experimento #2.

5.2.2. Experimento #2

No experimento #2, inverteu-se os conjuntos de treino e validação do experimento #1. Com isso, o conjunto de treino possui exemplos de um determinado mês e ano apenas, enquanto que o conjunto de validação possui exemplos criados um mês antes deste mês (conjunto cumulativo). Por exemplo, dado o mês de Fevereiro e o ano de 2015, o conjunto de treino irá conter exemplos criados apenas este mês e ano, enquanto que o conjunto de teste conterá exemplos criados até um mês antes (Janeiro/2015). Este experimento também simula o mundo real, entretanto, nós usamos apenas dados de um mês selecionado (presente) para treinar e o resto (passado, antes deste mês), para validação. Com isso, nós podemos ver o quão representativo exemplos de um mês são para todo o conjunto de dados. A Figura 2(a) mostra os resultados obtidos pelo *KNN*, enquanto que a Figura 2(b) mostra os resultados obtidos pelo *random forest*, com as métricas de acurácia, *f1score*, *recall* e *precision*. Analisando os resultados do *KNN*, é possível ver uma acurácia realmente muito baixa no começo (Janeir/2013), com apenas 6,78%, sendo o melhor resultado 87,15%, em Agosto/2013. O melhor *f1score* também foi obtido em Agosto/2013, com 81,82%, e o pior, também em Janeiro/2013, com 12,47%. É curioso olhar para o *recall* e a *precision*: o *recall* atinge bons resultados, com 100% em Janeiro/2013, Abril/2013 e Janeiro/2013 (melhores resultados) e começa a cair em Dezembro/2015, obtendo 74,50% em Dezembro/2016 (pior resultado). A *precision*, por outro lado, tem 6,65% em Janeiro/2013 (pior resultado) e começa a aumentar conforme o tempo passa, apesar de algumas quedas no caminho, chegando a 84,45% em Novembro/2016. Apesar do *random forest* apresentar melhores resultados em geral, seus gráficos são muito similares aos do *KNN*. O *Random Forest* obtem a melhor acurácia em Março/2015, com 82,78%, e o pior, em Janeiro/2013, com 10,26%. Seu pior *f1score* foi também obtido em

 © 2017 Sociedade Brasileira de Computação

Janeiro/2013, enquanto que o melhor foi obtido em Agosto/2016, com 87,57%. O comportamento do *recall* também é similar ao *KNN*, já que começa a cair em Dezembro/2015 e possui valores próximos à 100% após este mês (nunca chega a 100% como o KNN, o melhor resultado foi em Janiro/2014, com 99,99%) e o pior resultado também foi em Dezembro/2016, com 79,96%. A *precision* também é similar ao *KNN*, com o pior resultado em Janeiro/2013, com 6,88%, e o melhor, em Novembro/2016, com 88,36%.

6. Conclusão

Neste artigo, foi estudado, implementado e avaliado um método estático para a detecção de malware baseado em aprendizado de máquina para analisar a expiração de classificadores (SVM, MLP, KNN e *Random Forest*) ao longo do tempo. Os atributos usados foram extraídos de arquivos executáveis do Windows no formato PE, tanto da classe *goodware* quanto da classe *malware*. Os vetores de atributos (e a descrição destes) obtidos a partir da base de dados construída para esse trabalho estão disponíveis à comunidade científica, de forma que uma comparação justa com outros trabalhos possa ser realizada, bem como para que os resultados possam ser reproduzidos ou testados com outras técnicas. Dos experimentos realizados neste trabalho podemos destacar os seguintes resultados: (i) o classificador *Random Forest* foi aquele que permitiu obter o melhor resultado geral utilizando validação cruzada; (ii) a hipótese levantada da expiração de classificadores ao longo do tempo foi verificada com sucesso, destacando a necessidade de uma frequente atualização dos mecanismos de segurança para manutenção da acurácia.

Agradecimentos

Os autores agradecem ao Google pelo financiamento deste projeto via Programa Google LARA - *Latin America Research Award* 2017.

Referências

[Allix et al. 2015] Allix, K., Bissyandé, T. F., Klein, J., and Le Traon, Y. (2015). Are your training datasets yet relevant? In *ESSoS'15*.

[Anderson et al. 2011] Anderson, B., Quist, D., Neil, J., Storlie, C., and Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in Computer Virology*, 7(4):247–258.

[Anderson et al. 2012] Anderson, B., Storlie, C., and Lane, T. (2012). Improving malware classification: Bridging the static/dynamic gap. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, AISec '12, pages 3–14, New York, NY, USA. ACM.

[Bishop 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Carrera 2016] Carrera, E. (2016). pefile: A python module to read and work with pe (portable executable) files. `https://github.com/erocarrera/pefile`.

[Ceschin et al. 2016] Ceschin, F. et al. (2016). malware-machinelearning. `https://github.com/fabriciojoc/malware-machinelearning`, acessado em dezembro de 2016.

[CNET 2016] CNET (2016). CNET Downloads. `http://download.cnet.com/`.

 © 2017 Sociedade Brasileira de Computação

[Damien et al. 2016] Damien, A. et al. (2016). Tflearn. https://github.com/tflearn/tflearn.

[Gandotra et al. 2014] Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 5(2):56–64.

[Jordaney et al. 2017] Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I., and Cavallaro, L. (2017). Transcend: Detecting concept drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, Vancouver, BC. USENIX Association.

[Kantchelian et al. 2013] Kantchelian, A., Afroz, S., Huang, L., Islam, A. C., Miller, B., Tschantz, M. C., Greenstadt, R., Joseph, A. D., and Tygar, J. D. (2013). Approaches to adversarial drift. In *AISec'13*.

[Kolter and Maloof 2004] Kolter, J. Z. and Maloof, M. A. (2004). Learning to detect malicious executables in the wild. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 470–478, New York, NY, USA. ACM.

[Manning et al. 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Nataraj et al. 2011] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, pages 4:1–4:7, New York, NY, USA. ACM.

[Pedregosa et al. 2011] Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pietrek 1994] Pietrek, M. (1994). Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. https://msdn.microsoft.com/en-us/library/ms809762.aspx, acessado em novembro de 2016.

[Roberto Jordaney and Cavallaro 2016] Roberto Jordaney, Zhi Wang, D. P. I. N. and Cavallaro, L. (2016). Misleading metrics: On evaluating machine learning for malware with confidence. *Technical Report 2016-1 — Royal Holloway, University of London*.

[Roy et al. 2015] Roy, S., DeLoach, J., Li, Y., Herndon, N., Caragea, D., Ou, X., Ranganath, V. P., Li, H., and Guevara, N. (2015). Experimental study with real-world data for android app security analysis using machine learning. In *ACSAC'15*.

[Schultz et al. 2001] Schultz, M. G., Eskin, E., Zadok, E., and Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, SP '01, pages 38–, Washington, DC, USA. IEEE Computer Society.

[Singh et al. 2012] Singh, A., Walenstein, A., and Lakhotia, A. (2012). Tracking concept drift in malware families. In *AISec'12*.

[Slashdot Media 2016] Slashdot Media (2016). SourceForge. https://sourceforge.net/.

[Softonic Internacional S.A. 2016] Softonic Internacional S.A. (2016). Softonic. https://en.softonic.com/.

 © 2017 Sociedade Brasileira de Computação

Dizang : Uma solução para coleta de evidências forenses de ataques de injeção na nuvem

Hamilton J. S. Fonte II, Marcos A. Simplicio Jr.

[1]Escola Politécnica – Universidade de São Paulo (USP)
Av. Prof. Luciano Gualberto 380 - Butantã – 05.508-010 – São Paulo – SP – Brasil

Abstract. *Cloud architectures are becoming more common, and so are the number of security issues surrounding this technology. Unfortunately, due to the volatile nature of cloud resources, the task of gathering evidence for forensic analysis in this environment runs into practical and legal challenges. This paper analyzes proposals aimed at addressing existing challenges when collecting evidence in the cloud, discuss their limitations, and then presents a solution to overcome them. Specifically, the proposal focuses on the reproducibility of the collection process, without violating jurisdictions or the privacy of those not involved in the investigation.*

Resumo. *Arquiteturas em nuvem são cada vez mais comuns, e também o número de problemas de segurança envolvendo essa tecnologia. Infelizmente, devido à natureza volátil de recursos na nuvem, a coleta de evidências para análise forense nesse ambiente tem esbarrado em desafios práticos e legais. Este trabalho analisa propostas voltadas a resolver os desafios existentes na coleta evidências na nuvem, discute suas limitações, e então apresenta uma solução visando suplantá-las. Especificamente, a proposta tem como foco a reprodutibilidade do processo de coleta, sem com isso violar jurisdições ou a privacidade dos não envolvidos na investigação.*

1. Introdução

Técnicas de virtualização, replicação de serviços e compartilhamento de recursos entre múltiplos usuários (multi-inquilinato) dão a nuvens computacionais alta escalabilidade [Morsy et al. 2010]. Porém, tais mecanismos também levam à volatilidade dos recursos virtuais que executam aplicações. Afinal, quando submetida a uma carga elevada, uma aplicação hospedada na nuvem pode criar clones das máquinas virtuais (*virtual machines* – VMs) que a hospedam e balancear a carga entre elas, buscando atender à demanda sem prejuízos na qualidade do serviço oferecido. Após esse pico, as VMs clonadas são normalmente desativadas, seus recursos liberados e o sistema retorna à capacidade anterior, evitando-se custos desnecessários.

Embora interessante do ponto de vista de eficiência e custos, do ponto de vista forense a volatilidade da nuvem traz problemas em caso de ataques. Por exemplo, caso uma das instâncias de processamento virtuais criadas temporariamente seja alvo de ameaças que atuam diretamente na sua memória, sem deixar rastros em discos (e.g., arquivos de *log*), as evidências desse evento podem ser completamente perdidas após elas serem desativadas e terem seus recursos liberados. Essa dificuldade é ainda agravada por aspectos como multi-inquilinato e multi-jurisdição típicas de soluções em nuvem [Gilbert and Sujeet 2008]. Especificamente, o aspecto multi-inquilino dificulta a

 © 2017 Sociedade Brasileira de Computação

obtenção do *hardware* que executa as aplicações de interesse, pois, como ele é compartilhado por vários usuários, removê-los para análise poderia levar a uma violação de privacidade dos usuários não relacionados à investigação. Já a natureza distribuída da nuvem pode levar à alocação de informações relevantes à investigação em vários países, dificultando a obtenção das mesmas em especial quando não existem acordos de cooperação entre as entidades envolvidas [Dykstra and Sherman 2012]. Combinadas, tais características dificultam a coleta de evidências com a credibilidade necessária para que elas possam ser usadas em processos legais, o que exige o respeito à privacidade, à jurisdição e à cadeia de custódia, bem como a reprodutibilidade do processo de coleta [Rahman and Khan 2015].

Embora existam soluções na literatura voltadas à coleta de informações na nuvem para a análise forense, a maioria delas aborda a coleta, o transporte e o armazenamento de forma isolada. Por exemplo, trabalhos como [Dykstra and Sherman 2013] e [Reichert et al. 2015] tratam de fatores como multi-inquilinato e multi-jurisdição, discutindo formas de coleta e preservação da evidência fora da nuvem. Já estudos como [George et al. 2012] se concentram na análise forense para a coleta de evidência de VMs enquanto elas estão em execução, enquanto trabalhos como [Sang 2013] tratam a garantia de cadeia de custódia durante o transporte da evidência. Por outro lado, não foram identificadas propostas que (1) descrevam como o dado é coletado e armazenado observando a cadeia de custódia, e (2) permitam que, mesmo que um recurso virtualizado seja desalocado, haja condições de se reproduzir o processo de coleta de evidências.

O presente trabalho visa suplantar tais limitações, apresentando uma proposta com foco: (1) na reprodutibilidade do processo de coleta, (2) no estabelecimento de vínculo entre a evidência coletada e sua origem, (3) na preservação da jurisdição e da privacidade dos não envolvidos na investigação. Em suma, a solução descrita provê uma forma de correlacionar evidências e sua origem virtual, permitindo transportar e armazenar tais dados de modo a preservar sua credibilidade. Para isso, supõe-se que o sistema sendo monitorado é executado dentro de um contêiner, abordagem que se justifica pelo crescimento da adoção de conteinerização nos últimos anos, bem como pela previsão de que este será o modelo de implementação mais usado no futuro [Piraghaj et al. 2015]. A solução tem como alvo específico ataques de injeção de código [Case et al. 2014], pois estes, quando usados contra uma arquitetura em nuvem, não deixam rastros quando recursos de processamento virtuais são desativados e sua memória é liberada [Vömel and Stüttgen 2013], [Case et al. 2014].

Este documento está assim organizado. A Seção 2 discute brevemente soluções em nuvem e suas características. A Seção 3 analisa os trabalhos relacionados na área de forense de memória. A Seção 4 detalha a solução proposta e avalia como ela trata os desafios alvo deste trabalho. A Seção 5 apresenta as conclusões do estudo.

2. Adoção de arquiteturas em nuvem e contêineres

Uma nuvem computacional é um modelo de infraestrutura no qual recursos compartilhados em quantidade configurável, acessíveis via rede, são alocados e desalocados com esforço mínimo de gerenciamento por parte de um provedor de serviços. [Mell and Grance 2011] Há três modelos principais de comercialização de uso da nuvem [Mell and Grance 2011]: *software* como serviço (*Software as a Service* – SaaS), na qual se provê o *software* que será usado pelo cliente; plataforma como serviço (*Platform as a*

 © 2017 Sociedade Brasileira de Computação

Service – PaaS), na qual se provê o ambiente para que o cliente desenvolva, teste e execute seu *software*; e, o tipo mais pertinente para este trabalho, Infraestrutura como serviço (*Infrastructure as a Service* – IaaS), na qual são fornecidos recursos computacionais básicos, como processamento e memória, em geral de forma virtualizada.

A virtualização de recursos na nuvem, embora tradicionalmente feita por meio de VMs, vem sendo crescentemente feita também na forma de contêineres. De fato, segundo estudo realizado em 2016 com 235 empresas que têm desenvolvimento de software como sua atividade fim ou como suporte à atividade fim [DevOps and ClusterHQ 2016], 76% dos respondentes utilizam contêineres para melhorar a eficiência do processo de desenvolvimento e em suas arquiteturas de micro-serviços em nuvem. Diferentemente de VMs, que envolvem a criação de um *hardware* virtual e também de um SO acima do sistema nativo, a virtualização com contêineres é feita no nível do SO nativo, tem uma implementação mais simples eliminado camadas entre o aplicativo executado e o *hardware* físico. Uma tecnologia bastante utilizada para esse propósito são Contêineres Linux (LXC), que aproveitam-se de funcionalidades como cgroups e namespacing do kernel do Linux para auxiliar no gerenciamento e isolamento de recursos virtuais.

3. Trabalhos relacionados

Existem vários aspectos relativos à análise forense na nuvem, indo desde a coleta de informações até a garantia da cadeia de custódia de evidências. Para uma discussão mais estruturada dos trabalhos disponíveis na literatura sobre o tema, resumidos na Tabela 1, a seguir eles são apresentados com base nos diferentes aspectos que abordam.

Tabela 1. Comparativo de soluções de coleta de informações de memória de máquinas em nuvem para análise forense

	Dizang (esta proposta)	[George et al. 2012]	[Poisel et al. 2013]	[Dykstra and Sherman 2013]	[Barbara 2014]	[Reichert et al. 2015]	[Sang 2013]	[Dolan-Gavitt et al. 2011]	[Aljaedi et al. 2011]	[Dezfouli et al. 2012]	[van Baar et al. 2014]
Coleta é contínua?	✓	✘	✘	✘	✘	✘	✓	✘	✘	✓	✓
Reproduz o processo sem a VM?	✓	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
Garante cadeia de custódia?	✓	✘	✘	✘	✘	✓	✓	✘	✘	✘	✓
Preserva jurisdição e privacidade?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

3.1. Acessar e coletar as informações de memória na nuvem

Diversos trabalhos de análise forense na nuvem se concentram na coleta de dados "após o fato", ou seja, após a intrusão ser detectada [Reichert et al. 2015, Poisel et al. 2013, Dykstra and Sherman 2013, George et al. 2012, Sang 2013]. Os processos de coleta descritos nesses trabalhos podem ser iniciados de forma manual ou automaticamente, via integração com um mecanismo de detecção de intrusão. No caso específico de memória

© 2017 Sociedade Brasileira de Computação

volátil, tal forma de coleta não consegue descrever como era a memória antes da intrusão, pois o processo só é acionado depois da detecção do ataque. Tal limitação pode trazer prejuízos à investigação, dado que algumas análises dependem exatamente da capacidade de se comparar dois momentos da memória [Case et al. 2014]. Entre os trabalhos estudados, a única proposta encontrada que leva tal necessidade em consideração é [Dezfouli et al. 2012], que propõe que o dado seja armazenado no próprio equipamento sob análise. Infelizmente, entretanto, a aplicação de tal abordagem no cenário em nuvem é pouco viável, pois pode levar à perda de informações importantes caso a VM ou contêiner seja desativada, tendo seus recursos liberados.

Existem ainda trabalhos voltados à coleta de informações durante a execução do sistema, nos quais os dados são constantemente coletados sem distinção do que aconteceu antes ou depois do fato de interesse. Esse é o caso de trabalhos como [Poisel et al. 2013, Dykstra and Sherman 2013, Sang 2013], que adotam a estratégia de isolar e parar a VM para em seguida realizar o processo de coleta. Embora interessantes, as abordagens descritas nesses trabalhos podem levar a um elevado volume de dados coletados, além de também não tratarem o cenário em que é necessário coletar evidências quando os recursos virtuais contendo tais informações são liberados.

3.2. Capacidade de reproduzir o processo e obter os mesmos resultados

Se, durante uma análise forense, analistas diferentes obtêm resultados distintos ao executar o mesmo procedimento de coleta, a evidência gerada não tem credibilidade, inviabilizando seu uso em um processo legal. Por essa razão, a reprodutibilidade do processo de coleta é uma parte importante da geração de evidências para análise forense. Infelizmente, entretanto, nenhuma das propostas encontradas na literatura atualmente permite tal reprodutibilidade em cenários de nuvem em que VMs ou contêineres são desativados e seus recursos físicos liberados: todas elas dependem da existência do recurso virtual para a repetição do processo de coleta.

3.3. Não violar privacidade ou jurisdição das partes não envolvidas na investigação

Em um ambiente de nuvem pública, remover o *hardware* para análise posterior pode levar à violação de privacidade de usuários, uma vez que o multi-inquilinato desse cenário faz com que uma mesma máquina física guarde informações de diversos clientes, alguns dos quais podem não estar envolvidos na investigação em curso. Diversos trabalhos na literatura tratam esse problema adequadamente, por meio das duas estratégias principais: a primeira, adotada em [Reichert et al. 2015, George et al. 2012, Poisel et al. 2013, Dykstra and Sherman 2013], consiste em coletar dados pertinentes à investigação e armazená-los fora da nuvem; a segunda, empregada em [Sang 2013] e que constitui um caso específico de [George et al. 2012], depende da cooperação do provedor de serviços de nuvem para conseguir as informações necessárias à investigação. Depender do provedor de serviços de nuvem é uma estratégia pouco recomendada, entretanto, pois (1) o volume de dados de usuários pode forçar os provedores a limitar o tamanho dos *logs* armazenados, e (2) caso ocorra uma indisponibilidade causada por um ataque, o objetivo do provedor será o de restabelecer o serviço, não necessariamente o de preservar evidências[Alqahtany et al. 2015].

 © 2017 Sociedade Brasileira de Computação

4. Solução proposta: Dizang

A presente proposta tem como objetivo principal coletar memória de recursos computacionais virtuais de modo a conseguir: (1) identificar a fonte da evidência, mesmo se o recurso virtual não existir mais; (2) descrever o sistema antes e depois do incidente; (3) transportar e armazenar a memória coletada de uma forma que garanta sua integridade e confidencialidade; e (4) não violar a jurisdição e a privacidade de outros usuários que porventura tenham recursos alocados no mesmo servidor físico. A solução aqui apresentada, denominada Dizang , é descrita em detalhes a seguir.

4.1. Descrição

Em sistemas computacionais executados sobre uma infraestrutura física (i.e., não virtualizada), pode-se fazer uma associação direta entre um recurso qualquer, como uma informação da memória, imagem de disco ou pacotes trafegando na rede, e sua origem correspondente. Já em sistemas construídos sobre uma infraestrutura virtual, em especial quando esta é auto-escalável, os recursos computacionais são altamente voláteis e, portanto, podem ser desalocados a qualquer momento. Para conseguir correlacionar uma evidência a sua origem volátil, é necessário utilizar outro elemento em que persista a relação fonte-evidência, o que na presente proposta é feito por meio de contêineres linux (LXC). Embora um contêiner seja um *software* e, portanto, também volátil, cada imagem compilada e sua execução na forma de contêiner são normalmente atrelados a um *hash* que identifica univocamente essa relação. O contêiner também permite identificar com mais precisão a fonte de uma evidência, uma vez que é possível dividir as partes de um sistema em contêineres: por exemplo, um contêiner para o motor de páginas dinâmicas (e.g., Apache), outro com a lógica de negócios (e.g., *golang*) e um terceiro para um banco de dados (e.g., *Cassandra*).

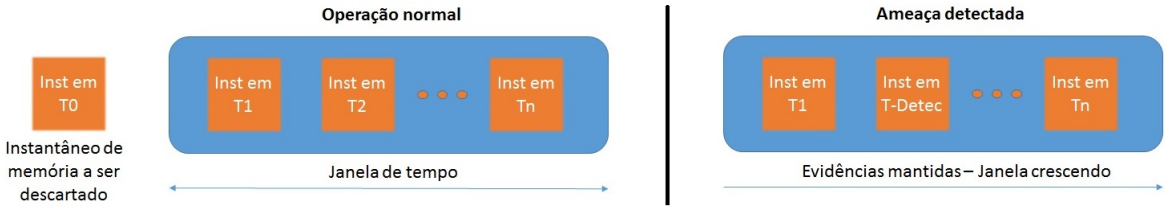
A cópia de memória não é uma atividade atômica, pois ela é executada em conjunto com outros processos. Portanto, caso um desses processos seja um código malicioso apagando traços de sua existência da memória do contêiner, informações possivelmente importantes para a investigação podem acabar sendo perdidas. Com o objetivo de deixar o processo de cópia da memória mais atômico, Dizang interrompe temporariamente a execução do contêiner, realiza a cópia de sua memória, e em seguida retoma sua execução. Essa técnica, que é semelhante àquela adotada em [Rafique and Khan 2013] para VMs, produz um instantâneo da memória volátil do contêiner; isso permite sua análise em um estado de repouso, ou seja, sem a necessidade de ter o contêiner em execução. Ao realizar a coleta em intervalos de tempo adequados, é possível construir um histórico do estado da memória durante a execução no contêiner.

A maioria das técnicas forenses mais usadas atualmente são voltadas à obtenção da informação em sua totalidade, seja via cópia bit a bit, seja por meio da obtenção do *hardware* físico [Simou et al. 2014] [Bem et al. 2008]. Embora tais técnicas possam parecer interessantes à primeira vista, elas muitas vezes acabam sendo responsáveis por um problema: o crescente volume de informações que os investigadores precisam analisar [Quick and Choo 2014]. Para mitigar essa dificuldade, em Dizang são adotadas duas estratégias: a primeira é a definição de um volume de dados que possa ser considerado *suficiente* para a realização de uma investigação; a segunda é a definição de uma *idade máxima* para a evidência enquanto o sistema trabalha em condições normais, isto

© 2017 Sociedade Brasileira de Computação

é, quando não está sob ataque. Para detectar e analisar intrusões na memória de processos, é necessário ter uma cópia da memória antes e depois da intrusão [Case et al. 2014]. Assim, a solução proposta implementa uma janela de instantâneos de memória cobrindo um intervalo de tempo pré-definido, como ilustrado na Fig. 1. Em condições normais de operação, as evidências são coletadas com certa periodicidade e coletas que atingem uma determinada idade são descartadas. Em contraste, após a detecção de um evento de ataque (e.g., por um sistema de detecção de intrusões), Dizang deixa de descartar as coletas mais antigas do *log* de monitoramento, sendo possível conhecer o sistema antes e depois do ataque e, assim, avaliar sua evolução.

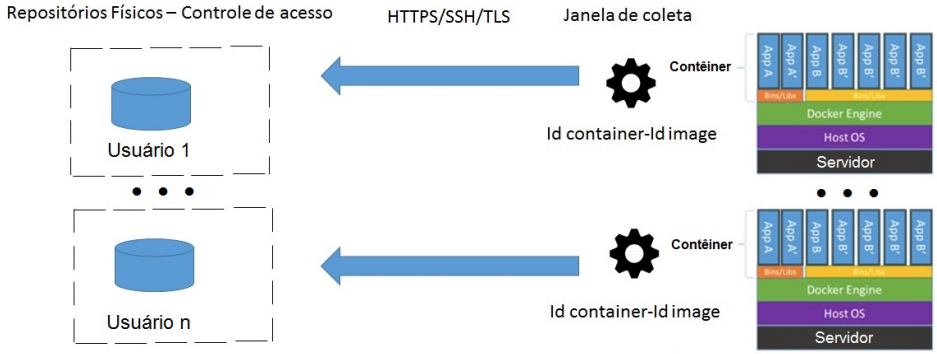
Figura 1. Janela deslizante de coleta de evidência



Para persistir a relação evidência-origem e garantir a integridade da mesma, a presente proposta calcula o hash do par [evidência, identificador da imagem do contêiner] e armazena a tripla [hash, identificador da imagem do contêiner, evidência]. Para evitar eventuais problemas com o armazenamento desses dados em países com jurisdições diferentes daquelas que devem ser aplicadas na investigação em questão, as evidências coletadas são armazenadas em um local físico fora da nuvem, após serem transportadas por meio de um canal seguro (e.g., via *Transport Layer Security* – TLS [Dierks T 2008]).

4.2. Implementação

Figura 2. Arquitetura geral da solução Dizang



Os métodos propostos foram implementados em uma plataforma de testes visando avaliar a eficácia de Dizang em coletar as informações de memória dos contêineres de forma reprodutível, sem violar jurisdições ou a privacidade de usuários. A solução, ilustrada na Fig. 2, consistiu na criação de 1 VM usando o Oracle Virtual Box 5.0em um notebook Intel i5 de 2.30Mhz e 4Gb de RAM com sistema operacional de 64 bits. Essas VMs possuem 2 Gb de memória RAM e emulam apenas 1 processador, e em cada uma

 © 2017 Sociedade Brasileira de Computação

delas foi instalado o Docker Engine 1.10e a API Docker 1.21, com os quais foram criados 3 contêineres executando o nginx 1.0em diferentes portas. Usando uma aplicação Java que descobre o identificador de processo associado a cada contêiner, pode-se copiar conteúdo do *descritor de alocação de memória não uniforme* (**/proc/pid/numa_maps**), o qual contém a alocação das páginas de memória, os nós que estão associados a essas páginas, o que está alocado e suas respectivas políticas de acesso [Unix Man Pages]. A cópia e gravação do arquivo é tal que, a cada minuto, a aplicação (1) pausa o contêiner em questão, (2) copia a diretório **numa_maps**, (3) concatena os dados obtidos com o *hash* de identificação da imagem do contêiner, (4) calcula o *hash* do conjunto e (5) salva o resultado em um arquivo cujo nome é o identificador da imagem do contêiner e a extensão é **.mem**. Após a conclusão do processo de cópia, a mesma aplicação verifica se existem arquivos **.mem** em disco mais antigos que um certo intervalo de tempo "t", descartando-os.

4.3. Resultados experimentais

Para avaliar a efetividade de Dizang na coleta de evidências, alguns experimentos foram realizados usando o ambiente implementado (descrito na Seção 4.2). Primeiramente, o sistema foi configurado para realizar coletas de memória em intervalos de 1 minuto, salvá-las em disco externo e apagar amostras coletadas há mais de 5 minutos. O sistema foi então executado por 30 minutos, tempo durante o qual foram coletadas como métricas (1) o uso de espaço em disco utilizado pelos instantâneos de memória salvos e (2) o tempo de pausa no contêiner necessário para a cópia das mesmas. A cada coleta, foi executado o comando `du -sh *.mem` do *Unix* no disco de armazenamento externo, para retornar a lista dos arquivos onde os instantâneos de memória foram armazenados e o espaço em disco ocupado pelos mesmos. Ao fim do experimento, os contêineres foram removidos.

A ocupação em disco devido aos instantâneos de memória capturados durante o experimento é mostrada no gráfico da Fig. 3. O gráfico mostra que o aumento do uso do espaço em disco é linear e o crescimento se interrompe quando é atingido o limite de tempo configurado para a janela, pois as coletas com tempo de vida maior que tal limite são apagadas do disco. Assim, que a solução mantém sob controle o espaço em disco ocupado pelas amostras coletadas. Ao mesmo tempo, instantâneos de memória salvos pela solução depois que os contêineres são removidos continuam no disco da máquina, podendo ser associados a sua origem (i.e., contêiner e imagem), conforme esperado para uma análise forense. Essa capacidade se mantém após a detecção de uma ameaça, pois nesse caso coletas mais antigas deixam de ser apagadas. Logo, é possível descrever o estado do sistema antes e depois do incidente [Case et al. 2014], permitindo-se, por exemplo, que ataques de injeção de código em memória sejam analisados.

Uma potencial limitação da solução proposta é que a pausa de um contêiner para coleta de dados poder, em princípio, causar perdas no desempenho da aplicação sendo executada. Para avaliar esse impacto, durante o experimento foram medidos os tempos de cópia da memória do contêiner. Os resultados são mostrados no gráfico da Fig. 4. É possível notar que, após a inicialização da aplicação, o tempo para realizar a cópia é bastante reduzido, variando entre 20 e 40 milisegundos. Em especial, para contêineres executando um motor de páginas web dinâmicas, como é o caso do experimento em questão, essa latência deve ser pouco perceptível por usuários finais.

© 2017 Sociedade Brasileira de Computação

Figura 3. Evolução do uso do espaço em disco com o Dizang .

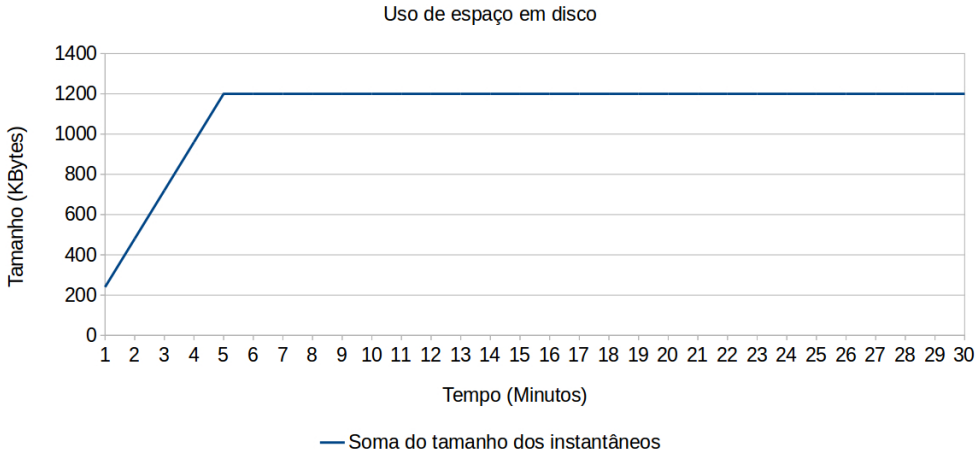
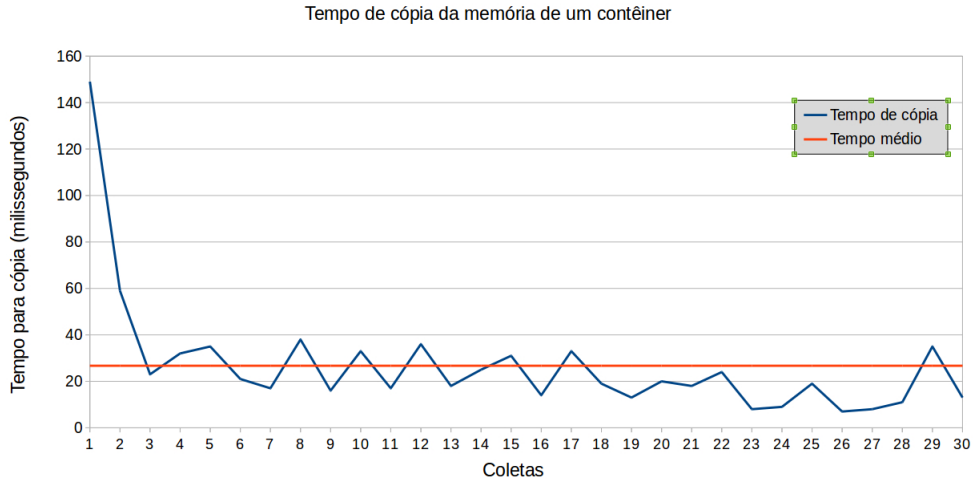


Figura 4. Tempo de cópia da memória de um contêiner



4.4. Limitações

Como a solução descrita tem como foco coletar informações de memória no espaço do usuário (*user space*), ela não consegue acessar o espaço de kernel (*kernel space*). Assim, Dizang em princípio não provê suporte a técnicas de investigação de malware que se baseiam em informações do *kernel space*, como, por exemplo, a comparação de informações do bloco do ambiente do processo (*Process Environment Block – PEB*), que ficam no *user space*, com informações do descritor de endereços de memória virtual (*Virtual Address Descriptor – VAD*), que fica no *kernel space*. Análise de ameaças que realizam manipulação direta dos objetos do kernel (D.K.O.M.– *Direct Kernel Object Manipulation*) também não se beneficiam com a solução aqui proposta.

5. Considerações Finais

Ameaças digitais que atuam diretamente na memória de sistema não costumam deixar rastros em disco após terem os recursos correspondentes desalocados, dificultando análises forenses posteriores. Esse problema é especialmente notável em sistemas de computação em nuvem, nos quais a alocação e desalocação de recursos virtualizados

 © 2017 Sociedade Brasileira de Computação

(e.g., VMs e contêineres) é frequente. Essa característica, aliada a aspectos como multi-inquilinato e multi-jurisdição de nuvens computacionais, dificulta a coleta de evidências para a investigação de incidentes.

Nesse cenário, a proposta apresentada visa relacionar o instantâneo de memória a sua origem, utilizando o *hash* calculado da imagem do contêiner como identificador da evidência armazenada. Para evitar uso excessivo de memória, a quantidade de dados armazenados usa uma janela de armazenamento, o que permite descrever a memória antes e depois de um ataque (e.g., de injeção de memória). Combinada com uma ferramenta para identificação de ameaças, essas características de Dizang o transformam em uma solução poderosa para prover evidências e, assim, viabilizar análises forenses na nuvem.

Como trabalho futuro, pretende-se estudar o impacto provocado pela pausa do contêiner no desempenho de um sistema em produção e realizar uma análise da memória dos contêiners deste mesmo sistema utilizando as coletas feitas por Dizang . Com esses resultados pretende-se implementar uma ferramenta mais flexível, que não precise ter acesso a memória completa da máquina, requisito comum de ferramentas de análise de malware atuais (e.g., FROST [Dykstra and Sherman 2013] e Volatility Framework [Volatility Foundation 2014]).

Referências

Aljaedi, A., Lindskog, D., Zavarsky, P., Ruhl, R., and Almari, F. (2011). Comparative analysis of volatile memory forensics: Live response vs. memory imaging. In *IEEE 3rd Int. Conf. on Privacy, Security, Risk and Trust*, pages 1253–1258.

Alqahtany, S., Clarke, N., Furnell, S., and Reich, C. (2015). Cloud forensics: A review of challenges, solutions and open problems. In *Int. Conference on Cloud Computing (ICCC)*, pages 1–9. IEEE.

Barbara, D. (2014). Desafios da perícia forense em um ambiente de computação nas nuvens. Technical report, Univ. do Planalto Catarinense. `revista.uniplac.net/ojs/index.php/tc_si/article/view/1911`.

Bem, D., Feld, F., Huebner, E., and Bem, O. (2008). Computer forensics - past , present and future. *Journal of Information Science and Technology*, 5(3):43–59.

Case, A., Ligh, M., Jamie, L., and Walters, A. (2014). *The Art of Memory Forensics: Detecting malware and threats in Windows, Linux and Mac memory*. Wiley.

DevOps and ClusterHQ (2016). Container market adoption survey 2016. `https://clusterhq.com/assets/pdfs/state-of-container-usage-june-2016.pdf`.

Dezfouli, F., Dehghantanha, A., Mahmoud, R., Sani, N., and Shamsuddin, S. (2012). Volatile memory acquisition using backup for forensic investigation. In *Int. Conf. on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, pages 186–189.

Dierks T, R. E. (2008). The Transport Layer Security (TLS) Protocol. IETF: `https://tools.ietf.org/html/rfc5246`.

Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., and Lee, W. (2011). Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *IEEE Symposium on Security and Privacy*, pages 297–312.

© 2017 Sociedade Brasileira de Computação

Dykstra, J. and Sherman, A. (2012). Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation*, 9:S90–S98. (Proc. of the 12th Annual DFRWS Conference).

Dykstra, J. and Sherman, A. T. (2013). Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform. *Digital Investigation*, 10:S87–S95. (Proc. of 13th Annual DFRWS Conference).

George, S., Venter, H., and Thomas, F. (2012). Digital Forensic Framework for a Cloud Environment. In *IST Africa*, pages 1–8, Tanzania. IIMC.

Gilbert, P. and Sujeet, S. (2008). *Advances in Digital Forensics IV*, volume 1. Springer-US, Orlando, 1 edition.

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. NIST SP 800-145. `csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`.

Morsy, A. M., Grundy, J., and Muller, I. (2010). An Analysis of the Cloud Computing Security Problem. In *APSEC Cloud Workshop*, Sydney, Australia.

Piraghaj, S., Dastjerdi, A., Calheiros, R., and Buyya, R. (2015). A framework and algorithm for energy efficient container consolidation in cloud data centers. In *IEEE Int. Conf. on Data Science and Data Intensive Systems (DSDIS)*, pages 368–375.

Poisel, R., Malzer, E., and Tjoa, S. (2013). Evidence and cloud computing: The virtual machine introspection approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 4(1):135–152.

Quick, D. and Choo, K. K. R. (2014). Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, 11(4):273–294.

Rafique, M. and Khan, M. N. A. (2013). Exploring Static and Live Digital Forensics: Methods, Practices and Tools. *IJSER*, 4(10):1048–1056.

Rahman, S. and Khan, M. N. A. (2015). Review of live forensic analysis techniques. *International Journal of Hybrid Information Technology*, 8(2):379–388.

Reichert, Z., Richards, K., and Yoshigoe, K. (2015). Automated forensic data acquisition in the cloud. *IEEE Int. Conf. on Mobile Ad Hoc and Sensor Systems*, pages 725–730.

Sang, T. (2013). A log-based approach to make digital forensics easier on cloud computing. *Intelligent System Design and Engineering Applications (ISDEA)*, pages 91–94.

Simou, S., Kalloniatis, C., Kavakli, E., and Gritzalis, S. (2014). Cloud forensics: Identifying the major issues and challenges. In *Advanced Information Systems Engineering (CAiSE 2014)*, volume 8484, pages 271–284.

Unix Man Pages. Numa Maps - Non Uniform Memory Architecture. `man7.org/linux/man-pages/man7/numa.7.html`.

van Baar, R. B., van Beek, H. M. A., and van Eijk, E. J. (2014). Digital Forensics as a Service: A game changer. *Digital Investigation*, 11:S54–S62.

Volatility Foundation (2014). Volatility Framework. `www.volatilityfoundation.org/`.

Vömel, S. and Stüttgen, J. (2013). An evaluation platform for forensic memory acquisition software. *Digit. Investig.*, 10:S30–S40. Elsevier Science Publishers.

 © 2017 Sociedade Brasileira de Computação

ForensicWork: Desenvolvimento de um *Framework* para Perícia Forense em Computação na Nuvem

Cátia Oliveira de Jesus, Mailson Sousa Couto

Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA)
Av. Amazonas, 3150 - Zabelê, Vitória da Conquista - BA, 45030-220 – Brasil

catia.infor@gmail.com, mailsoncouto@ifba.edu.br

Abstract. *When a crime is practiced through cloud service platforms, the police investigation finds serious obstacles that could compromise the elucidation of the case and even the culpability of the culprits, given the difficulty in locating the evidence of the crime, which may have been committed in different countries, and disseminated on several servers. Given this type of situation, which is increasingly present in modern and computerized society, this article proposes to develop a conceptual framework that allows the exploration of data stored in cloud services in a timely manner, respecting the legislation of the different jurisdictions involved and the chain of custody, so as to systematize the investigative process in the virtual environment.*

Resumo. *Quando um delito é praticado por intermédio de plataformas de serviços em nuvem, a investigação policial encontra sérios entraves que podem comprometer a elucidação do caso e inclusive responsabilização dos culpados, dada a dificuldade em se localizar as provas do ilícito, que pode ter sido cometido em diferentes países, e disseminado em diversos servidores. Tendo em conta esse tipo de situação, cada vez mais presente na sociedade moderna e informatizada, esse artigo se propõe a desenvolver um framework conceitual que possibilite a exploração de dados armazenados em serviços em nuvem em tempo hábil, respeitando a legislação das diferentes jurisdições envolvidas e a cadeia de custódia, de modo a sistematizar o processo investigatório no ambiente virtual.*

1. Introdução

A evolução da tecnologia e o crescimento exponencial do número de pessoas e dispositivos conectados à Internet em todo o mundo, têm tornado a *web* um ambiente altamente propício para a execução de práticas criminosas. Neste cenário, surgiu a Perícia Forense Computacional. Segundo [Sá 2011], a palavra Perícia é proveniente do latim *Peritia*, cujo significado é conhecimento ou experiência, podendo também ser definida como um instrumento especial de constatação, prova ou demonstração científica da veracidade de situações, coisas ou fatos [Alberto 2012]. Assim, a perícia forense computacional pode ser entendida como o processo de identificar, preservar, analisar e apresentar provas digitais relativas a um determinado delito de forma legal e aceitável perante um tribunal de justiça.

 © 2017 Sociedade Brasileira de Computação

Um processo de investigação criminal deve considerar a utilização de métodos e protocolos bem definidos, visando à análise eficaz e ampla aceitação na esfera jurídica. Em uma investigação forense, as provas digitais podem ser extraídas de diferentes fontes, como discos rígidos, memórias, sistemas operacionais, dispositivos móveis e no ambiente de rede.

Todas as modalidades de perícia citadas no parágrafo anterior já possuem metodologias de coleta, extração e análise bem definidas, que são formalmente aceitáveis nos tribunais e amplamente difundidas entre os profissionais técnicos que realizam tais procedimentos, ao contrário da perícia em nuvem. Sob esta perspectiva, este trabalho desenvolve um estudo para apresentar os desafios, e os avanços da investigação forense no âmbito da computação em nuvem, esboçando um panorama geral do andamento das investigações criminais nesta esfera e apresentando um modelo conceitual de *framework* que permita sistematizar tal processo.

2. Revisão Teórica

A computação em nuvem surgiu da ideia de se utilizar, em qualquer lugar e independentemente de plataforma, diversas aplicações armazenadas na Internet, sem a necessidade de instalação prévia no dispositivo em que se deseja utilizar tal produto. De acordo com [Hurwitz et al. 2010], a nuvem pode ser entendida como um agrupamento de módulos processadores que oferecem serviços e conexões, permitindo assim a entrega da computação como um serviço. Aplicações em nuvem incluem a oferta de *software*, infraestrutura e armazenamento, tudo isso através da Internet, seja com componentes separados, seja uma plataforma completa, com base na demanda do usuário.

2.1. Computação Forense na Nuvem

A computação forense é definida por [Galvão 2009] como a ciência que trata de todos os processos de investigação forense e análise de incidentes que ocorreram por meio de dispositivos computacionais, contribuindo assim com a identificação e comprovação da autoria de tais delitos. De acordo com [Reis et al 2004] a computação forense visa realizar uma investigação com o maior nível de transparência possível, sendo então necessário que o profissional responsável pela perícia seja muito cuidadoso com todas as possíveis evidências, para que estas possam se tornar provas confiáveis perante a justiça.

Antes, a maioria das atividades de computação era realizada de forma local, na própria máquina, porém, é notável a tendência de migração destas atividades para a nuvem, sejam elas lícitas ou ilícitas. Com o fenômeno da virtualização, a forense computacional teve de se adaptar às peculiaridades destes ambientes virtualizados, quando um perito tem o desafio de realizar a análise forense em alguma plataforma da nuvem, ele esbarra em diversas questões técnicas e jurídicas, que em muitas vezes inviabilizam até o serviço.

Além das questões técnicas desta tecnologia, como criptografia, formatos de dados desconhecidos e descentralização do armazenamento, é preciso lidar com as questões jurídicas de cada país. [Reilly 2010] questiona a falta de ferramentas específicas para auxiliar os especialistas com investigações na nuvem, e afirma que esta

 © 2017 Sociedade Brasileira de Computação

escassez de instrumentos acaba inviabilizando muitas perícias. Para [Gellman 2009], por se tratar de uma tecnologia distribuída geograficamente entre diferentes nações, é preciso que os peritos atuem de forma cordial com a justiça, selando acordos nacionais e internacionais, para que se respeitem tanto as jurisdições de cada Estado, quanto às questões de privacidade dos usuários. Neste contexto, [Reilly 2010] chama a atenção para a falta de material que indique como devem ser executados os processos de investigação nos ambientes em nuvem, e conclui sinalizando a necessidade de desenvolvimento de novas metodologias ou aperfeiçoamento das existentes.

3. Trabalhos Relacionados

Diante de toda a relevância que o tema em questão representa, outros *frameworks* em computação em nuvem já foram propostos por pesquisadores e, assim, contribuíram bastante com o desenvolvimento deste projeto. A seguir será apresentado um resumo de alguns destes trabalhos.

[Karie et. al. 2013] abordam as principais dificuldades encontradas no processo da investigação forense na nuvem, devido a sua infraestrutura complexa. Neste sentido, eles propuseram um quadro conceitual que apresenta os diferentes componentes essenciais da computação compartilhada, como os serviços e os provedores. Os autores afirmam que o quadro proposto expõe de forma clara os processos envolvidos e a estrutura da nuvem, reduzindo assim a complexidade do entendimento de tal tecnologia, e isso ajuda os pesquisadores a compreender os procedimentos e técnicas que precisam ser adotados em uma investigação, bem como auxilia a concepção de novas ferramentas para automatizar ou melhorar uma perícia nesse tipo de ambiente.

[Sibiya et al. 2012] desenvolveram um *framework* denominado *Live Digital Forensic Framework for a Cloud* (LDF2C), projetado para resolver o problema da associação dos dados armazenados na nuvem ao usuário que de fato é o proprietário, além de indicar mecanismos que podem ser utilizados para a coleta de evidências em aplicações virtualizadas. O LDF2C é implementado em um servidor virtual e possui um algoritmo utilizado para descobrir arquivos de *log* de locais considerados relevantes, como a área de trabalho remota hospedada ou o dispositivo cliente acessado. Quando acionado, ele executa a análise forense entre duas organizações virtuais e monitora a interação.

Outro trabalho relevante é o de [Mohan 2016], que a abordou os problemas que ocorrem em investigações cujos dados são dinâmicos e distribuídos em diversos servidores espalhados pelo mundo. O autor sugere um modelo baseado em *logs* para ajudar a reduzir a complexidade da perícia, permitindo coletar rapidamente informações na nuvem através dos registros de *logs*.

4. ForensicWork

Para [Wangler e Paheerathan 2000], os *frameworks* podem ser classificados em duas categorias básicas: vertical e horizontal. Os verticais são aqueles cuja atuação envolve a resolução de problemas de escopo bem definido e especificado. Já os horizontais são mais genéricos e possuem uma atuação ampla, não existindo um domínio específico. Sendo assim, o *framework* conceitual proposto neste artigo pode ser classificado como

© 2017 Sociedade Brasileira de Computação

vertical, pois o objetivo é ser aplicado exclusivamente no âmbito de investigações forenses em nuvem.

A Figura 1 ilustra a modelagem inicial do *framework* que, por enquanto, é composto por seis etapas básicas, sendo estas:

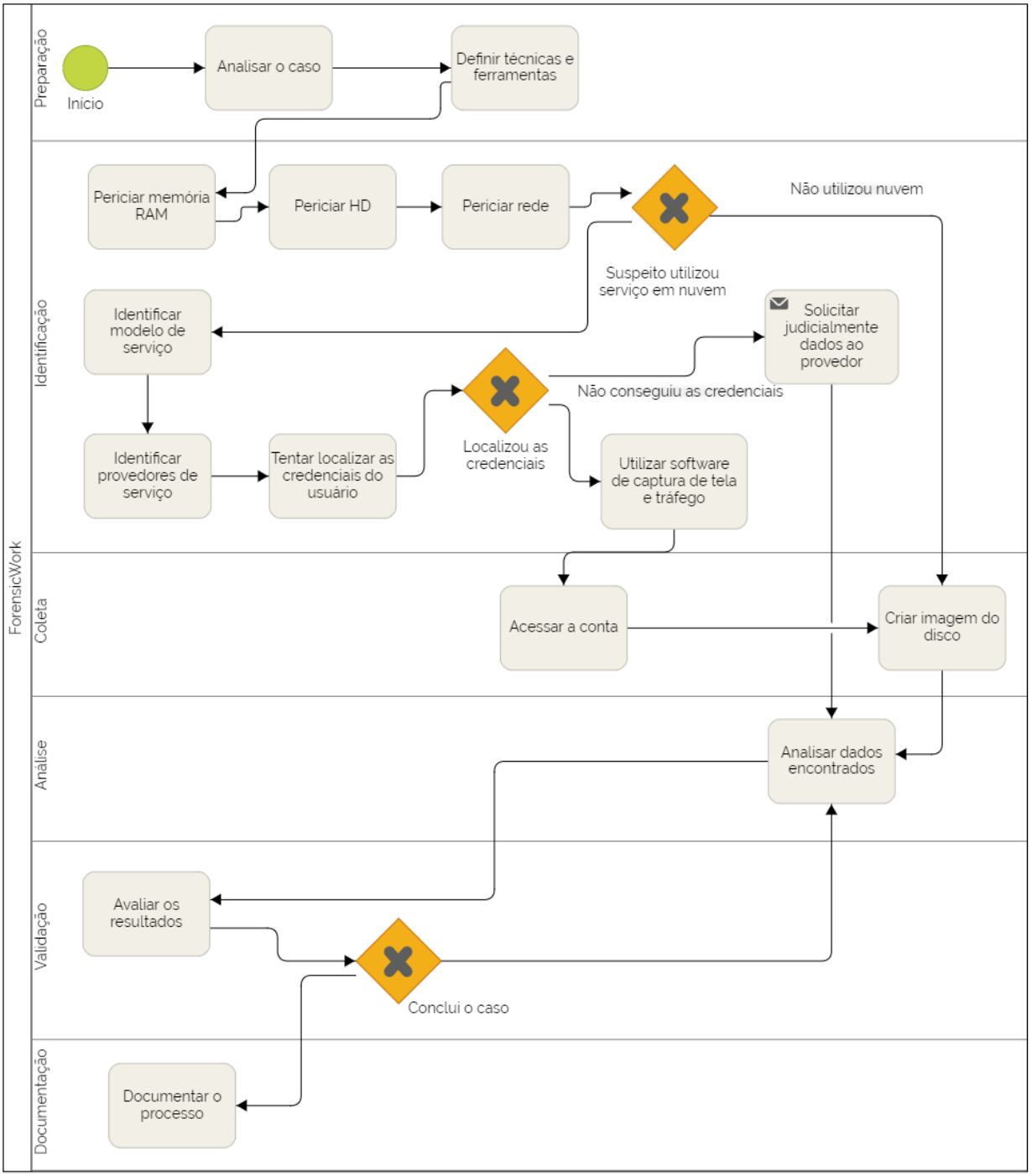


Figura 1. Fluxo de atividades do *framework* proposto.

- Preparação: é feita uma análise do caso para determinar quais tipos de dispositivos serão periciados e quais as ferramentas e *softwares* que poderão ser utilizados;

© 2017 Sociedade Brasileira de Computação

- Identificação: são realizadas as perícias convencionais na memória RAM e no armazenamento local do dispositivo, bem como na rede, em busca de dados que indiquem a utilização de algum serviço virtualizado ou dados importantes para a investigação. Caso haja indícios do uso de algum serviço em nuvem, tenta-se definir qual foi o modelo de serviço, qual o provedor, e possíveis credencias de acesso. De posse das credenciais, deve-se utilizar *softwares* para captura de tela e análise de tráfego da rede. Caso não se consiga o acesso a conta, uma solicitação judicial deve ser emitida ao provedor em questão.
- Coleta: caso a conta do suspeito tenha sido acessada, os dados devem ser baixados para perícia, e devem ser realizadas imagens do disco do dispositivo para posterior análise.
- Análise: algumas análises deverão ser realizadas em cima dos dados coletados.
- Validação: caso os dados encontrados sejam suficientes para concluir o processo, passa-se para a próxima etapa.
- Documentação: é feita a documentação com base na cadeia de custódia.

5. Conclusões e Trabalhos Futuros

O armazenamento de dados em nuvem tem representado um grande desafio para os profissionais da computação forense. A computação em nuvem oferece diferentes tipos de serviços e cada um deles possibilita a prática de atividades criminosas. No processo de investigação, as principais dificuldades enfrentadas pelos peritos estão nas fases de identificação e levantamento das provas.

Após ser determinado que uma conta de armazenamento em nuvem possui provas relevantes para um inquérito, o examinador deve comunicar judicialmente aos prestadores de serviços envolvidos para que a conta daquele usuário seja periciada e que seus dados não possam ser alterados. Além disso, outros dados podem ser obtidos através da aquisição dos arquivos de *log* do *software* cliente, arquivos de captura de tráfego de rede e captura dos dados disponíveis na memória do dispositivo.

Ainda em fase inicial, esta pesquisa visa contribuir com a melhora da compreensão do processo investigatório de crimes cometidos na nuvem, bem como oferecer aos peritos um *framework* para sistematizar o processo investigatório neste ambiente. Nas próximas etapas deste trabalho, pretende-se detalhar mais o escopo do *framework*, deixando os processos mais bem definidos, além da inclusão da fase de testes, definindo-se as ferramentas e *softwares* específicos para a validação do mesmo.

Referências

Alberto, Valder Luiz Palombo. (2012). Perícia Contábil. São Paulo: Editora Atlas, 5.

Galvão, Ricardo Kléber (2009). Perícia Forense Computacional SegInfo 2009 :: IV Workshop de Segurança da Informação. http://grsecurity.com.br/apostilas/forense/seginfo2009_2_rk.pdf

© 2017 Sociedade Brasileira de Computação

Gellman, R. (2009). Privacy in the Clouds: Risks to Privacy and Confidentiality from Cloud Computing. World Privacy Forum.
http://www.worldprivacyforum.org/pdf/WPF_Cloud_Privacy_Report.pdf

Hurwitz, Judith; bloor, Robin; Kaufman, Marcia. (2010). Cloud Computing For Dummies. HPSpecial Edition. Indianapolis: Wiley Publishing.

Mohan, Aswathy. (2016), A Log-Based Approach to Make Digital Forensic Easier on Cloud Computing. International Research Journal of Latest Trends in Engineering and Technology (IRJLTET).

N.M. Karie and H.S. Venter, "An Ontological Framework for a Cloud Forensic Environment," in the Proceedings of the European.

Reilly, D.; Wren, C.; Berry, T. (2010). Cloud Computing: Forensic Challenges for Law Enforcement. International Conference for Internet Technology and Secured Transactions (ICITST). p. 1-7.

Reis, Marcelo Abdalla dos, Geus, Paulo Lácio de. (2004) Análise Forense de Intrusões em Sistemas Computacionais: Técnicas, Procedimentos e Ferramentas. Instituto de Computação - Universidade Estadual de Campinas.

Sá, Antonio Lopes de.(2011) Perícia Contábil. São Paulo: Editora Atlas, 10. Ed.

Sibiya, G., Venter, H. S., & Fogwill, T. (2012). Digital forensic framework for a cloud environment. IIMC International Information Management Corporation.

Wangler, B. and S.J. Paheerathan (2000), Horizontal and Vertical Integration of Organizational IT Systems, in Information Systems Engineering: State of the Art and Research Themes, (S. Brinkkemper, E. Lindencrona and A. Solvberg (eds.)), Springer.

 © 2017 Sociedade Brasileira de Computação

WGID - VII Workshop de Gestão de Identidades Digitais
Artigos completos

© 2017 Sociedade Brasileira de Computação

Supporting the Self-adaptation of Authorization Infrastructures

Welkson Renny de Medeiros[1], **José Diego Saraiva**[1], **Carlos Eduardo da Silva**[2]

[1]Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN)
Natal – RN – Brazil

[2]Digital Metropolis Institute
Federal University of Rio Grande do Norte (UFRN)
Natal – RN – Brazil

{welkson.medeiros,diego.saraiva}@ifrn.edu.br, kaduardo@imd.ufrn.br

Abstract. *Insiders pose a great threat to organizations due to their capacity of exploiting privileged access for inappropriate gain. Traditional access control solutions are not able to deal with insiders, and some solutions apply concepts of self-adaptation to handle such problems. Existing work has been focused on detecting or how to respond to a detected insider. However, in order to allow the dynamic adaptation of access control policies, it is necessary to clearly specify what modification actions can be applied to a policy. Such actions can then be used for the definition of adaptation plans. Thus, this paper describes a generic Application Programming Interface (API) for manipulating access control policy based on Attribute-Based Access Control (ABAC). Our API follows a functional specification of ABAC, and aims to abstract away implementation details of access control engines, providing an effector that can be integrated into a self-adaptation approach.*

1. Introduction

Insider threats arise when trusted users of an information system can exploit their access permissions to compromise the confidentiality, integrity or availability of an organization's information assets [Cappelli et al. 2012]. These trusted users might include employees, contractors and business partners who can cause harm intentionally (e.g. for personal gain or revenge) or through error (e.g. due to negligence or insufficient training). Insider threats have been causing severe financial and reputational loss [Legg et al. 2015] [Cole 2015]. For example, in 2011, Kweku Adoboli worked at the Swiss Bank UBS when he was arrested on suspicion of making unauthorized transactions, leading the bank to have losses of approximately 2 billion of dollars [Yaseen 2016].

Software systems usually employ access control mechanisms to limit the actions that a legitimate user of a system can perform. These mechanisms, in general, implement an access control model based on roles (Role-Based Access Control - RBAC) [ANSI 2004] or based on attributes (Attribute-Based Access Control - ABAC) [Hu et al. 2013], where access control policies capture a set of rules defining permissions to perform operations on information assets that are associated with roles, or attributes, respectively. In this way, access control mechanisms, such as RBAC/ABAC models, deny requests that are in disagreement with the access control policy.

 © 2017 Sociedade Brasileira de Computação

However, these mechanisms are not able to detect abusive behaviour, such as a large number of requests in a short period of time. Such anomaly could be detected by relying on human administrators to monitor audit logs. However, this approach is impractical, representing a key security problem. In this context, automated mitigation systems that can adapt and update themselves according to the surrounding environment is crucial to current authorization infrastructures.

Self-adaptation is an effective approach in dealing with the dynamically changing of many application domains, including user access control mechanisms. A self-adaptive system is able to modify its own behavior and/or structure during run time in response to changes in its environment, requirements, or the system itself [Yuan et al. 2014]. Self-adaptation can be applied to existing access control mechanisms to improve upon the handling of insider threat, through dynamic adaptation of access control policies and user privileges [Bailey et al. 2011] [da Silva et al. 2017]. Such systems are able to observe and analyze access control requests and decisions logging for identifying when an adaptation is necessary, generating a plan to deal with the detected situation by means of dynamic changing access control policies.

To effectively implement dynamic access control policies through self-adaptation, it is necessary to define the basic operations that will be made available for changing access control policies. Based on these operations, adaptation plans can then be generated and verified before being applied to the system, in order to maintain the consistency of the access control mechanisms. In this context, this paper aims to provide a product-agnostic API for manipulation of access control policies based on the functional specifications of the RBAC [ANSI 2004] and ABAC [Gouglidis et al. 2017] models. Our API acts as an effector for a self-adaptive controller, abstracting away both the controller and the underlying authorization server in place on the controlled system. A prototype has been developed and integrated into two authorization servers for demonstrating the feasibility of our approach.

The rest of the paper is organized as follows. Section 2 presents background information and contextualization for this work. Section 3 presents the API proposal, and describes the implementation. Section 4 introduces a real case study used to illustrate and evaluate our approach. Section 5 compares our approach with related research, and Section 6 concludes the paper with a brief summary.

2. Background and Contextualization

The context in which this proposal was evaluated, and the tools and access control models involved will be presented in the following sections.

2.1. Self-adaptive System and Self-protection

Self-adaptation has been identified as a potential solution for improving traditional authorization management [Bailey et al. 2011] [da Silva et al. 2017]. Applying self-adaptive techniques enables the infrastructure to observe, reason, and act on its the own configuration of access control. Figure 1 presents a general view of the components of a self-adaptive system concerned with self-protection capabilities. The meta-level subsystem, usually called *autonomic manager* or *controller*, is responsible for monitoring and controlling a protected subsystem (at the base-level). The activities of the controller are organized according to the MAPE-K feedback control loop, in which a *monitoring* component

 © 2017 Sociedade Brasileira de Computação

collects information that is analyzed to identify the need for adaptation. A *Planning* component then synthesizes a plan for modifying the underlying protected subsystem, which is performed by an *execution* component. All components share knowledge through the *Knowledge* component. Autonomic managers communicate with the protected subsystem through *sensor* and *effector* interfaces [Yuan et al. 2014]. *Sensors*, often called probes or gauges, allow the collection of information about the managed element, while *effectors* provide a set of operations for carrying out changes to the base-level.

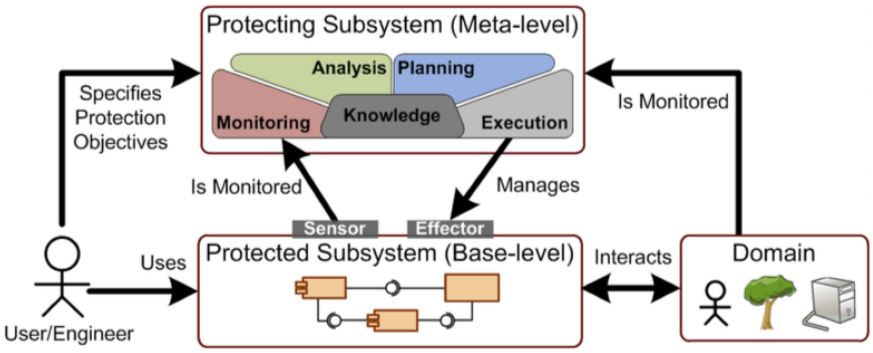


Figure 1. MAPE-K model for self-adaptation based on IBM reference architecture

In our context, the *base-level* is composed by the authorization infrastructure, and thus, part of the operations defined by the effector interface must support the modification of access control policies. Since these operations are the base for the adaptation plans generated by autonomic managers, it is important that they are very well defined and with clear semantics. In this sense, the functional specifications of RBAC/ABAC models, which have been defined though a formal notation, constitute an ideal definition for the effector operations related to the management of access control, allowing the provision of assurances about generated adaptation plans. However, as previously mentioned, authorization servers often do not follow these specifications. Thus the motivation for our work, which aims at defining an effector for authorization infrastructure based on the functional specification of the RBAC/ABAC model, allowing an autonomic manager to abstract away from specific characteristics of the authorization infrastructure.

2.2. Access Control Models and Mechanisms

As previously mentioned, the Role-Based Access Control (RBAC) model [ANSI 2004] works by associating permissions with roles, and roles to users. The term role is usually related to job functions within the context of an organization, while a permission is the approval to perform operations in a particular protected object. On the other hand, the Attribute Based Access Control (ABAC) model [Hu et al. 2013] is a method of access control where permissions are associated with attributes, in the key-value format, that are related to the subject (e.g., name, age), object (e.g., id, localization), action (e.g. GET, POST) or environmental condition (e.g., date/time, IP address, current threat level).

These different access control models rely on a set of components that implement the functionality of an access control mechanism [Hu et al. 2013]. As such, one of the components of an authorization infrastructure is a Policy Administration Point (PAP) which allows the manipulation of access control policies. Such manipulation should occur

 © 2017 Sociedade Brasileira de Computação

through well defined operations, as the means for providing assurances about a policy. In this way, the RBAC model [ANSI 2004] defines a functional specification in Z notation, including several *administrative operations* for handling access control policies, such as *addUser*, *addRole*, *assignUser*. On the other hand, the ABAC model [Hu et al. 2013] has not defined any formal specification of administrative operations. However, a proposal has been presented by [Gouglidis et al. 2017].

To create access control policies it is necessary to use a consistent language, and one of the most used is the Extensible Access Control Markup Language (XACML) [Parducci et al. 2013]. There are several open-source and commercial *authorization servers* that implement Attribute-Based Access Control (ABAC) using *XACML* [Parducci et al. 2013] as policy language. These solutions provide API's that allow the manipulation (e.g. add, update, delete) of the XACML policies in PAP (Policy Administration Point), as well as PDP API's (*Policy Decision Point*) that determine whether a request should be allowed according to their attributes and related policies. However, the manipulation of policies in the PAP does not follow a standard and must be used as defined by the manufacturer, for example, the *WSO2 Identity Server* [1] provides API's for policy manipulation using SOAP Web services, while the AuthZForce product [2] offers operations using REST Web services. Table 1 presents the main administrative operations of the specification presented by [Gouglidis et al. 2017] and two existing authorization servers: the WSO2 and FIWARE AuthZForce.

Gouglidis et al. (2017)	WSO2 IS (PAP API)	AuthZForce (PAP API)
AddPolicy(sattr, oattr, env, ops)	addPolicy(policyDTO)	/domains/pap/policies (POST)
DeletePolicy(sattr, oattr, env, ops)	removePolicy(policyId, dePromote)	/domains/pap/policies (DELETE)
AddAttribute(sub | obj, ATTRs, ATTRo)	-	-
ModifyAttribute(sub | obj, ATTRs, ATTRo)	-	-
AddObject(obj)	-	-
AddOperation(ops)	-	-
AssignSubject(sub, attr)	-	-

Table 1. ABAC administrative operations.

Functional operations not supported by PAP's *WSO2* and *AuthZForce* (e.g. ModifyAttribute) will be handled by *SecAuthAPI*. *SecAuthAPI* will be presented in the following section.

3. The SecAuthAPI

The *SecAuthAPI* consists on an effector for supporting the self-adaptation of authorization infrastructures. *SecAuthAPI* provides facilities for manipulating access control policies in authorization servers (*product-agnostic*) using the XACML language [Parducci et al. 2013] based on the functional operations proposed for [Gouglidis et al. 2017].

A general view of our proposal is presented in Figure 2. Since we define an effector, the client of *SecAuthAPI* is a self-adaptive controller (e.g., *saRBAC* [da Silva et al. 2017], *SAAF* [Bailey et al. 2011] or *Ariadne* [Tsigkanos et al. 2015]), which is able to identify anomalous activities and to

[1] http://wso2.com/identity-and-access-management

[2] https://authzforce.ow2.org

© 2017 Sociedade Brasileira de Computação

make decisions on how to respond to those anomalies. These tools can communicate with *SecAuthAPI* through a REST interface provided by the *SecAuthAPI PAP* component for performing the actions of an adaptation plan. This component comprises the functional operations presented in the previous section.

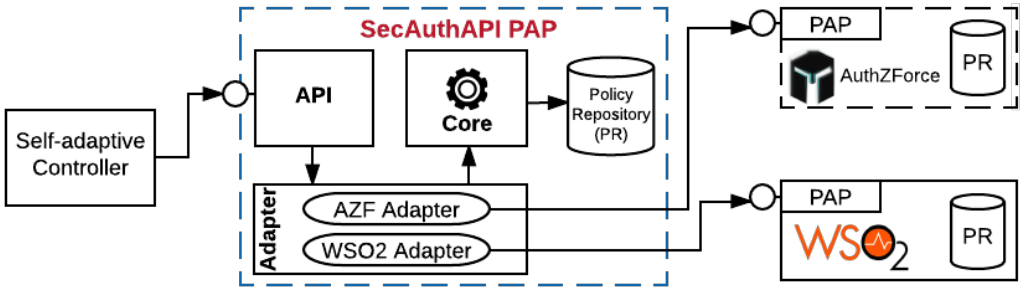


Figure 2. SecAuthAPI - Architecture

The *SecAuthAPI PAP* processes the requests for policy manipulation, and routes them to the *Adapter* component. This component is responsible for implementing specific calls for each *authorization server*. As previously mentioned, each authorization server might provide a different set of operations and access protocols. Thus, each authorization server requires a specific *Adapter*, for example, an adapter for the **WSO2 Identity Server** or for the Fiware **AuthZForce**. The proposed architecture is flexible, allowing new adapters to be implemented for other authorization servers (e.g. OpenAM[3], AT&T XACML[4]).

The *Core* module is responsible for local manipulation of policies, interacting with a *Policy Repository* for persistence. The *Adapter* consumes methods of *Core* when dealing with client requests, mediating the manipulation of policies when the requested operation is not supported by the *authorization server*.

In the sequence, we present the API of the *SecAuthAPI PAP* component, followed by a description of the inner workings of *SecAuthAPI*.

3.1. SecAuthAPI Design

The *SecAuthAPI PAP* component follows the functional specification proposed by [Gouglidis et al. 2017] for the ABAC model. In this way, this component offers a generic ABAC API that provides administrative operations for manipulating *XACML* access control policies at a finer granularity level than supported by existing authorization servers, for example, modify a subject expected attribute on a rule.

The main administrative operations offered by the *SecAuthAPI PAP* component are presented in Table 2. Our component is based on the REST architectural model, thus the operations are presented in terms of HTTP verbs (e.g., *GET, POST*) and resources (e.g., */policy*).

Requests received through our API are routed to the Adapter component. Those operations that are directly supported by the authorization server (e.g., Add policy) can be directly routed without any kind of manipulation. On the other hand, more complex

[3]https//forgerock.org/
[4]https://github.com/att/XACML

 © 2017 Sociedade Brasileira de Computação

HTTP Verb	Method	Description
GET	/policy	List All Policies in PAP
POST	/policy	Insert Policy in PAP
GET	/policy/policy_name	Get Policy Data
PUT	/policy/policy_name	Modify Policy in PAP
DELETE	/policy/policy_name	Delete Policy in PAP
PUT	/policy/policy_name/rule_name/	Modify subject/object attribute on policy

Table 2. API - Administrative Operations

operations, such as modification of a subject attribute of a particular policy, which are not supported by *authorization servers* must be handled locally within SecAuthAPI before they can be forwarded to the *authorization server*.

Figure 3 presents an example of such behaviour, in which a Self-Adaptive Controller decided to add a subject attribute to the policy. Upon receiving the request, the *SecAuthAPI PAP* routes it to the active adapter (in this example, an adapter for the WSO2 Identity Server). The Adapter, knowing which operations are supported by the authorization server, follows the adapter architectural pattern for dealing with those operations that are not supported. Thus, in case an operation A is not supported by an authorization server, the adapter must orchestrate a sequence of operations (a, b, c) in terms of the operations supported by the authorization server and the methods of the *Core* component for manipulating policies. For example, in case the Self-Adaptive Controller issued a request for the *Add_Subject_Attribute()* operation, the API *SecAuthAPI* will route the call to the *WSO2 Adapter*, which will retrieve the policy from the PAP authorization server *WSO2 IS*, then the *WSO2 Adapter* will check if the attribute already exists in policy, and if not, a call to *Core* will be held to handle the policy and include this attribute, then the policy is returned to the *WSO2 Adapter*, which will update the same in PAP authorization server *WSO2 IS*, and finally returns to the *Self-adaptive Controller* the HTTP status code.

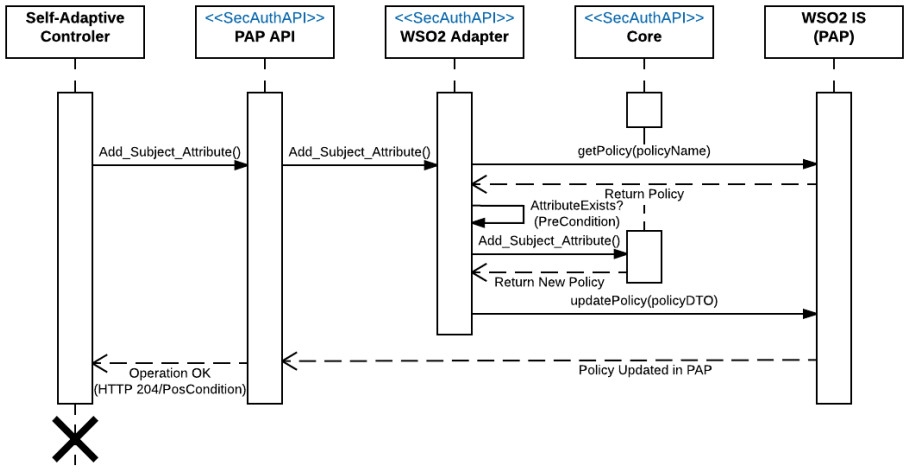


Figure 3. SecAuthAPI - UML Sequence Diagram

© 2017 Sociedade Brasileira de Computação

3.2. Implementation Details

The prototype was implemented in *Python*[5] using the *Django*[6] Web framework. The *Django* assisted in the development of the persistence layer of the policies in the database, in the development of a Web interface for administration policies, and unit tests. To implement the REST API the *Django Rest Framework*[7] was used, providing mechanisms for authentication, data serialization, browsable web API, and others.

Figure 4 presents an UML diagram highlighting the main classes of SecAuthAPI. In *Policy* class model the *name* property is used to store the name of the policy, in *description* a more detailed description about the same, and in *content* the *XACML* code of the policy. The CRUD methods (e.g. create, read, update, delete) model are implemented by the *Django Framework*, however, the *Policy* class was necessary to override the *save* method to enable the capture of the policy name from the XACML code, since the policy name is used by API's authorization services (e.g. WSO2 IS).

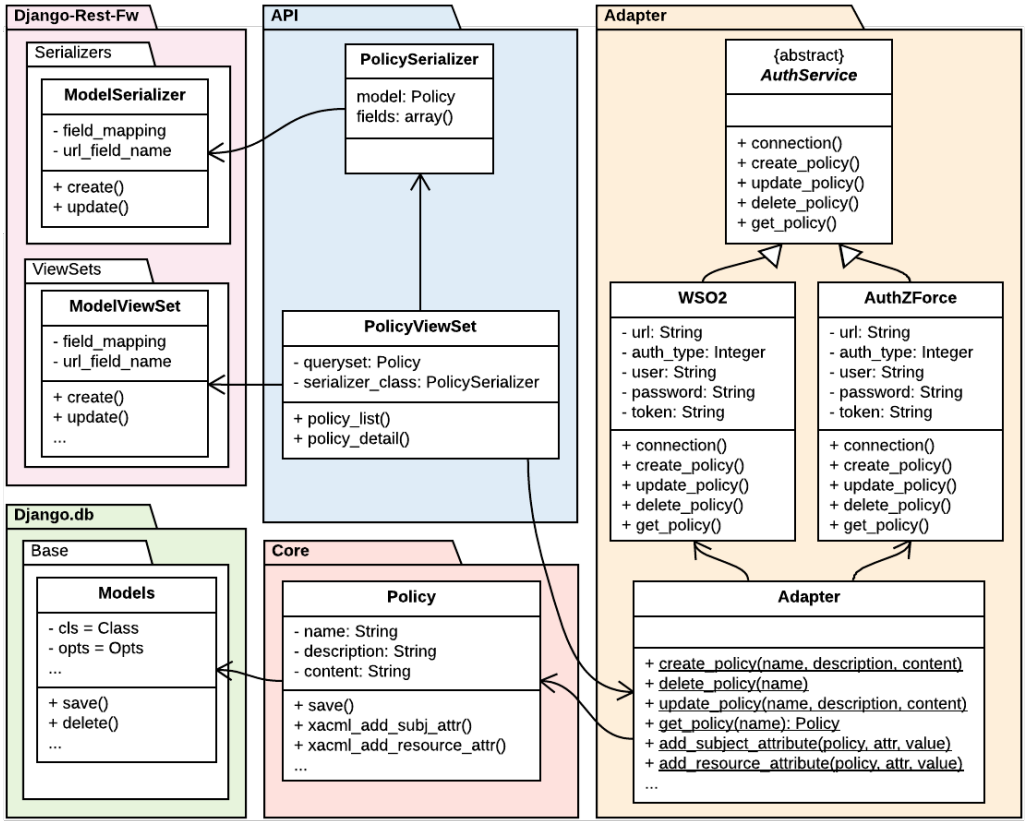


Figure 4. SecAuthAPI - UML Class Diagram

In *PolicySerializer* class are specified the domain class (model) and the properties of the model that should be serialized by the *Django-Rest-Framework* in the operations with the API. The *PolicyViewSet* class implements the API endpoints, i.e. when an endpoint of the API is fired, a *PolicyViewSet* class method is executed, then the operations such as checking the HTTP verb (e.g. GET, PUT, POST) are evaluated, and the results

[5]https://www.python.org/
[6]https://www.djangoproject.com/
[7]http://www.django-rest-framework.org/

 © 2017 Sociedade Brasileira de Computação

are serialized and returned to the client of the API, and when necessary, operations are carried out (e.g. insert policy) authorization systems (e.g. WSO2 IS).

Managing authorization servers policies is carried out from the *Adapter* class, which based on the settings defined in the *settings.py* (e.g. auth_type, url, user, password, token), fires methods to the authorization system selected in the configuration *auth_type* (e.g. 1 = WSO2 IS, 2 = AuthZForce). WSO2 and AuthZForce classes are concrete implementations of the abstract class *AuthService*.

Functional operations like *Add Policy* and *Delete Policy* are already supported by authorization servers *WSO2* and *AuthZForce*. However, the AuthZForce allows the manipulation of policies via REST API, while WSO2 uses SOAP. Thus the need of Adapter to handle the policy according to the protocol used in each authorization server. This manipulation is performed by product-specific methods, for example, the enveloping and sending via SOAP protocol policy required by WSO2 is performed by the method *add_Policy()* of WSO2 concrete class when the Adapter receives a call to insert, and policy and is configured to use this product.

Functional operations not supported by products, such as modifying a subject attribute, were implemented in the *SecAuthAPI* in the product-specific classes (e.g. WSO2 class). The XACML policy is retrieved, the XML is converted to python dictionary (dict) using the *XMLToDict*[8] library, and finally the dict is again converted to XML, validated against the XACML XML Schema, and the XML (XACML policy) is published on PAP of the *authorization server*.

The operations follow the functional specification proposed by [Gouglidis et al. 2017], for example, the functional operation *ModifyAttribute (sub | obj, ATTRs, ATTRo)* is mapped in the API for the HTTP verb *PUT* and */policy/policy_name/rule_name* resource. Additional data as attribute to be changed and the new value for the attribute are sent in JSON format along with the body of the request.

4. Evaluation

In this section will be presented the case study used to demonstrate the feasibility of our approach.

We used the example presented in [da Silva et al. 2017] as it contains a very simple access control policy together with possible adaptations. The example consists of a business process of an IT support system, in which clients open tickets raising issues that are dealt with by members of the support team. This process has an access control policy according to the RBAC model, defining three roles: Client, Support and Admin. The *Client* is any user who needs an IT service and expressed such need opening tickets. A *Support* attendant is an IT team member responsible for dealing with *Clients* tickets. Finally, the *Administrator* supervises support work, interacting with tickets when it is necessary.

The IT Support process includes permissions for opening a ticket, either by a client (Open ticket) or by a support attendant on behalf of a customer (Open ticket of behalf), e.g., the client might go to the IT department personally to report a problem. Opened

[8]https://github.com/martinblech/xmltodict

© 2017 Sociedade Brasileira de Computação

tickets can be allocated to a support attendant that can work on the issue, or cancel the ticket. Upon finishing dealing with the ticket, the support may mark it as solved, and the client then can close or reopen the ticket.

This example considers that users of the system might abuse their permissions, e.g., a client that reopens many tickets when compared to the remaining clients, or a support attendant that opens tickets on behalf of users, allocates the ticket to itself, and then cancel it, artificially increasing its productivity levels. In case an abuse is detected, the access control policy is modified in order to mitigate it. The detection of an abuse, and the decision making on how to respond to such abuse is outside the scope of this paper, which provides the supporting infrastructure needed for conducting the generated adaptation plan.

In our scenario, we considered the case where a particular user has been prohibited of opening new tickets. We have created a REST service with operations for each of the activities of the business process, and then protected this service using an XACML policy with the WSO2 Identity Server. This policy, presented on Listing 1, defines which attributes are expected on a request (resourceId=”/new_ticket”, role=”client” or ”support”, and actionId=”POST”). Due to space constraints, the example of policies are presented using the *ALFA* [9] Eclipse plugin, which generates XACML policies from a simplified syntax. This policy states that for opening new tickets the user must be in role “client” or “support”. The created XACML has then been inserted into WSO2IS through its PAP interface.

```
1  policy new_ticket {
2      apply denyUnlessPermit
3      rule can_new_ticket {
4          target clause ticket.resourceId == ”/new_ticket”
5                 clause ticket.actionId == ”POST”
6                 clause user.role == ”client” or
7                        user.role == ”support”
8          permit
9      }
10 }
```

Listing 1. XACML Policy: new_ticket

To simulate the opening of new tickets, we created a Python prototype that makes XACML requests to the PDP interface of WSO2, and receives the XACML response produced. We consider that user attributes (e.g., userId, role) can be obtained by our prototype and are included on the requests. We then created and sent several XACML requests to check whether the policy was being correctly evaluated, e.g., requests with role “visitor”, “client” and “support”.

To simulate the self-adaptive controller, we consider that a particular user has been detected as abusive and that the decision was to change the access control policy to deny requests from this particular user. The adaptation plan then creates a new access control policy with a new attribute for denying access to the identified user using the REST API resource */policy/* and *POST* verb. The plan has been simulated by means of a Python script in terms of the operations defined by *SecAuthAPI*, which can be verified according

[9]https://www.axiomatics.com/product/developer-tools-and-apis/

 © 2017 Sociedade Brasileira de Computação

to the formal functional specification before being applied into the authorization server. The resulting access control policy is presented on Listing 2 using *ALFA*, where we can notice a new attribute to the target rule (*user.userId == "userxyz"*) and the result is to deny the access.

```
policy deny_new_ticket_userxyz {
    apply denyUnlessPermit
    rule deny_new_ticket_userxyz {
        target clause ticket.resourceId == "/new_ticket"
                clause ticket.actionId == "POST"
                clause user.userId == "userxyz"
        deny
    }
}
```

Listing 2. XACML Policy: deny_new_ticket_userxyz

After the adaptation plan has been successfully executed, we have repeated the XACML requests and compared the results, noticing that the targeted user has been indeed denied access.

The case study developed has demonstrated that our approach is indeed able to support the dynamic modification of access control policies. Moreover, we are able to support operations that are not supported by authorization servers. By following a formal specification for the operations of our interface we allow their use for the automatic derivation of adaptation plans while providing assurances about the consistency of modified policies by means of pre- and post-conditions associated with each operation. Additionally, our approach conducts a structural validation of XACML policies by employing its XML Schema definition.

5. Related work

There are a plethora of works on dynamic access control. [McGraw 2009] introducted the RAdAC (Risk-Adaptable Access Control) model based on the idea that access control decisions should consider a trade-off between risk and potential benefits in allowing access, supportin dynamic decision making in granting of access. [Shaikh et al. 2012] presented a method for risk-based access control decision, in which risk and trust are calculated based on historic user behaviour of granted access. [Kandala et al. 2011] proposed an approach that combines access control with risk and trust, using an abstract formal model to express risk-awareness concepts in Usage Control (UCON).

However, there is a distinction between dynamic access control decisions and dynamic modification of access control policies. The approaches based on the first consider that risk related information is encoded in the access control policy beforehand, restricting the decision making process to whether or not to grant access to a particular request.

On the other hand, approaches based on the dynamic modification of access control policies focuses on adapting authorization policies during runtime to broaden or narrow access permissions. For example, *Ariadne* [Tsigkanos et al. 2015], *SAAF* (Self-Adaptive Authorization Framework) [Bailey et al. 2011] or *saR-BAC* [da Silva et al. 2017]. Such approaches rely on self-adaptive controllers for dynamic modifying access control policies in response to threats and violations detected on the environment.

© 2017 Sociedade Brasileira de Computação

All cited works demand the development of ad hoc code to each distinct adaptation and target component, which represents an overburden to developers. A better approach would be to tailor generic effectors supplied by some technology that already interacts nicely with one or more of the sub-systems or sub-system components. Our works address this issue providing means to abstract away from the peculiarities of individual effectors implementation details, with a uniform software layer.

Some works have focused on the verification of access control policies. Fisler et al. [Fisler et al. 2005] presented a tool called Margrave that uses multi-terminal binary decision diagrams to verify user-specified properties and perform change-impact analysis. [Bailey et al. 2011] propose rbacDSML, a model-driven self-adaptive approach to provide assurances that the deployed adaptation conforms to the system requirement by representing RBAC policies using UML profiles. [Basin et al. 2006] defines the abstract syntax to annotate UML diagrams with information about access control. The authors introduce concepts like User, Role, and Permission as well as relationships between them as elements of a domain specific meta-model. To the best of our knowledge, ours is the first work that is based on formal functional specification of management operations for manipulating access control policies, contributing for the provision of assurances about access control policies.

6. Conclusion and Future Work

This paper has presented *SecAuthAPI*, an approach for supporting the dynamic modification of access control policies on existing authorization infrastructures. Our approach is focused on implementing a well-defined interface for fine-grain manipulation of access control policies using XACML. For this, we take as basis a formal functional specification of ABAC management operations, acting as an agnostic PAP, supporting the use of self-adaptive controllers for dynamically manipulating access control policies on different authorization servers without the need to implement specific interfaces.

The feasibility of our approach has been demonstrated through a prototype that has been used for simulating an adaptation on a real authorization server. However, one limitation of our approach is the need of defining a flow of operations for achieving those manipulations not supported by authorization servers. Such flow can be defined based on the pre- and post-conditions of the formal specification of the management operations, thus allowing the provision of assurances about the modified policy.

As future work, we intent to implement all methods defined by the ABAC formal specification, and then to formally verify this implementation. Additionally, our approach can also be used to support graphical tools for the definition of access control policies.

References

ANSI (2004). Role based access control. Technical Report ANSI INCITS 359-2004, ANSI.

Bailey, C., Chadwick, D. W., and De Lemos, R. (2011). Self-adaptive authorization framework for policy based rbac/abac models. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 37–44. IEEE.

 © 2017 Sociedade Brasileira de Computação

Basin, D., Doser, J., and Lodderstedt, T. (2006). Model driven security: from uml models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15:2006.

Cappelli, D., Moore, A., and Trzeciak, R. (2012). *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. Addison-Wesley Professional.

Cole, E. (2015). Insider threats and the need for fast and directed response. Technical report, SANS Institute InfoSec Reading Room.

da Silva, C. E., da Silva, J. D. S., Paterson, C., and Calinescu, R. (2017). Self-adaptive role-based access control for business processes. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '17, pages 193–203, Piscataway, NJ, USA. IEEE Press.

Fisler, K., Krishnamurthi, S., Meyerovich, L. A., and Tschantz, M. C. (2005). Verification and change-impact analysis of access-control policies. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 196–205.

Gouglidis, A., Hu, V. C., Busby, J. S., and Hutchison, D. (2017). Verification of resilience policies that assist attribute based access control. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 43–52. ACM.

Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., et al. (2013). Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162).

Kandala, S., Sandhu, R., and Bhamidipati, V. (2011). An attribute based framework for risk-adaptive access control models. In *2011 Sixth International Conference on Availability, Reliability and Security*, pages 236–241.

Legg, P. A., Buckley, O., Goldsmith, M., and Creese, S. (2015). Caught in the act of an insider attack: detection and assessment of insider threat. In *2015 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6.

McGraw, R. W. (2009). Risk adaptable access control (RAdAC). Technical report, National Institute of Standards and Technology, McLean and Clifton, VA, United States.

Parducci, B., Lockhart, H., and Rissanen, E. (2013). Extensible access control markup language (xacml) version 3.0. *OASIS Standard*, pages 1–154.

Shaikh, R. A., Adi, K., and Logrippo, L. (2012). Dynamic risk-based decision methods for access control systems. *Comput. Secur.*, 31(4):447–464.

Tsigkanos, C., Pasquale, L., Ghezzi, C., and Nuseibeh, B. (2015). Ariadne: Topology aware adaptive security for cyber-physical systems. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 729–732.

Yaseen, Q. (2016). Insider threat in banking systems. *Online Banking Security Measures and Data Protection*, page 222.

Yuan, E., Esfahani, N., and Malek, S. (2014). A systematic survey of self-protecting software systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(4):17.

© 2017 Sociedade Brasileira de Computação

Análise de sistemas de gestão de identidade baseados em blockchain

Antônio Unias de Lucena[1], Marco Aurélio Amaral Henriques[1]

[1]Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas (Unicamp)
Campinas-SP-Brasil CEP 13083-852

{alucena,marco}@dca.fee.unicamp.br

Abstract. *A distributed and fraud robust environment, as offered by blockchains, can increase the safety and confidentiality of Identity Management Systems, as Remote Identity Proofing (RIP) and Public Key Infrastructure (PKI). This paper analyzes some prototypes of RIP and PKI systems built over blockchains, discussing their features and both strong and weak points. The analysis concludes that (a) a private blockchain is not more secure than a redundant database storing hashes of its data, (b) a RIP or a PKI built over a public blockchain may require an external storage structure and (c) a consortium type blockchain enables the gathering of RIPs or PKIs under a blockchain allowing them to manage the creation of new blocks and makes them more transparent to users.*

Resumo. *Um ambiente robusto a fraudes e distribuído, características do blockchain, pode aumentar a confiabilidade e segurança de sistemas de gestão de identidades, como os de comprovação remota de identidade (RIP) e de infraestruturas de chaves públicas (PKI). Este artigo analisa alguns protótipos de RIPs e PKIs construídos sobre blockchains, levantando suas características e pontos fortes e fracos. A análise conclui que (a) um blockchain privado não fornece mais segurança que um banco de dados com servidores replicados e que armazena hashes de seus dados, (b) um RIP ou uma PKI construídos sobre um blockchain público podem requerer uma estrutura externa para auxiliar no armazenamento de dados e (c) blockchains tipo consórcio possibilitam a reunião de várias RIPs ou PKIs para o gerenciamento da criação de novos blocos e possibilitam que sejam mais transparentes a seus usuários.*

1. Introdução

O conceito de blockchain surgiu com a criptomoeda Bitcoin [Nakamoto 2008], tendo como finalidade armazenar todas as suas transações financeiras. As características do blockchain de ser robusto a fraudes e distribuído têm chamado a atenção para o seu uso por outras aplicações tais como contratos inteligentes (smart contracts), serviços de cartório, serviços de nomes de domínio e gestão de identidades, incluindo Mecanismos de Comprovação Remota de Identidade (RIP - Remote Identity Proofing) e Infraestruturas de Chaves Públicas (Public Key Infrastructure - PKI).

RIPs são vitais em ambientes que requerem segurança e PKIs, por sua vez, são responsáveis pela garantia de vínculo entre uma identidade e uma chave pública, fato

 © 2017 Sociedade Brasileira de Computação

que possibilita comunicações seguras entre duas partes, viabilizando serviços como comércio eletrônico, online banking e atualização de software, entre outros.

Incidentes recentes como o relacionado à autoridade certificadora (CA - Certification Authority) TurkTrust [Ducklin 2013], que emitiu indevidamente certificados para duas autoridades certificadoras intermediárias, expuseram como um erro na cadeia de certificados da arquitetura X.509 pode desencadear consequências funestas. Outro incidente foi com a DigiNotar [Prins 2011]: esta CA foi comprometida e passou a emitir certificados fraudulentos. Deste incidente, ficou clara a importância de se tornar mais transparentes as atividades de uma CA, através da divulgação imediata de todos os certificados emitidos por ela.

O desenvolvimento de PKIs com base em blockchains tem como principais vantagens a possibilidade de trazer mais transparência às atividades da PKI, uma vez que a emissão e a revogação de certificados são registradas em blockchain, tornando tais operações mais visíveis e auditáveis por todos. Além disso, listas de certificados emitidos e revogados, no caso de PKIs, e listas de identidades, no caso de RIPs, são armazenadas de forma distribuída em blockchains, podendo ser consultadas de forma mais eficiente e segura, pois o uso do blockchain possibilita o acesso por múltiplos pontos, dificultando ataques tipo man-in-the-middle e DoS (Denial of Service).

Este artigo faz uma análise das características e dos pontos fortes e fracos de protótipos de RIPs e de PKIs construídos sobre blockchains. A seção 2 detalha conceitos básicos de um blockchain; a seção 3 apresenta uma breve revisão sobre RIPs e PKIs, e a seção 4 traz uma análise de algumas implementações de RIPs e PKIs existentes. Por fim, a seção 5 resume as principais conclusões obtidas da análise e aponta direções para trabalhos futuros.

2. Blockchain: conceitos e características

O blockchain é uma lista encadeada crescente de dados, organizados em blocos, protegida contra adulteração e auditada pelos membros de uma rede peer-to-peer. O blockchain possui as seguintes propriedades: a) descentralização: inexistência de uma autoridade central controladora; b) persistência: uma vez armazenado no blockchain, um dado não pode ser apagado; c) anonimato: as transações realizadas no blockchain estão relacionadas a endereços públicos genéricos e não a identidades individuais; d) auditabilidade: qualquer usuário pode verificar e auditar as operações feitas no blockchain; e) caráter distribuído: o conteúdo desta cadeia é replicado entre todos os nós membros de uma rede peer-to-peer.

A ideia do blockchain surgiu com a criptomoeda Bitcoin como solução para se evitar gasto duplo [Nakamoto 2008]. Viu-se depois que as características do blockchain também poderiam ser úteis em aplicações não relacionadas a transações financeiras.

A Figura 1 ilustra como um blockchain é organizado. As operações (transações financeiras, por exemplo) realizadas em um determinado período de tempo têm seus hashes organizados em forma de árvore de Merkle e a raiz desta árvore, Merkle root, é armazenada no cabeçalho deste bloco. As transações são assim organizadas para tornar detectável qualquer tentativa de fraude em qualquer transação. Além disso, o cabeçalho de um bloco é construído utilizando-se, dentre outros itens, o nonce (resultado do desafio matemático solucionado pelo minerador), o timestamp (especificando o horário em que o bloco foi criado) e o hash do cabeçalho do bloco anterior. Este último item é

 © 2017 Sociedade Brasileira de Computação

responsável por ligar fortemente todos os blocos, de tal maneira que qualquer alteração nos dados de um bloco já existente ficará prontamente evidente.

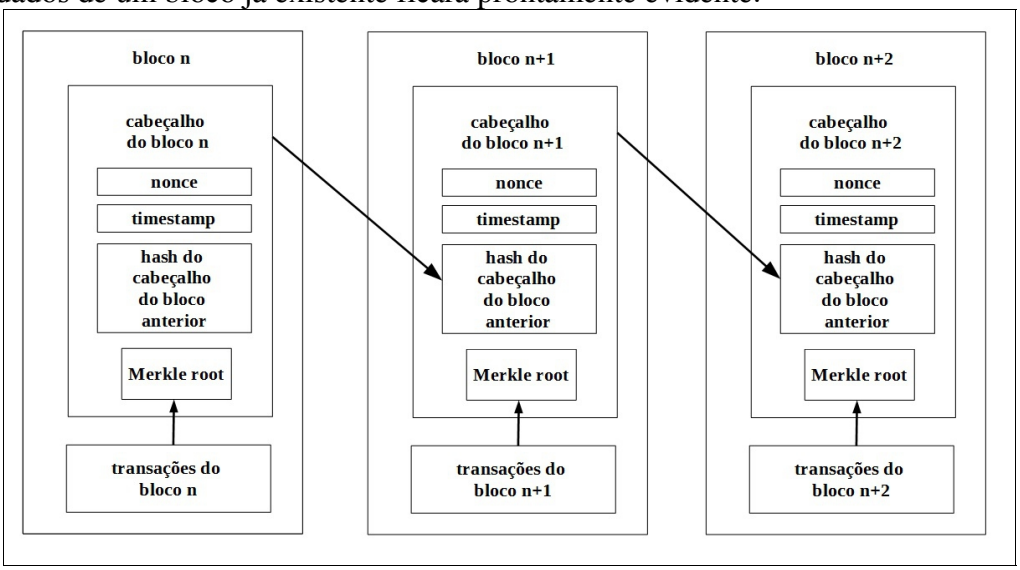


Figura 1. Modelo simplificado de um blockchain

Uma das principais diferenças entre os blockchains é o mecanismo de consenso adotado, determinando como um novo bloco é criado pelo blockchain. É justamente no projeto do mecanismo de consenso que estão concentradas as maiores atenções quando se projeta um novo sistema de blockchain. Dentre os mecanismos de consenso existentes destacam-se: PoW (Proof-of-Work), PoS (Proof-of-Stake) e o PBFT (Practical Byzantine Fault Tolerance)**.**

O PoW baseia-se na solução de um desafio matemático em que vários nós do blockchain competem entre si até que algum encontre uma solução a este desafio e, assim, possa registrar um novo bloco no blockchain. A criptomoeda Bitcoin recompensa o nó que vence a corrida do PoW e cria um novo bloco. Por isso, este processo passou a ser chamado de mineração e os nós responsáveis por ele, de mineradores. O PoW tem como pontos negativos a concentração de mineração em torno de usuários com maior poder computacional, alto consumo de energia elétrica durante o processo de mineração, o desperdício de energia elétrica dos mineradores que não venceram a disputa para ser o primeiro a encontrar a solução do PoW, alta latência entre blocos e baixa taxa de transferência de transações [Vukolic 2016].

No mecanismo de consenso PoS (Proof-of-Stake), em vez de os nós competirem entre si para determinar quem será o primeiro a encontrar uma solução para um desafio matemático, ocorre um processo de seleção do nó responsável pela criação do próximo bloco. A probabilidade de um nó ser selecionado para ser o minerador do próximo bloco é proporcional à quantidade de moedas que ele possui. Esta abordagem leva o PoS a gastar menos energia que o PoW [Tschorsch et al 2016], porém, o PoS possibilita concentração de riqueza, pois os ricos têm mais chance de ficarem cada vez mais ricos. Um exemplo de blockchain que utiliza o PoS é o Blackcoin [Vasin 2014].

O PBFT, por sua vez, é um algoritmo desenvolvido em 1999 como solução ao problema de se encontrar consenso entre usuários de uma rede em que não existe confiança entre os participantes da rede, devido à existência de usuários maliciosos [Castro e Liskov 1999]. Este algoritmo retorna uma resposta rápida ao problema do

 © 2017 Sociedade Brasileira de Computação

consenso, porém, exige que todos os participantes no consenso sejam conhecidos e que exista no máximo 33% de usuários maliciosos no ambiente. Permite a ocorrência de uma quantidade muito maior de transações por intervalo de tempo quando comparado ao PoW [Vukolic 2016]. É utilizado no projeto Hyperledger [Hyperledger project 2017].

A Tabela 1 sumariza um comparativo entre os mecanismos de consenso PoW, PoS e PBFT.

Tabela 1. Comparação entre os mecanismos de consenso PoW, PoS e PBFT

	PoW	**PoS**	**PBFT**
Ponto forte	Não requer nenhum processo prévio para definir minerador	Menor consumo energético que o PoW	Número de transações por segundo maior que o do PoW
Ponto fraco	Alto consumo energético e latência entre blocos	Concentração de riqueza	Não pode ser utilizado em blockchains públicos
Exemplo	Bitcoin	Blackcoin	Hyperledger

Além de serem classificados quanto ao mecanismo de consenso, os blockchains existentes podem ser classificados também quanto à forma como são administrados: públicos, consórcios ou privados.

Um blockchain é dito público quando o mesmo é gerenciado por todos seus nós e quando a entrada de qualquer novo nó é permitida. Como exemplo de um blockchain público, temos o Bitcoin. Um blockchain é classificado como consórcio, quando a entrada de novos nós é controlada e apenas alguns nós gerenciam o funcionamento do blockchain. Um exemplo de blockchain consórcio é o Hyperledger. Por fim, um blockchain será privado quando for gerenciado por apenas um nó (ou por um grupo de nós pertencentes a uma mesma organização), que é responsável pela geração dos novos blocos e pela permissão de entrada de novos nós. O Gem Health [Prisco 2016], blockchain voltado ao armazenamento de dados médicos, é um exemplo de blockchain privado.

3. Revisão sobre Mecanismos de Comprovação Remota de Identidade e Infraestruturas de Chaves Públicas

Mecanismos de Comprovação Remota de Identidade (RIP - Remote Identity Proofing) são responsáveis por assegurar uma associação entre a identidade averiguada e os dados correspondentes a esta armazenados remotamente.

Infraestruturas de Chaves Públicas (PKI - Public Key Infrastructure) são responsáveis por garantir associação entre uma identidade e uma chave pública. Esta associação é garantida por certificados, que são documentos assinados digitalmente utilizando a chave privada de um intermediário confiável, a autoridade certificadora (CA - Certification Authority), de acordo com um padrão preestabelecido como, por exemplo, o padrão X.509 [Myers et al 1999].

Existem duas principais arquiteturas para PKIs: as baseadas em autoridades certificadoras (CA-PKI) e as baseadas em redes de confiança (web of trust: WoT-PKI). A CA-PKI tem como pontos fracos a centralização de suas operações e a necessidade de se transmitir e verificar cadeias de certificados quando há vários níveis de CAs. Uma

© 2017 Sociedade Brasileira de Computação

CA possui transparência quando são tornados públicos, em tempo real, todos os certificados emitidos e revogados por esta. Com esta transparência, incidentes como o da DigiNotar [Prins 2011], em que a CA foi hackeada e passou e emitir certificados fraudulentos, poderiam ser mais facilmente percebidos. Para lidar com ameaças semelhantes ao caso da DigiNotar, a empresa Google criou o serviço Certificate Transparency [Laurie et al 2017] que registra a emissão de novos certificados e envia um comunicado ao suposto titular do registro imediatamente após a emissão de um certificado.

A estrutura em árvore de CAs, seguindo a arquitetura X.509, permite a emissão pela CA-raiz de certificados a outras CAs, ditas intermediárias ou de segundo nível, autorizando-as a emitir novos certificados com o aval da CA-raiz. Com isto, erros na emissão destes certificados provenientes de uma CA localizada em um nível acima na estrutura em árvore permitem o surgimento de uma CA fraudulenta certificada por outra CA confiável, caso do incidente com a CA TurkTrust [Ducklin 2011]. Neste caso, a publicação imediata de toda e qualquer atividade da CA tornaria evidente que esta CA não estava funcionando corretamente.

A WoT-PKI, por sua vez, foi proposta por Phil Zimmermann no manual do PGP (Pretty Good Privacy) [Zimmermann 1995]. Em sua estrutura não existe uma autoridade centralizadora, sendo novos usuários certificados por usuários já existentes, criando uma rede de confiança. Por outro lado, a certificação de um novo usuário pode levar tempo para se tornar amplamente aceita pela comunidade e existe a possibilidade de um usuário criar múltiplas chaves. Além disso, grupos de usuários maliciosos podem certificar a entrada de outros usuários maliciosos. Outra característica desta arquitetura é o fato de as chaves não expirarem. Elas só deixam de ser válidas se forem revogadas pelos seus donos. Existe uma estimativa de que pelo menos 40% das chaves PGP existentes estão associadas a contas de e-mails inacessíveis e cujos provedores nem mais existem [Leiding e Dähn 2012].

Fredriksson em seu trabalho [Fredriksson 2017] fornece um perfil desejável de uma PKI implementada em blockchain: (a) retenção de identidade (uma identidade pode ser modificada apenas por seu proprietário), (b) expiração de certificados antigos, (c) transparência (qualquer atividade da CA, emissão e revogação de certificados, é publicada e facilmente acessada) e (d) suporte a clientes com baixa capacidade de armazenamento de dados (celulares, por exemplo).

A retenção de identidade é importante, pois provê confiabilidade ao certificado de um usuário. Já a expiração de identidade é importante para diminuir o tamanho da base de dados armazenada e evitar ataques que usem certificados velhos e sem uso. A transparência, por sua vez, permite que o comprometimento de uma CA seja percebido com facilidade. E, por fim, um dos maiores desafios em desenvolver uma PKI em blockchain é assegurar que equipamentos com baixa capacidade de processamento e armazenamento de dados possam acompanhar a atividade de uma PKI e a lista de seus certificados revogados sem grandes dificuldades.

4. Avaliação de RIPs e PKIs implementados sobre blockchain

Esta seção traz uma análise de RIPs e PKIs implementados sobre blockchain. A principal vantagem da construção de RIPs e PKIs sobre blockchains é que seus serviços seriam prestados e registrados em uma plataforma distribuída, transparente e robusta

 © 2017 Sociedade Brasileira de Computação

contra fraudes. Além disso, com o uso de blockchains, um usuário poderia buscar uma identidade, no caso de RIPs, ou um certificado, no caso de PKIs, em qualquer um dos vários nós do blockchain, eliminando as ameaças existentes ao acesso a sistemas centralizados. Já existem trabalhos com o de Fredriksson [Fredriksson 2017] em que limitações de algumas PKIs implementadas em blockchain são discutidas, mas Fredriksson se restringe a discutir os efeitos da concentração de mineração, escalabilidade e integração das propostas de PKIs em blockchain com as implementações de PKI já existentes utilizando o padrão X.509.

Este artigo propõe comparar algumas propostas de RIPs e PKIs em métricas consideradas determinantes para o funcionamento eficaz de sistemas de gestão de identidade em blockchain: a) segurança contra alterações em blocos antigos do blockchain; b) possibilidade de criação de blocos fraudulentos; c) taxa de armazenamento de dados; d) capacidade de armazenamento de textos arbitrários; e) capacidade de oferecer um serviço distribuído.

A segurança contra alterações em blocos antigos é um quesito importante para não se perder todo o investimento realizado para o desenvolvimento de um RIP ou PKI em blockchain, caso os criadores de novos blocos passem a adulterar blocos anteriormente armazenados. A possibilidade de criação de blocos fraudulentos é relacionada à probabilidade de um novo bloco ser criado com erro, com identidades ou certificados fraudulentos, pelos mineradores do blockchain. A taxa de armazenamento de dados define a capacidade de armazenamento de dados no blockchain que o sistema de gestão de identidade possuirá. A capacidade de armazenamento de textos arbitrários, por sua vez, é relevante por permitir o armazenamento de informações atreladas a uma identidade, no caso de RIPs, ou a um certificado, no caso de PKIs. Por fim, a capacidade de oferecer um serviço distribuído é fundamental para evitar ataques man-in-the-middle e DOS quando um sistema possui um único ponto de acesso.

Este artigo também levanta os principais pontos positivos e negativos das propostas de PKI e RIP analisadas, contribuindo para a criação de diretrizes para o projeto de sistemas de gestão de identidade em blockchain.

A seguir são avaliadas uma proposta de RIP e quatro propostas de PKI construídas sobre blockchain, mostrando os pontos positivos e negativos que cada uma apresenta. Estes sistemas foram escolhidos

4.1. Certcoin

Certcoin [Fromknecht et al 2014] é um protótipo de CA que utiliza a arquitetura de PKI descentralizada e que possibilita retenção de identidade. Utiliza o blockchain do Namecoin [Namecoin 2017] para o armazenamento de dados,

O Certcoin oferece os serviços de registro, atualização e revogação de chaves. Não armazena um certificado no padrão X.509, mas apenas duas chaves públicas (denominadas online e offline) associadas a uma identidade. Um usuário utiliza a chave online para autenticar mensagens, ao passo que a chave offline é utilizada para revogar e atualizar sua chave pública existente (chave online). Além disso, é possível ao usuário a revogação de sua chave offline, informando a outros usuários que esta foi comprometida.

O Certcoin também não verifica identidades, sendo possível um usuário associar uma identidade qualquer a uma chave pública, passando-se por outra pessoa. Além

© 2017 Sociedade Brasileira de Computação

disso, este protótipo de CA propõe mecanismos de buscas de certificados armazenados em blockchain.

4.2. Authcoin

Authcoin [Leiding et al 2016] também é uma CA descentralizada que permite registro de uma identidade associada a uma chave pública em blockchain. Não possui mecanismo de autenticação da identidade, mas provê o serviço de validação de identidade através de e-mail.

Como o objetivo de uma web o trust é garantir a associação entre uma chave pública e um e-mail, o Authcoin utiliza um mecanismo de desafio-resposta na comunicação entre seus usuários. Neste mecanismo, um usuário, Ana, envia uma mensagem ao e-mail de outro usuário, Beto, cifrada com a chave pública que ela acredita ser dele. Beto terá que decifrar a mensagem com a chave privada correspondente e responder Ana. Se Ana receber a mensagem com o conteúdo esperado, saberá que existe uma ligação entre a chave pública de Beto e o e-mail atribuído a Beto, mas não saberá se foi realmente Beto que respondeu tal e-mail.

Authcoin oferece serviços de revogação, expiração e recuperação de chave. A busca por uma chave é realizada percorrendo-se somente parte do blockchain, devido ao prazo de validade limitado (tipicamente 12 meses) de um certificado. Este fato permite que o usuário armazene apenas uma cópia parcial de todo o blockchain, o que é o mais adequado para uso em dispositivos com baixa capacidade de armazenamento de dados, como celulares e dispositivos móveis, por exemplo.

Sua arquitetura foi concebida para ser implementada tanto sobre um blockchain privado como em um blockchain público. Para lidar com a falta de garantia de autenticação de identidades, o Authcoin também armazena no blockchain todos as atividades de seus usuários (respostas dos desafios trocados entre usuários antes de se estabelecer uma comunicação via e-mail).

O Authcoin, se construído sobre um blockchain privado, pode ser substituído por um banco de dados distribuído em vários servidores, para ter a redundância de um blockchain, e este banco de dados deverá ter vários hashes calculados a partir de seus dados para evitar alterações triviais como se faz hoje em bancos de dados comuns. Para se aumentar a confiabilidade deste banco de dados, seria possível também o armazenamento de hashes dos dados contidos neste banco de dados em um blockchain público.

4.3. Pomcor

A proposta de armazenamento de informações de usuários em blockchain da Pomcor [Lewison e Corella 2016] é um Mecanismo de Comprovação Remota de Identidade voltado para usuários de bancos. Nos dados armazenados no blockchain constam informações como dados pessoais e dados de biometria dos usuários do banco. Quando um usuário vai se identificar para o sistema do banco, este busca as informações do usuário armazenadas no blockchain e confere com as informações fornecidas pelo usuário no instante da autenticação.

O RIP da Pomcor utiliza um blockchain privado construído sobre o blockchain público Ethereum [Wood 2014], sistema de uso geral para criação de aplicações e contratos inteligentes sobre blockchain, como suporte para armazenar informações de

 © 2017 Sociedade Brasileira de Computação

seus clientes.

4.4. PrimeKey Solutions

A proposta de implementação de PKI utilizando blockchain apresentada pela PrimeKey Solutions [Fredriksson 2017] consiste no agrupamento de várias PKIs, que atuam como mineradores, sobre um blockchain tipo consórcio utilizando o PoS como mecanismo de consenso para a criação de novos blocos, onde são armazenadas as informações de atualização do estado das PKIs participantes do blockchain.

As chaves públicas armazenadas por uma PKI são organizadas em uma árvore de Merkle e apenas o Merkle root é armazenado em cada bloco do blockchain. O mecanismo de busca é realizado através de Merkle proofs, onde prova-se que uma chave está associada a um Merkle root sem a necessidade de fazer o download de toda árvore de Merkle. Tal característica torna possível o uso desta arquitetura de PKIs em dispositivos com baixa capacidade de armazenamento de dados. Esta proposta oferece retenção de identidade e transparência aos seus usuários, bem como as operações de busca, inserção, atualização e eliminação de certificados. A proposta de PKI tem como ponto fraco a possibilidade de que CAs compitam entre si, sabotando o sistema durante a criação de novos blocos.

4.5. Blockstack

Blockstack [Ali et al 2016] é uma aplicação voltada para o armazenamento de dados no blockchain do Bitcoin. Nesta aplicação, o usuário armazena dados na nuvem do Blockstack e este armazena o hash destes dados no blockchain do Bitcoin. A grande vantagem disto é garantir robustez a fraudes dos dados armazenados na nuvem do Blockstack.

Tanto um RIP como uma PKI desenvolvidos sobre o Blockstack teriam caráter centralizado utilizando um blockchain público apenas para o armazenamento do hash de identidades e certificados. Para realizar a busca de certificados ou identidades é necessário acessar a nuvem do Blockstack e, portanto, sistemas de gestão de identidade apenas terão caráter distribuído e redundante, se sua base de dados for replicada por diversas nuvens do Blockstack.

O Blockstack tem como ponto forte o fato de já fornecer uma estrutura externa ao blockchain para auxiliar no armazenamento de dados. Porém, esta estrutura necessita ser construída de forma replicada e distribuída para fornecer as mesmas características do blockchain para quem construir sistemas de gestão de identidade sobre o Blockstack.

4.6. Comparação entre as propostas apresentadas

Os sistemas de gestão de identidade analisados se diferem principalmente quanto ao tipo de blockchain utilizado (público, consórcio ou privado) por eles. Como os blockchains públicos utilizados, em geral. possuem baixa taxa de transferência de dados, RIPs ou PKIs construídos sobre eles necessitam utilizar uma estrutura externa ao blockchain para auxiliar no armazenamento de dados e apenas o hash destas informações é armazenado no blockchain. Este é o caso de sistemas utilizando o blockchain do Bitcoin e do Namecoin. No entanto, é necessário que a estrutura externa para auxiliar no armazenamento de dados seja replicada em vários servidores, para que os usuários destes sistemas tenham acesso de forma distribuídas às informações, evitando falhas e

© 2017 Sociedade Brasileira de Computação

ataques comuns a ambientes com um único ponto de acesso.

Tabela 2. Comparação entre as propostas de implementação de RIP e PKI sobre blockchain

	Certcoin	**Authcoin**	**Pomcor**	**PrimeKey Solutions**	**Blockstack**
Blockchain utilizado	Namecoin	Privado ou público	Privado construído sobre o Ethereum	Projeto de blockchain consórcio	Bitcoin
O que é armazenado no blockchain	Dois pares de chaves públicas	Chave pública e atividades dos usuários	Dados referentes à identidade	Certificado em formato proprietário	Hash do certificado (padrão X.509)
Caráter distribuído	Não	Sim	Não	Sim	Não
Taxa de armazenamento de dados	1,7 byte/s	NA	NA	8,8 bytes/s	NA
Capacidade de armazenamento de textos arbitrários	Não	Sim	Sim	Sim	Sim
Segurança contra alterações em blocos antigos	Alta	Baixa	Alta	Média	Alta
Possibilidade de criação de blocos fraudulentos	Sim	Sim	Sim	Sim	Sim
Pontos fortes	Uso de dois pares de chaves	Validação de e-mails e armazenamento de toda atividade dos usuários no blockchain	Armazenamento do hash de seus dados no blockchain do Ethereum	Possibilidade de definir o tamanho do bloco armazenado no blockchain	Possibilidade de uso de sua nuvem para armazenamento de dados
Pontos fracos	Ausência de autenticação de identidades	Proposta poderia ser realizada por um banco de dados	Proposta poderia ser realizada por um banco de dados	Possibilidade de sabotagem por CAs	Base da dados não replicada e distribuída

Quanto à característica de fornecer transparência e ter caráter distribuído, apenas as PKIs do Authcoin e da PrimeKey Solutions possuem estas capacidades. Isto é devido ao fato de o Authcoin poder ser construído sobre um blockchain privado e o PrimeKey Solutions fazer uso de um blockchain consórcio, respectivamente e, desta forma,

 © 2017 Sociedade Brasileira de Computação

poderem definir qual será a capacidade de armazenamento de dados do bloco criado pelo blockchain destes. O Blockstack possui capacidade de armazenar textos arbitrários e a atividade da CA em seus servidores, no entanto, estes servidores não possuem caráter distribuído.

A taxa de armazenamento de dados de um blockchain define o quanto um RIP ou uma PKI construídos sobre blockchain poderão armazenar a cada bloco. O Bitcoin e o Namecoin permitem o armazenamento de, no máximo, 1MB a cada 10 minutos, ou seja, ambos possuem taxa de armazenamento de dados de 1,70 byte/segundo. O PrimeKey Solutions projeta um blockchain capaz de armazenar 5,2MB a cada 10 minutos, portanto com um 8,8 bytes/segundo como taxa de armazenamento de dados. As propostas do Authcoin e da Pomcor não definem a capacidade de armazenamento de seus blockchains e, por fim, o Blockstack não informa a taxa de armazenamento de dados em sua nuvem.

Os principais blockchains públicos fornecem baixa capacidade de armazenamento de dados: 83 bytes e 520 bytes por transação para o Bitcoin e Namecoin, respectivamente. Esta baixa capacidade de dados influencia no que será armazenado no blockchain, levando RIPs e PKIs a armazenar apenas o hash de identidades e certificados no blockchain. Blockchain privados e consórcios permitem a definição do tamanho do seu bloco, possibilitando, assim o armazenamento de maior quantidade de informação por bloco. Desta forma, PKIs construídas sobre blockchains públicos não permitem o armazenamento de textos arbitrários, caso do Certcoin. Por outro lado, PKIs e RIPs construídos em blockchains privados, permitem o armazenamento de textos arbitrários. O Blockstack, por utilizar uma estrutura externa ao blockchain para o armazenamento de dados, também tem a capacidade de armazenar textos arbitrários.

Outra métrica importante para o desenvolvimento de RIPs e PKIs em blockchain é a segurança contra adulteração de blocos antigos. Blockchains públicos são considerados praticamente impossíveis de serem adulterados e, por isto, seguros. A proposta de RIP da Pomcor e a de PKI e RIP do Blockstack, fornecem segurança pelo fato de armazenarem valores de hash de seus dados em blockchains públicos. Propostas como a do Authcoin, por poder ser construído sobre blockchain privado, e a do PrimeKey Solutions, por utilizar blockchain consórcio, respectivamente, fornecem menor segurança quanto a adulteração de blocos antigos. O Authcoin, caso seja construído sobre um blockchain privado, gerido por uma única pessoa ou organização, tem menor confiabilidade que um blockchain consórcio, já que neste último é necessário que a maioria dos nós entrem em consenso a fim de fraudar blocos antigos do blockchain.

A possibilidade de criação de blocos fraudulentos existe mesmo em blockchains públicos, quando há concentração de mineração (determinado minerador possui mais de 51% do hashing power do sistema). Blockchains do tipo consórcio também possibilitam a criação de blocos fraudulentos, desde que haja um conluio entre a maioria de seus nós. Por sua vez, blockchains privados, por serem centralizados em torno de um único nó também possuem a probabilidade de emissão de blocos fraudulentos.

A Tabela 2 traz um comparativo entre as propostas de implementação de RIP e PKI utilizando blockchain, salientando suas principais características, pontos fortes e fracos e bem como características esperadas para RIPs e PKIs desenvolvidos em

© 2017 Sociedade Brasileira de Computação

blockchain.

5. Conclusões e trabalhos futuros

É possível realizar uma implementação segura e confiável de sistemas de gestão de identidades como Mecanismos de Comprovação Remota de Identidade (RIPs) e Infraestruturas de Chaves Públicas (PIKs) utilizando blockchains. Entretanto, a opção de construir um sistema destes sobre um blockchain privado não adiciona mais segurança que o uso de banco de dados com servidores replicados em que o hash dos dados contidos nos servidores também são armazenados.

A construção de RIPs e PKIs sobre blockchains públicos necessita usar estruturas exteriores ao blockchain para auxiliar no armazenamento de dados devido à baixa capacidade de armazenamento nos principais blockchains públicos. Tanto no caso de uma PKI ou de um RIP, o uso de blockchain público garante segurança contra fraudes, graças à possibilidade de qualquer um conferir os hashes dos dados armazenados na estrutura externa com os hashes armazenados no blockchain. Porém, para que usuários de RIPs e PKIs tenham acesso de forma distribuída às informações armazenadas nestes, é necessário a replicação das estruturas externas ao blockchain, para não haver falhas ou ataques devido à existência de um único ponto de acesso.

Por fim, o uso de blockchains tipo consórcio é viável, quando são reunidos em um mesmo blockchain entidades (RIPs e PKIs) que atestem a criação de novos blocos, garantindo a confiabilidade dos dados armazenados neste blockchain. Além disso, este tipo de blockchain permite que uma PKI armazene em blockchain tanto as informações de sua atividade (transparência) como o status de seus certificados, permitindo que seus usuários tenham acesso a uma base de dados distribuída e robusta a fraudes.

Temos como trabalho futuro a implementação de um RIP e PKI em blockchain. A primeira implementação será no blockchain do Bitcoin, por este ser o blockchain mais utilizado. Uma subsequente implementação é prevista para o Ethereum, devido ao fato de este ser voltado para a construção de aplicações em blockchain.

Referências

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (acesso: 12/09/2017).

Ducklin, P. (2013). The TURKTRUST SSL certificate fiasco-what really happened, and what happens next. https://nakedsecurity.sophos.com/2013/01/08/the-turktrust-ssl-certificate-fiasco-what-happened-and-what-happens-next/ (acesso: 12/09/2017).

Prins, J. R., & Cybercrime, B. U. (2011). DigiNotar Certificate Authority breach "Operation Black Tulip". https://www.rijksoverheid.nl/binaries/rijksoverheid/documenten/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf (acesso: 12/09/2017).

Vukolić M. (2016) The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. In: Camenisch J., Kesdoğan D. (eds) Open Problems in Network Security. iNetSec 2015. Lecture Notes in Computer Science, vol 9591. Springer, Cham

Tschorsch, F., & Scheuermann, B. (2016). Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys & Tutorials, 18(3), 2084-2123.

Vasin, P. (2014). Blackcoin's proof-of-stake protocol v2. https://blackcoin.co/blackcoin-

 © 2017 Sociedade Brasileira de Computação

pos-protocol-v2-whitepaper.pdf (acesso: 13/10/2017).

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 151. http://gavwood.com/paper.pdf (acesso: 12/09/2017).

Castro, M., & Liskov, B. (1999, February). Practical Byzantine fault tolerance. In OSDI (Vol. 99, pp. 173-186).

Hyperledger project. (2017) https://www.hyperledger.org/ (acesso: 12/09/2017).

Prisco, J. (2016). The Blockchain for healthcare: Gem launches Gem Health Network with Philips Blockchain Lab. https://gem.co/health/ (acesso: 12/09/2017).

Bitcoin Wiki. (2017) https://en.bitcoin.it/wiki/Scalability (acesso: 14/09/2017).

Bitcoin Cash. (2017) https://www.bitcoincash.org/ (acesso: 14/09/2017).

Myers, M., Ankney, R., Malpani, A., Galperin, S., & Adams, C. (1999). X. 509 Internet public key infrastructure online certificate status protocol-OCSP (No. RFC 2560). https://tools.ietf.org/html/rfc6960 (acesso: 12/09/2017).

Laurie, B., Langley, A., & Kasper, E. (2013). Certificate transparency (No. RFC 6962). https://tools.ietf.org/html/rfc6962 (acesso: 12/09/2017).

Zimmermann, P. R. (1995). The official PGP user's guide. MIT press. https://web.pa.msu.edu/reference/pgpdoc1.html (acesso: 12/09/2017).

Leiding, B., & Dähn, A. (2016). Dead Letters to Alice-Reachability of E-Mail Addresses in the PGP Web of Trust. arXiv preprint arXiv:1605.03162. https://arxiv.org/pdf/1605.03162.pdf (acesso: 12/09/2017)

Namecoin. (2017) FAQ – Namecoin. https://namecoin.org/docs/faq/(acesso: 12/09/2017).

Bitcoin. (2017). Bitcoin release 0.12.0. https://bitcoin.org/en/release/v0.12.0. (acesso: 12/09/2017).

Fredriksson, B. (2017). A Distributed X. 509 Public Key Infrastructure Backed by a Blockchain. https://helix.stormhub.org/data/exjobb/Compact.pdf (acesso: 12/09/2017).

Fromknecht, C., Velicanu, D., & Yakoubov, S. (2014). A Decentralized Public Key Infrastructure with Identity Retention. IACR Cryptology ePrint Archive, 2014, 803. https://eprint.iacr.org/2014/803.pdf (acesso: 12/09/2017).

Leiding, B., Cap, C. H., Mundt, T., & Rashidibajgan, S. (2016). Authcoin: Validation and Authentication in Decentralized Networks. arXiv preprint arXiv:1609.04955. https://arxiv.org/ftp/arxiv/papers/1609/1609.04955.pdf (acesso: 12/09/2017).

Lewison, K., & Corella, F. (2016). Backing Rich Credentials with a Blockchain PKI. https://pomcor.com/techreports/BlockchainPKI.pdf (acesso: 12/09/2017).

Ali, M., Nelson, J. C., Shea, R., & Freedman, M. J. (2016, June). Blockstack: A Global Naming and Storage System Secured by Blockchains. https://www.usenix.org/system/files/conference/atc16/atc16_paper-ali.pdf (acesso: 12/09/2017).

Zheng, Z., Xie, S., Dai, H. N., & Wang, H. (2016). Blockchain Challenges and Opportunities: A Survey. https://www.researchgate.net/publication/319058582_Blockchain_Challenges_and_Opportunities_A_Survey (acesso: 12/09/2017).

© 2017 Sociedade Brasileira de Computação

WGID - VII Workshop de Gestão de Identidades Digitais Resumos Estendidos

© 2017 Sociedade Brasileira de Computação

Case Study: Augmenting Security of Cloud-Based Digital Signatures.

Roberto Gallo, Conrado P. L. Gouvêa[1]

[1]KRYPTUS Information Security SA.

{gallo,conradoplg}@kryptus.com

Abstract. *In this paper, we present a case study of Cloud-based user-key management architecture that combines the use of traditional services provided by hardware security modules (HSM), with secure code execution of authentication protocols such as Time- and Hash-based One-time Password Algorithms (TOTP, HOTP), and cloud-enabled key management interfaces for improved security, user experience, and interoperability. Our contribution is threefold: (i) a reference system architecture fit for cloud providers, (ii) security discussion, and (iii) performance figures.*

1. Introduction

The importance of digital signatures is well established for many applications such as authentication and authorization, non-repudiation of documents, and even key agreement protocols. Nonetheless, the usage of digital certification is partially hindered by the challenge of making the user's private key secure and promptly available.

While hardware-based user-side key management is preferable regarding key usage control, cryptographic tokens, smart cards, and other types of physical wallets have many practical drawbacks, such as significant costs, interoperability issues (usually requiring platform dependent device drivers and physical ports), resulting in reduced availability for the users. In the other hand, the software-based user-side alternative is at most as secure as the user's weakest endpoint platform that stores his keys. As the mainstream platforms (such as Windows, Android multi-verse) are most of the time vulnerable, conscious users should avoid this alternative when possible.

In this context, industry applications started to provide cloud-based user-key management, where users instead of storing their keys locally, would rely on authorization mechanisms to unlock the usage of the user's key, on their behalf.

Obviously, this setup gives rise to many new concerns: (i) are user's keys securely stored? (ii) how likely and what happens if an adversary gains control over the cloud service provider? (iii) how easily could a rogue party authorize the user's key usage?

In this paper, we present a case study of Cloud-based user-key management architecture that combines the use of traditional services provided by HSMs, with secure code execution of authentication protocols (such as HOTP [M'Raihi et al. 2005] and TOTP [M'Raihi et al. 2011]), and cloud-enabled key management interfaces for improved security, user experience, and interoperability.

 © 2017 Sociedade Brasileira de Computação

2. Case Study

2.1. Proposed System Architecture

We aim at providing augmented overall system security that: (i) does not increase the complexity of users' interactions when compared to other typical services that employ two-factor authentication (2FA); (ii) does not require special software or drivers on the devices where users consume the crypto services, (iii) facilitates integration.

For proper security, any cloud-service should rely on an HSM for secure key management. Currently, 100% of the NIST FIPS-140-2 [National Institute of Standards and Technology 2001] certified devices support RSA Labs PKCS#11 (P11) [OASIS 2016b] interface, originally proposed in 1994. Although ubiquitous, P11 suffers from notetaking limitations: (i) it is defined as an ANSI-C interface, thus requiring native code on client side, (ii) recognizes only two roles, not even users, so that developers must bind key and users externally, (iii) does not support alternative authentication methods.

Figure 1 depicts what arguably could be the simplest system architecture that enables cloud-based crypto services with 2FA using a legacy P11 HSM. The HSM stores both users private keys $(k_0, k_1, ..., k_i)$ and TOTP or HOTP seeds $(s_0, s_1, ..., s_i)$, where PKCS#11 CKU_USER role owns all keys. The web application implementing the end user service, say digital signature, needs to: (i) serve user's front-end interface; (ii) perform billing and other business logic; (iii) (pre-) authorize user's signature request using a T/HOTP verification component; (iv) dispatch the signature operation to a user-aware crypto service. In this scenario, both the user-aware crypto service and the T/HOTP component interface with the PKCS#11 through the P11 Service, which in turns requires PINs for its role-based credential scheme.

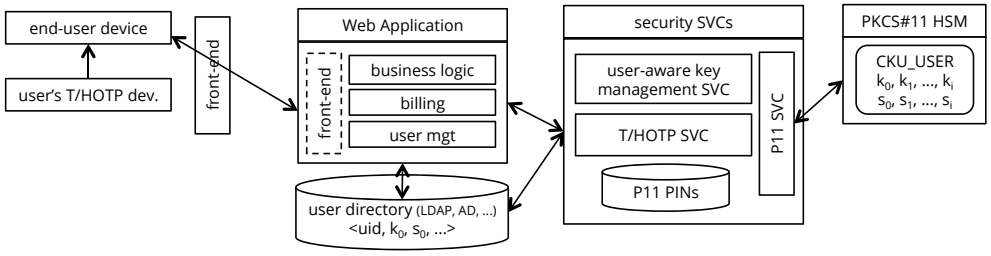


Figure 1. Cryptographic cloud service based on PKCS#11-enabled HSM with 2FA.

A significant portion of the architecture complexity in figure 1 is caused by PKCS#11. Therefore, for our architecture, we have decided to use the more modern OASIS Key Management Interoperability Protocol (KMIP) [OASIS 2016a] as one of the enablers of our proposal. KMIP is extensible and network oriented enabling the usage of crypto services by remote clients without the installation of any native or proprietary service. Currently, KMIP is supported by the main HSM vendors, including IBM and Gemalto. For our architecture, we have picked the kNET HSM from Kryptus, as it has KMIP as its native interface and secure code execution capabilities.

Our proposed architecture is then presented in figure 2. Again, the HSM stores both users private keys $(k_0, k_1, ..., k_i)$ and TOTP or HOTP seeds $(s_0, s_1, ..., s_i)$, however,

 © 2017 Sociedade Brasileira de Computação

keys are owned by end users and not by a device role. This fundamental difference eliminates the need of a user-aware key management service. And because of its secure code execution capabilities, we could move the T/HOTP authentication service into the HSM allowing atomic (signature) transactions to be dispatched directly from the client's endpoint. For that, we used the extensibleness capabilities of KMIP to allow for 2FA credentials on the remote signature operation. With those modifications, the front-end served by the web application can now communicate directly with the HSM, reducing the exposition and attack surface of the documents to be signed. The result is a design with a smaller footprint and improved security, as argued in the next section.

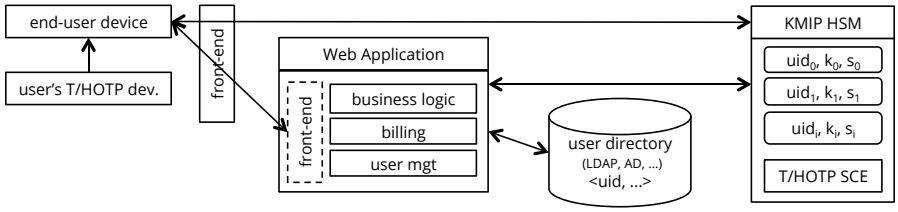


Figure 2. Cryptographic cloud service based on KMIP-enabled HSM with secure code execution with 2FA.

2.2. Security and Performance Analysis

The kernel of the proposed improvement is to use the secure execution capabilities of the target device to amplify the trustworthiness of the 2FA scheme, enabling its execution in a tamper-proof environment, directly tied to the user-aware key management system KMIP. Our proposal outperforms or at least equals the canonical P11 approach on most of the attack surface:

- **Attacks to the web application and its served front-end** are those intended to deny or to dispatch unauthorized transactions, replace documents to be signed, or compromise the confidentiality of user documents. In our proposal, document privacy is easily achievable as the document is not sent to the web application, instead, is processed directly (or its hash) by the HSM, dispatched from the front-end. Unauthorized transactions are not possible because the HSM signature is atomic. Secure code execution capabilities hinder attacker with physical access from tampering with the authorization code and secure loading prevents attacks on software load time. Finally, the front-end can be served from a host separated from the web application, allowing for further data-flow segregation.
- **Attacks to the backend authorization service** are those intended to allow for usage of nonauthorized keys, bypass of 2FA, T/HOTP seed replacement, user's document loss of confidentiality. Because of the lack of user awareness, a P11 HSM will allow for the usage of any key after a successful call to the C_Login function. On an online attack, once logged, the adversary will be able to use any user private-key. Our proposition completely removes the backend authorization service. Moreover, on the P11 version, the CKU_USER PIN must be available on the server.
- **Attacks on the user directory** are those intended to violate permissions, gain access to a user account or compromise user privacy. While in P11 proposition, web app users are tied to T/HTOP users and keys (k_i, s_i) through the directory,

© 2017 Sociedade Brasileira de Computação

database modifications can actually bypass 2FA or even worst, allow for private key operations. In our proposition, the bond among users and their keys is native and executing on the HSM tamper proof environment.

- **Attacks to the end user device** are those intended to deny a transaction, replacing a document to be signed, or acquiring the 2FA token to use in another operation. Both proposals perform similarly;
- **Attacks to the HSMs** are those intended to violate the device's security policy either by denying transactions, extracting keys or allowing unauthorized operations. Both proposals perform similarly.

One argument against our proposed architecture could be the performance impacts. However, for the target device model, where each RSA 2048 private-key operation takes 0.3 milliseconds, the T/HOTP operation adds only 0.1 milliseconds, with that, the performance stays over 2,000 transactions per second, aligned with the performance of PKCS#11 HSMs.

3. Related Work

Apple's Cloud Key Vault [Krstić 2016] similarly uses HSMs to store user's secrets such as passwords and credit card numbers; the user connects directly to the HSM, authenticating with their device PIN, to fetch the stored data. However, it does not support 2FA and does not support executing cryptographic operations with the keys stored in the HSM.

4. Conclusion and Further Work

We presented in this paper a security architecture for cryptographic services served over the cloud that leverages on KMIP protocol, HSM security, and secure code execution for the protection of 2FA logic. Our threat analysis shows improved security with minimal performance penalty. As the next step, we plan to release the source code of our experiment.

References

Krstić, I. (2016). Behind the scenes with iOS security. Presented at Black Hat 2016.

M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., and Ranen, O. (2005). HOTP: An HMAC-based one-time password algorithm. RFC 4226, RFC Editor.

M'Raihi, D., Machani, S., Pei, M., and Rydell, J. (2011). TOTP: Time-based one-time password algorithm. RFC 6238, RFC Editor.

National Institute of Standards and Technology (2001). FIPS 140-2: Security requirements for cryptographic modules.

OASIS (2016a). Key management interoperability protocol specification version 1.3.

OASIS (2016b). PKCS #11 cryptographic token interface base specification version 2.40.

 © 2017 Sociedade Brasileira de Computação

Eduroam Testbed: Um Ambiente Real para Experimentação em AAA*

Edelberto F. Silva[1], Luciano F. da Rocha[2], Javier R. Q. Ancieta[3], André Forigato[2], Anderson Almeida[2], Jonathan Welley[2]

[1]Universidade Federal de Juiz de Fora - UFJF/DCC
[2]Rede Nacional de Ensino e Pesquisa - RNP
[3]Inictel-Uni, Peru

Abstract. *This work shows the first worldwide initiative to offer a RADIUS and 802.1X infrastructure for experimentation. Based on eduroam federated environment, it will be possible to validate settings through virtual machines geographically distributed. Moreover, this real testbed environment allows its integration with other testbeds or simulation environments which has been promising to the research.*

Resumo. *Este trabalho apresenta a primeira iniciativa mundial de oferta de uma infraestrutura AAA baseada no protocolo RADIUS e 802.1X para experimentação. Baseado no mesmo modelo ofertado no ambiente federado de autenticação eduroam, será possível validar configurações e instalações através de máquinas virtuais geograficamente distribuídas. E, muito além disso, o ambiente de experimentação real permite a integração com outros ambientes de experimentação ou simulações, tornando-se promissor à pesquisa.*

1. Introdução

O protocolo RADIUS (*Remote Authentication Dial-In User Service*) [Rubens et al. 2000] oferece a comunicação entre cliente e servidor para AAA (Autenticação, Autorização e *Accounting*). Quando utilizado em conjunto com o IEEE 802.1X [Smith et al. 2003], o RADIUS é capaz de realizar autenticação segura a dispositivos conectados à rede sem fio. O caso de maior sucesso desta solução é o projeto eduroam[1] [Wierenga et al. 2015]. O eduroam fornece acesso seguro e transparente para usuários de instituições parceiras com base no conceito de federação, permitindo que o usuário se autentique à rede em qualquer instituição parceira utilizando suas credenciais únicas a partir do seu provedor de identidade de origem, facilitando o intercâmbio entre estudantes e pesquisadores.

Com a motivação da oferta de uma infraestrutura real baseada no caso de sucesso do eduroam, este trabalho apresenta o *eduroam testbed*. Este ambiente se apoia nos conceitos citados, de autenticação federada sobre o protocolo RADIUS e 802.1X, para criar um ambiente de avanços e experimentações a pesquisadores. Com o *testbed* é possível realizar desde validações práticas e técnicas até maiores avanços em pesquisa sobre conceitos de GId (Gestão de Identidade). Além disso, é possível utilizar o ambiente para validações de desempenho e escalabidade de soluções baseadas nos protocolos, ou como

*edelberto@ice.ufjf.br, jquinto@inictel-uni.edu.pe, luciano.rocha@rnp.br
[1]http://www.eduroam.org.br

 © 2017 Sociedade Brasileira de Computação

fonte de autenticação a serviços ofertados por outras infraestruturas de GId. O ambiente proposto tem como público alvo pesquisadores e profissionais da área de TI interessados em testar e desenvolver soluções utilizando os protocolos RADIUS e 802.1X.

O artigo está organizado em 5 seções. Além desta seção introdutória, a Seção 2 apresenta a infraestrutura, na Seção 3 são descritas as possibilidades de utilização do ambiente, a Seção 4 relata o *status* atual de utilização do ambiente, e a Seção 5 conclui este trabalho.

2. Infraestrutura

A atual configuração do *eduroam testbed* pode ser vista na Figura 1, onde há a interconexão entre os servidores RADIUS. Tem-se os níveis: local, federação e confederação. O nível local é referente ao IdP (Provedor de Identidade) e SP (Provedor de Serviço) da instituição[2], Atualmente tem-se 2 servidores no Brasil, 1 no Peru e 1 no México. Os servidores no Brasil estão na UFJF e na RNP, já os servidores do Peru se encontram todos no INICTEL-UNI, em Lima, e o servidor local da instituição no México na NREN CUDI. No nível da Federação tem-se 3 servidores, representando exatamente os 3 países, e no nível da confederação 2 servidores representam a ligação entre os países, sendo 1 no Brasil e outro no Peru. Cada servidor RADIUS local está conectado diretamente a seu servidor federado, utilizando um canal de comunicação TCP/TLS seguro através do chamado `RadSec`, e cada servidor no nível da federação se conecta a dois servidores confederados por meio de UDP RADIUS[3]. Os softwares e versões são: `FreeRadius 3.0.15` nos níveis local e confederados, e `radsecproxy 1.6.9` para o nível da federação.

3. Possibilidades

Como já mencionado, tanto aqueles vinculados à academia, como técnicos vinculados a projetos de implantação de soluções AAA em suas instituições, se beneficiarão do ambiente *eduroam testbed*. Nesta seção veremos algumas possibilidades classificados em duas categorias: técnica e científica. Estas duas categorias são descritas nas subseções a seguir.

3.1. Técnica

A primeira possibilidade de aplicação é a do **treinamento profissional**. O treinamento profissional visa auxiliar profissionais da área de TI que necessitem de treinamento para a implantação do serviço eduroam em sua organização. Com a utilização do *eduroam testbed* as validações de roteiros práticos, ofertados por centros de treinamento das NRENs (*e.g.* ESR/RNP), poderão se apoiar nessa infraestrutura real e distribuída, assim como a aplicação destes roteiros no formato EAD. A segunda possibilidade é a **utilização do ambiente pelos setores de TI das entidades** que já possuem o serviço eduroam. Este ponto é importante sob o ponto de vista da necessidade do conhecimento, testes, desenvolvimento e validações de configuração. Para ilustrar um cenário onde esta necessidade surge, tem-se a atualização de *software* do serviço por uma instituição já integrante da federação eduroam. O *eduroam testbed* pode ser usado para validar toda a configuração

[2]O IdP refere-se ao diretório de credenciais de usuários e o SP é o servidor RADIUS. Cada membro da federação em nível local possui e oferta seu IdP e SP.

[3]O UDP RADIUS é baseado na comunicação UDP com troca de mensagens em MD5 a partir de um segredo compartilhado.

 © 2017 Sociedade Brasileira de Computação

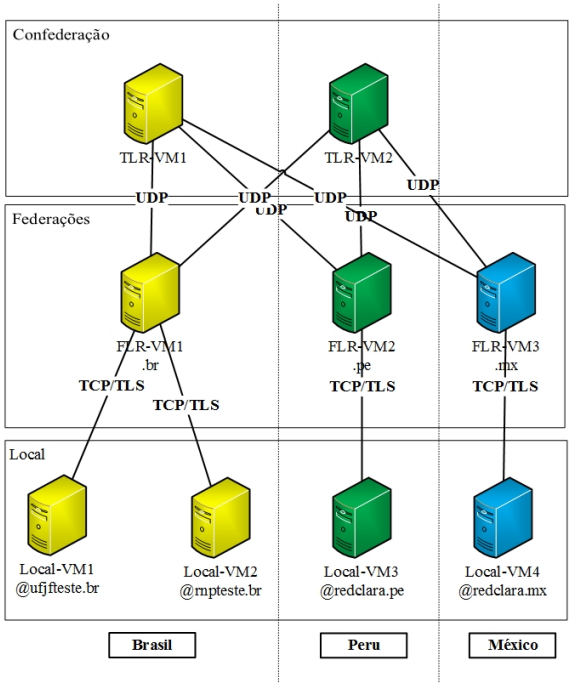


Figura 1. Infraestrutura de conexão do *eduroam testbed*.

antes de que essas atualizações sejam efetivamente implantadas no ambiente real e de produção do serviço na instituição. Uma terceira possibilidade é a utilização do ambiente *eduroam testbed* para **validação de *softwares*** para o serviço. É interessante realizar testes e validações sob ferramentas e soluções que já existem atualmente no ambiente eduroam, por exemplo, como o *software* de AAA `FreeRadius`, ou a verificação da possibilidade de incorporação de novas soluções ou ferramentas voltadas para, por exemplo, monitoramento ou configuração automatizada de servidores.

3.2. Científica

Diversas outras possibilidades surgem no campo científico, onde é possível visualizar mais do que apenas procedimentos técnicos ou validações de *software*, mas também o desenvolvimento de técnicas para o ambiente distribuído AAA do *testbed eduroam* ou usufruindo dele. Brevemente são citadas três linhas de pesquisa atualmente em curso e que utilizam a infraestrutura do *eduroam testbed*. Duas pesquisas têm utilizado o ambiente como fonte de autenticação e oferta dos atributos dos usuários de uma federação. Esta pesquisa é volta à autenticação e autorização em SDN com base em controle de acesso baseado em atributos, risco e adaptativos. Já uma terceira pesquisa tem verificado a possibilidade do desenvolvimento de uma solução de descoberta de serviço (*Discovery Service*) para o ambiente federado sob o `FreeRadius` do eduroam. Investigações sobre segurança na infraestrutura ofertada é outra alternativa em potencial. Como exemplo, tem-se a avaliação do uso de certificados digitais para federações acadêmicas, assim como o desempenho de outros protocolos, como o `RADIATOR`[4]. Outa linha de pesquisa é a verificação de escalabilidade dos protocolos envolvidos no serviço. Tais avaliações só são possíveis em uma infraestrutura AAA com servidores distribuídos geograficamente, tornando o *eduroam testbed* uma excelente alternativa para tais pesquisas.

[4]`https://www.open.com.au/radiator/`

© 2017 Sociedade Brasileira de Computação

Para concluir esta descrição, acredita-se que o *eduroam testbed* possa ser incorporado a outros ambientes de experimentação, como o FIBRE[5] ou o GIdLab[6]. No FIBRE o *eduroam testbed* pode ser tratado como uma ilha que forneça autenticação para soluções propostas pelos cientistas que utilizam o ambiente. De forma semelhante, o GIdLab pode se beneficiar de uma nova fonte distribuída e real que pode ser incorporada ao seu ambiente como um novo método de autenticação, possibilitando que desenvolvedores incorporem autenticação RADIUS e 802.1X às suas propostas de gestão de identidade. Outros ambientes de experimentação e desenvolvimento podem propor a integração e utilização do ambiente *eduroam testbed* como sua fonte de autenticação.

4. Utilização Atual

No Brasil o ambiente foi utilizado para apoiar a evolução do serviço eduroam no ano de 2017. Esta atualização contou com a implantação do serviço na última versão estável dos *softwares* `FreeRadius`, e do Sistema Operacional `Ubuntu 16.04 LTS`. Além disso, tem servido como base para o estudo e desenvolvimento de outras ferramentas que auxiliarão tecnicamente na manutenção do ambiente de serviço do eduroam, como a solução de instalação e configuração supervisionada Ansible[7]. De forma parecida, o Peru utilizou o ambiente para estudo e implementação da nova versão do `FreeRadius`, a `3.0.15`, servindo como base à utilização do `RadSec` para comunicação entre as instituições e a federação (algo novo neste momento). Como comentado na Seção 3.2, está em andamento entre Brasil e Peru para utilização de controle de acesso sobre SDN utilizando a infraestrutura do *testbed*. No momento, o México tem utilizado o *testbed* para apoiar instituições na utilização do `RadSec` com o objetivo de adicionar mais segurança aos serviços oferecidos a seus usuários.

5. Conclusões

Foi apresentada a primeira iniciativa mundial de um ambiente real e geograficamente distribuído em AAA RADIUS e 802.1X voltado à pesquisa e desenvolvimento de soluções em GId. Espera-se que a iniciativa seja adotada por outras NRENs de outros países, além dos que atualmente participam (*i.e*. Brasil, Peru e México). O ingresso de outros países como parte do projeto irá facilitar tanto a disseminação quanto a utilização por parte de seus pesquisadores.

Agradecimentos

Agradecemos à RNP e RedClara pelo apoio ao desenvolvimento deste trabalho.

Referências

Rubens, A., Rigney, C., Willens, S., and Simpson, W. A. (2000). Remote Authentication Dial In User Service (RADIUS). RFC 2865.

Smith, A., Zorn, G., Roese, J., Ph.D., D. B. D. A., and Congdon, P. (2003). IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines. RFC 3580.

Wierenga, K., Winter, S., and Wolniewicz, T. (2015). The eduroam Architecture for Network Roaming. RFC 7593.

[5] `http://fibre.org.br/`
[6] `https://gidlab.rnp.br/`
[7] `https://www.ansible.com/`

 © 2017 Sociedade Brasileira de Computação

Avaliação do Custo de Autenticação em VANETs

Ivo Carlson R. Barros, Artur Souza, Guilherme Maia, Leonardo B. Oliveira

[1]Universidade Federal de Minas Gerais (UFMG)

{ivo.carlson, arturluis, jgmm, leob}@dcc.ufmg.br

Resumo. *O sucesso das chamadas redes veiculares (VANETs) depende da concepção de novas soluções de autenticação entre veículos. Isso porque muitas das aplicações de VANETs demandam extrema eficiência na comunicação, o que dificulta a aplicação de soluções tradicionais. Frente a essa demanda, buscamos neste trabalho avaliar os custos atrelados à autenticação em VANETs em cenários reais. Para isso, simulamos uma VANET utilizando dados da cidade de Colônia, na Alemanha, e medimos o número médio de autenticações que cada veículo realiza ao longo do seu trajeto. Nossos resultados indicam que um veículo realiza aproximadamente 3 autenticações por segundo, porém, esse valor varia consideravelmente devido a variações na densidade da rede.*

1. Introdução

Muito já foi feito em busca da consolidação das redes veiculares ad hoc (VANETs). Avanços nessa área vão desde a definição de padrões de comunicação até a proposição de diversas aplicações pioneiras, exclusivas a esses sistemas [Wangham et al. 2014]. Apesar desses avanços, porém, muitos desafios para a consolidação dessas redes permanecem sem solução. Um desses desafios está em como proporcionar a esse contexto uma infraestrutura de autenticação e autorização, componente fundamental para a gestão de identidades [Silva et al. 2015]. Infelizmente, peculiaridades dessas redes como a alta mobilidade dos nós e aplicações sensíveis ao tempo tornam soluções tradicionais ineficazes nesse contexto [Wangham et al. 2014, Al-Momani et al. 2015].

Diante desse desafio, diversos trabalhos anteriores propõem novas soluções de autenticação, específicas para o contexto das VANETs. Dentre as soluções propostas, destacam-se o trabalho pioneiro de Raya e Hubaux [Raya and Hubaux 2007], soluções baseadas em infraestrutura de acostamento [Jiang et al. 2017], soluções baseadas em assinaturas de grupo [Shao et al. 2016] e autenticação baseada em propriedades do canal de comunicação [Al-Momani et al. 2015]. Grande parte dessas novas propostas de soluções surge em resposta à dificuldade de se propor soluções de custo adequado para a alta demanda por eficiência das VANETs. Nessas redes, o custo de mecanismos de autenticação se torna uma preocupação em particular devido à alta densidade da rede, a extrema mobilidade dos nós e a agilidade exigida por algumas das suas aplicações [Raya and Hubaux 2007].

Neste trabalho, avaliamos uma dimensão do custo computacional atrelado à autenticação em VANETs. Mais detalhadamente, estimamos o número médio de autenticações que um veículo realiza ao participar em uma VANET em um cenário urbano. Para isso, simulamos uma VANET utilizando o simulador Veins[1] e um *trace* da cidade de Colônia,

[1]http://veins.car2x.org/

 © 2017 Sociedade Brasileira de Computação

na Alemanha[2] (Figura 1). Acreditamos que o volume de autenticações realizadas serve como base para a avaliação da viabilidade de soluções de autenticação e como ponto de partida para avaliações de eficiência mais aprofundadas. As seções seguintes apresentam detalhes da simulação (seção 2) e os resultados obtidos (seção 3), juntamente com as nossas análises e conclusões (seção 4).

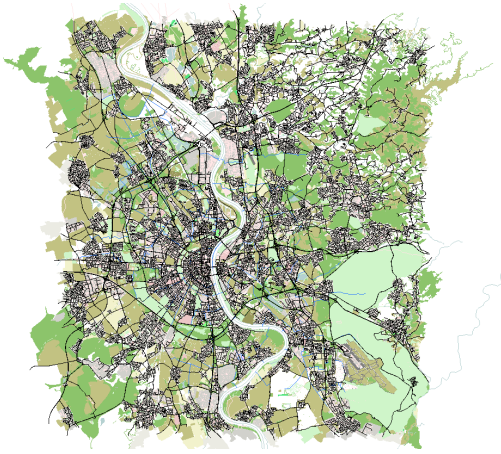


Figura 1. Trace de Colônia

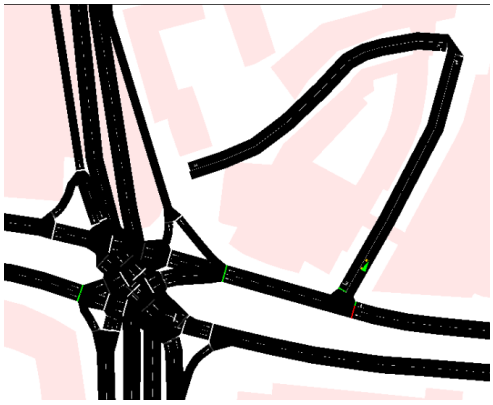


Figura 2. Simulação no SUMO

2. Simulação

Utilizamos o simulador Veins versão 4.4 em nossas simulações. O Veins é um sistema que integra duas ferramentas: o simulador de redes OMNeT++ [3] e o simulador de redes de mobilidade urbana SUMO [4]. A simulação foi executada em uma versão modificada do *trace* de Colônia que foi utilizada para diminuir o tempo total gasto na simulação. Essa modificação utiliza as entradas do *trace* original no intervalo de 6:00 às 7:00 da manhã e conta com aproximadamente 44.000 veículos.

Para a simulação, foi implementado um algoritmo que combina funções do OMNeT++ e do Veins para contar o número de vezes que dois veículos se encontram. Para contar o número de encontros, os veículos constantemente enviam mensagens que contêm sua identidade em *broadcast*. Os veículos em um raio de alcance de aproximadamente 200m recebem essas mensagens e salvam a identidade do emissor em uma tabela hash. Essa estrutura é utilizada a fim de evitar a recontagem de veículos.

Executamos a simulação utilizando nosso *trace* modificado. O cenário de simulação consiste das estradas que compõem a rede e as rotas possíveis. Os arquivos utilizados pelo SUMO informam o instante em que o veículo entra na simulação e as suas coordenadas nesse momento, a Figura 2 mostra um veículo simulado no SUMO. O OMNeT++ é responsável por especificar o comportamento dos veículos que compõem a rede.

3. Resultados

A Figura 3 mostra a distribuição do número total de encontros entre veículos. Nota-se que existe uma grande variação no número de encontros: alguns veículos tiveram apenas

[2]http://sumo.dlr.de/wiki/Data/Scenarios/TAPASCologne
[3]https://omnetpp.org/
[4]http://sumo.dlr.de/index.html

© 2017 Sociedade Brasileira de Computação

um encontro enquanto outros se encontraram com mais de 800 outros veículos. Um dos motivos para essa discrepância é, naturalmente, a diferença entre os trajetos realizados pelos veículos. Os que passam por regiões mais movimentadas apresentam um número maior de encontros. Um segundo motivo para essa diferença está no tempo que cada veículo permanece ativo na simulação, alguns veículos participam da rede por apenas alguns minutos, enquanto outros ficam ativos ao longo de toda a simulação.

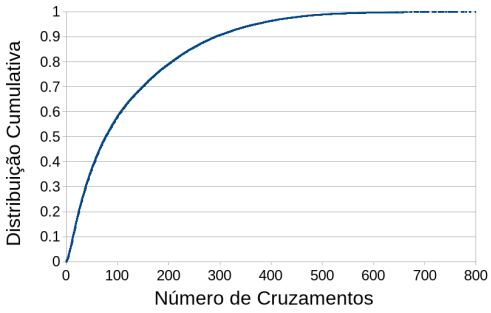


Figura 3. Distribuição cumulativa do número de autenticações.

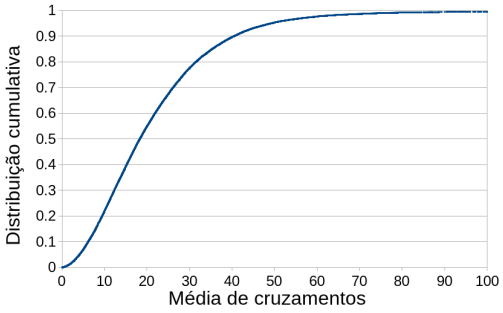


Figura 4. Distribuição cumulativa do número de autenticações por minuto.

Para compensar pelas diferenças causadas pelo tempo ativo do veículo na simulação, a Figura 4 apresenta o número de encontros normalizado pelo tempo que o veículo permaneceu ativo. Nota-se que a curva da distribuição ficou mais suave, indicando que a distribuição se tornou mais uniforme, porém, ainda há uma diferença considerável no número de encontros por veículos. Nos casos mais extremos, os veículos chegam a apresentar até 100 encontros por minuto.

As Figuras 5 e 6 mostram a mediana do número de encontros por veículo e de encontros por segundo por veículo, desconsiderando *outliers* (mais de dois desvio-padrões da mediana). A mediana indica que os veículos se encontram com aproximadamente 71 outros veículos ao longo de todo o seu trajeto e com aproximadamente 1 veículo a cada 3 segundos (0,3 veículo por segundo). Note que, mesmo removendo os outliers, os valores continuam apresentando variações consideráveis em torno da mediana.

Para estimar o número de autenticações realizadas, consideramos que cada veículo envia mensagens periódicas a cada 0,1 segundo [Engoulou et al. 2014]. Combinando esse dado com os resultados obtidos, encontramos que um veículo recebe, e deve autenticar, aproximadamente 3 mensagens por segundo. Ressaltamos, porém, que esse valor considera apenas mensagens periódicas. Em um cenário real, um veículo recebe também mensagens de segurança (por exemplo, alertas de colisão) e, possivelmente, mensagens de aplicações gerais (por exemplo, aplicações de entretenimento). Portanto, esse valor provavelmente subestima o valor real, porém, apresenta um mínimo fiel para o número de autenticações realizadas.

4. Conclusão

O desafio de como prover autenticação e autorização permanece aberto em VANETs, principalmente devido aos rigorosos requisitos de eficiência e escalabilidade dessas redes. Para auxiliar na avaliação e concepção de novas soluções de autenticação, avaliamos neste

© 2017 Sociedade Brasileira de Computação

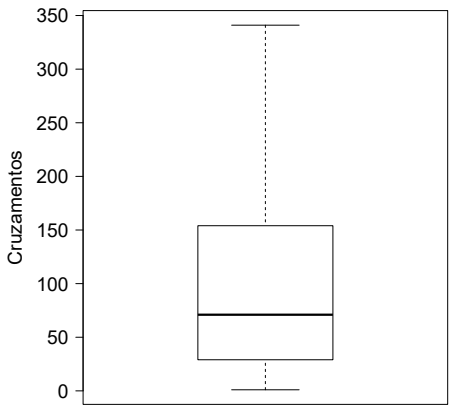


Figura 5. Média do número de cruzamentos

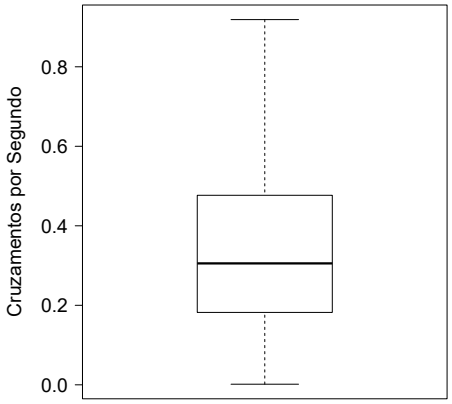


Figura 6. Média do número de cruzamentos por segundo

trabalho os custos atrelados à autenticação em redes veiculares. Precisamente, analisamos o número de autenticações que um veículo realiza em um cenário urbano típico. Nossos resultados revelam que um veículo realiza aproximadamente 3 autenticações por segundo, considerando apenas mensagens periódicas. Encontramos também que a densidade dessas redes varia bastante e, portanto, em alguns momentos os veículos podem precisar realizar um número muito maior de autenticações.

Referências

Al-Momani, A., Kargl, F., Waldschmidt, C., Moser, S., and Slomka, F. (2015). Wireless Channel-based Message Authentication. In *Vehicular Networking Conference (VNC)*.

Engoulou, R. G., Bellaïche, M., Pierre, S., and Quintero, A. (2014). VANET Security Surveys. *Computer Communications*, 44:1 – 13.

Jiang, W., Li, F., Lin, D., and Bertino, E. (2017). No One Can Track You: Randomized Authentication in Vehicular Ad-hoc Networks. In *Conference on Pervasive Computing and Communications (PerCom)*.

Raya, M. and Hubaux, J.-P. (2007). Securing Vehicular Ad Hoc Networks. *Journal of Computer Security*, 15(1):39–68.

Shao, J., Lin, X., Lu, R., and Zuo, C. (2016). A Threshold Anonymous Authentication Protocol for VANETs. *IEEE Transactions on vehicular technology*, 65(3):1711–1720.

Silva, E. F., Muchaluat-Saade, D., Fernandes, N. C., and Niterói, R. (2015). Modelagem do ACROSS: Um Arcabouço de A&A Baseado em Polıticas e Atributos para Organizações Virtuais. In *Workshop de Gestão de Identidades, SBSeg*.

Wangham, M. S., Nogueira, M., Fernandes, C. P., Paviani, O., and da Silva, B. F. (2014). Segurança em Redes Veiculares: Inovações e Direções Futuras. In *Minicursos: Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg)*.

 © 2017 Sociedade Brasileira de Computação

WRAC+ – III Workshop sobre Regulação, Avaliação da Conformidade, Testes e Padrões de Segurança

© 2017 Sociedade Brasileira de Computação

Caminhos para a Segurança: Normalização ou Adequação de Padrões

Carlos Roberto Viana[1], **Raphael Machado**[1,2]

[1]PPCIC - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca – CEFET-RJ

[2]Inmetro - Instituto Nacional de Metrologia, Qualidade e Tecnologia

carlos.filho@eic.cefet-rj.br, rcmachado@inmetro.gov.br

Abstract. *One of the advantages of using international standards and norms is to increase the safety levels of various systems and products. However, their adoption can be complex and technically and/or financially infeasible, leading to several agencies not adopting standardization. In Brazil, several organs in the most diverse spheres, edit, promulgate, adopt and regulate rules of the most varied forms and standards, being a simple adoption of international norms or even the standardization. This article seeks clarification on the adoption of international safety standards and their benefits, as well as an adaptation of standards to our reality.*

Resumo. *Uma das vantagens de se utilizar as normas e padronizações internacionais, é a elevação dos níveis de segurança de vários sistemas e produtos. No entanto, a sua adoção pode ser complexa e inviável técnica e/ou financeiramente, levando vários organismos a não adoção de padronização alguma. No Brasil, vários órgãos nas mais diversas esferas, editam, promulgam, adotam e regulam regras das mais variadas formas e padrões, podendo ser a simples adoção das normas internacionais ou mesmo a padronização das mesmas. Este artigo procura esclarecer sobre a adoção de normas internacionais de segurança e seus benefícios, bem como a adaptação destas normas a nossa realidade.*

1. Introdução

Nos dias de hoje, em que se observa a proliferação de sistemas de informação, dispositivos inteligentes e equipamentos dotados de software embarcado, garantir a segurança de informação e de sistemas computacionais tornou-se tão importante quanto garantir a proteção de ativos físicos. Um importante caminho para garantir níveis adequados de segurança é a adoção de padrões, normas e recomendações consolidadas junto à comunidade técnica internacional. Ao mesmo tempo, por se tratar de uma área nova e de alta complexidade tecnológica, a adoção de padrões internacionais na área de segurança da informação tende a ser um processo complexo e custoso, demandando um cuidadoso estudo de viabilidade antes de sua concretização. No presente trabalho, realizamos uma análise das estratégias para a adoção de padrões de segurança da informação. Estudamos modelos de avaliação da conformidade na área de segurança da informação adotados no Brasil e no mundo, identificando as principais normas de segurança e estudando a forma como tais padrões têm sido implementados para diversos produtos e sistemas. O levantamento realizado evidencia que a adoção de uma norma internacional na área de

 © 2017 Sociedade Brasileira de Computação

segurança, pode elevar os custos, comprometendo a própria sustentabilidade de um mercado ou setor, especialmente, quando se trata de uma padronização compulsória, tipicamente associada a um programa de avaliação da conformidade concebido pelo Governo. Argumentamos, no entanto, que mesmo que estes mercados e setores não possam adotar integralmente normas internacionais de segurança, é possível seguir estratégias baseadas na adoção parcial ou adaptada de tais normas, sempre buscando garantir a harmonia conceitual e a compatibilidade técnica em relação ao padrão original. Finalmente, validamos essa ideia por meio da análise do Programa de Avaliação da Conformidade aplicado aos equipamentos da Infraestrutura de Chaves Públicas brasileira, mostrando que, embora baseado em regulamentação técnica nacional, ele está tecnicamente alinhado ao padrão FIPS (Federal Information Processing Standard) 140-2 que, por sua vez, é reconhecido como padrão internacional ISO (International Organization of Standardization) /IEC (International Electrotechnical Commission) 19790.

2. Padronizações

Conforme descrito pela ABNT (Associação Brasileira de Normas Técnicas) a

> "Normalização, tal como definida pelo Vocabulário Geral da ABNT, é uma atividade que estabelece, em relação a problemas existentes ou potenciais, prescrições destinadas à utilização comum e repetitiva com vistas à obtenção do grau ótimo de ordem, em um dado contexto. Em particular, a atividade consiste nos processos de elaboração, difusão e implementação de normas. A normalização proporciona importantes benefícios, melhorando a adequação dos produtos, processos e serviços às finalidades para as quais foram concebidos, contribuindo para evitar barreiras comerciais e facilitando a cooperação tecnológica" [**?**].

As normatizações surgiram principalmente das normas militares, pois diversos países tinham muitas dificuldades para o compartilhamento de munições. Com isto, foram criadas as normas militares de qualidade como as do Canadá – Série Z 299, França – AFNOR X 50-110, Alemanha – DIN 55-355, Reino Unido – Série BS 5750, EUA – ANSI/ASQC Z-1.15 etc e OTAN – Série AQAP [**?**].

Padronizações são referências necessárias para assegurar o cumprimento de requisitos. Enquanto a normalização é a tecnologia consolidada, que permite confiar e reproduzir algo por várias vezes [**?**], as normas são documentos que estabelecem uma conformidade aprovada por uma instituição reconhecida, que passa a fornecer regras, orientações ou características para a obtenção de resultados esperados através da utilização de determinados procedimentos [**?**].

Economicamente a padronização está relacionada à produtividade, pois padrões protegem a segurança, facilitam o comércio, a interoperabilidade e ajudam no desenvolvimento. Por outro lado, a sua utilização de maneira incorreta, reduz opções, competitividade, podendo criar barreiras técnicas [**?**]. A sua adoção pode elevar os custos devido aos ensaios necessários, levar a criação de novos procedimentos e diminuir o número de laboratório que tenham capacidade para a sua realização [**?**].

3. Benefícios da Adoção de Padrões

Padrões por sua essência, são instrumentos de governança que podem interferir na qualidade de vida de uma população, sendo empregados pelos setores privados para criar

 © 2017 Sociedade Brasileira de Computação

incentivos para o mercado, estimulando a inovação e desenvolvimento tecnológico. Para o setor público, permite a transparência, fornecendo os requisitos fundamentais que serão utilizados como referencial para a regulamentação de leis. Neste contexto, podemos verificar que os padrões são ferramentas eficientes, fornecendo requisitos para segurança, processo, desempenho, política e legislação. Portanto, podemos verificar que existe uma relação entre padronizações e formulação de políticas, que leva a utilização de normas internacionais como uma boa prática de governança [**?**].

A ampliação do comércio mundial e o avanço da ciência, formam condições ideais para a elaboração de norma técnicas, mas essas por si só não são condições necessárias para a adoção. Com a maior relação econômica entre os países e o envolvimento de produtos de maior complexidade técnica, surge a necessidade de compatibilização de inúmeras estruturas técnicas e sociais [**?**]. Para empresas, existem benefícios diretos na adoção da padronização, tais como: a racionalização das operações internas que proporciona a agilização dos processos internos, redução do tempo de atividade, diminuição do desperdício, inovação e ampliação de operações. Isso permite que as empresas gerenciem seus produtos de forma mais eficaz, e consequentemente, criando novos mercados sejam estes internos ou externos [**?**]. O progresso tecnológico pode ser mensurado pela produtividade que representa a eficácia da utilização dos recursos, economia de escala, concorrência e a disseminação da tecnologia, provando com isto que a padronização desempenha um papel significativo neste processo [**?**]. O desempenho econômico, e a sua relação com a padronização é um tema pesquisado em diversos países como Alemanha, Reino Unido, Canadá, França e Austrália, baseado na associação do crescimento econômico e dos padrões. Os padrões são apontados como um bem público, que estão disponíveis a um custo baixo para todos os produtores e consumidores. Esta informação é o modelo para o aperfeiçoamento, segurança, interoperabilidade e comércio, que são criadas por comitês de especialistas com o objetivo do melhoramento de práticas nacionais e internacionais. Sendo assim, os padrões são uma ferramenta fundamental para a divulgação do conhecimento [**?**].

4. Cenário Brasileiro

No Brasil não existe uma lei específica de segurança da informação e não existe um único órgão responsável pela ordenação destas. A Receita Federal do Brasil dispõe sobre a utilização de certificado digital entre esta e a sociedade. O Banco Central edita normas sobre a segurança da informação que deve ser adotada pelas instituições financeiras em operação no Brasil. O Conselho Nacional de Justiça edita normas para os entes do judiciário. Na história da humanidade, atos para a segurança da informação são adotados para a proteção da informação. Atualmente podemos notar vários casos de vazamento de informações. O Senado Federal do Brasil através de uma CPI (Comissão Parlamentar de Inquérito), demonstrou que vários órgãos de diferentes esferas estavam vulneráveis a ataques [**?**]. O Tribunal de Contas da União, adota as normas ISO e IEC, porque além de reconhecidas pela entidade normalizadora Brasileira, a ABNT, são mundialmente reconhecidas e utilizadas por Entidades Fiscalizadoras Superiores, órgãos de governo, empresas públicas e privadas nacionais e internacionais [**?**].

4.1. Casos

O Decreto n.º 3.505/2000 que instituiu uma política de segurança da informação no âmbito dos órgão públicos federais, destaca a compreensão das pessoas sobre os requisi-

 © 2017 Sociedade Brasileira de Computação

tos da segurança da informação, bem como a adoção de normas para a efetivação destas, tais com: padrões de recursos criptográficos, normas para comercialização criptográficos, normas para a confidencialidade de dados, normas para a emissão de certificados de conformidade e normas para a elaboração de sistema para com isto garantir a interoperabilidade, níveis de segurança e disponibilidade dos dados [**?**].

5. Modelo de Adoção de Padrões

Vimos que a adoção integral de padrões pode trazer desafios do ponto de vista da viabilidade técnica e econômica. Nesta seção, estudamos tais desafios e discutimos a possibilidade da adoção parcial ou adaptada de um padrão, analisando um caso concreto de tal estratégia, que são os padrões para equipamentos da ICP-Brasil (Infraestrutura de Chaves Públicas Brasileiras) [**?**].

5.1. Adoção integral de normas internacionais

Um documento que preveja a utilização comum, que estabeleça regras e diretrizes de um modo repetitivo, que contenha características para atividades ou seus resultados é um padrão. A adoção de padrões internacionais pode ser tida como empregada, quando normas regionais ou nacionais são idênticas ou modificadas em sua relação. Adotar normas internacionais no âmbito nacional é muito intricado, se regras ou tradições, nacionais relacionadas a estas não convergirem. Toda energia deve ser direcionada para a redução de desvios para um fator mínimo. Se a versão internacional for simplesmente uma reedição será bastante fácil discernir os desvios técnicos do padrão original [**?**].

5.2. Common Criteria

A certificação em TI da Rússia vinha desde 1995 com uma certificação similar ao Orange Book dos Estados Unidos, e a partir de 2002 adotando o Common Criteria como metodologia, e desde 2012 vem avançando nesta certificação, criando e aprovando novos perfis de proteção, e com isso aumentando a eficiência na detecção de vulnerabilidades em 50%, conforme indica o site do Serviço federal de Técnica e Controle de Exportação da Rússia. As primeiras certificações seguiam os TOE de outros países, ou seja, acompanhavam as solicitações do padrão internacional, no entanto, estas foram sendo substituídas gradualmente por um novo padrão Russo, obrigando os desenvolvedores a manter os produtos certificados no seu ciclo de vida, e que acesso aos códigos fontes dos produtos fosse fornecido aos laboratórios deste país [**?**].

5.3. Adoção parcial ou adaptada de normas internacionais

Destacamos que a adoção "parcial" de uma norma internacional pode ser uma estratégia que viabiliza as melhores práticas de mercado, como ilustra o estudo de Caso do ICP-Brasil, que foi criada através da medida provisória Nº 2.200, com a finalidade de garantir a autenticidade, integridade e a validade jurídica de documentos e suas aplicações.

A ICP-Brasil é ainda responsável por adotar medidas necessárias e coordenar a implantação e o funcionamento desta, através do estabelecimento de políticas, critério e normas técnicas para o credenciamento de Autoridades Certificadoras (AC) e de Autoridades Registro (AR), além do estabelecimento de diretrizes e normas técnicas para as políticas e regras gerais, sendo o Instituto de Tecnologia da Informação a AC Raiz da

 © 2017 Sociedade Brasileira de Computação

ICP-Brasil [**?**]. Entre os documentos descritos, podemos verificar nos MCTs (Manuais de conduta técnica) de 1 a 11, referências ou a utilização de várias normas e padrões internacionais. Os MCTs servem de base para que os LEA (Laboratórios de Ensaios e Auditoria), possam avaliar e emitir laudos correspondentes, e que o ITI (Instituto Nacional de Tecnologia da Informação) homologue o objeto avaliado. Esta avaliação contém vários itens, e dentre eles podemos verificar a identificação de requisitos obrigatórios e opcionais de homologação e as discrepâncias encontradas [**?**].

Para validar os argumentos defendidos ao longo do texto, apresentamos um estudo em maior nível de detalhes a respeito do MCT 7, que é o manual que contém as orientações para homologação dos HSM (Hardware Security Modules). Uma análise desta publicação demonstra que o manual é aderente a diversos padrões internacionais, descrevendo os requisitos técnicos necessários para a homologação de módulos de segurança criptográficos, cujas premissas de segurança são derivadas do padrão FIPS 140-2 do NIST (National Institute of Standards and Technology), a qual abrange diferentes áreas para a implantação de um módulo criptográfico. Dentre os padrões adotados podemos destacar: ANSI (X9.31, X9.62, X9.80, X9.81-1, X9.82-1), IEC.(CISPR 22, CISPR 24, 60050–161), ISO/IEC 8825-1, FIPS (197, 800-38C, 46-3, 140-2, 186-2, 196, 198, 180-2), NIST (SP 800-17, SP 800-20, 800-38B) [**?**].

Podemos notar que diversos requisitos de segurança presentes no MCT 7 são derivados do padrão americano FIPS 140-2 publicado pelo NIST, alguns destes, adotados integralmente como o Requisito III.4.1. Outros são adotados de forma adaptada como a especificação dos métodos do módulo criptográfico no Requisito III.4.3 ambos presentes no item *4.4 Finite State Model* da referida norma internacional [**?**].

Dentre as áreas de atuação contidas no FIPS 140-2 e apresentada no MCT 7 podemos encontrar documentação do módulo criptográfico, identificação de portas e interfaces do módulo criptográfico, nível de identificação de papéis, serviços e autenticação do operador, descrição do modelo de estado finito, nível de segurança física, ambiente operacional, gerenciamento de chaves criptográficas, interferência e compatibilidade eletromagnética, auto-testes, garantia do projeto e mitigação de outros ataques [**?**].

Adicionalmente, o manual aborda outros temas como Algoritmos criptográficos obrigatórios presente no capítulo 2, interoperabilidade e restrição de substância nocivas [**?**].

6. Conclusão

Podemos verificar que a adoção de padrões traz enormes benefícios, sejam estes para empresas privadas ou órgãos públicos, pois melhoram o processo, reduzem o tempo, evitam o desperdício e promovem a melhoria de intercâmbio entre as nações. Ainda são instrumentos de governança, que podem interferir na vida de uma população, pois são ferramentas que fornecem requisitos para vários setores, inclusive aqueles relacionados à segurança. No Brasil, não há leis específicas que regulamentem as questões de segurança e padronizações. Diversos órgãos utilizam diferentes padrões de segurança, sejam estes oriundos de outras normas internacionais ou próprias. Com a criação dos MCTs, o ITI passa a adotar diversos padrões internacionais, principalmente o conjunto de normas do FIPS. Com a utilização dessas padronizações internacionais, e ainda com a adoção de padrões internos Brasileiros, os MCTs, passam a proporcionar os requisitos e os docu-

 © 2017 Sociedade Brasileira de Computação

mentos técnicos para a homologação de cartões criptográficos, leitoras de cartões, tokens criptográficos e software para assinatura digital dentre outros. Com isto, há um enorme salto na qualidade do conhecimento, apontando para um processo de normatização e padronização de segurança. Como estratégia, sugerimos como solução ideal, a adoção de uma padronização internacional para as questões de segurança. Na impossibilidade da sua aplicação total, deve-se adotar uma padronização parcial e, na impossibilidade desta última, que seja utilizada uma normatização nacional, parcial, ou adaptada para os quesitos de segurança .

 © 2017 Sociedade Brasileira de Computação

Testes de sobrecarga: uma avaliação sobre requisitos de Disponibilidade e Desempenho

Carlos A M S Teles[1], **Raphael Carlos Santos Machado**[1,2]

[1]PPCIC – Centro Federal de Educação Tecnológica Celso Suckow da Fonseca – CEFET-RJ

[2]Inmetro – Instituto de Metrologia, Qualidade e Tecnologia

{carlos.teles}@eic.cefet-rj.br, rcmachado@inmetro.gov.br

Abstract. *In the present paper we will evaluate how the protection to computer systems and information security has its relevance recognized progressively. With greater focus on confidentiality and integrity requirements, there is little advance in establishing availability and performance standards. We will analyze the information availability, performance and security requirements in Brazilian public notices, showing a moderate concern to set strict standards of availability and performance. We set definitions for such requirements, and we argue that they can be rigorously validated through systematic testing. We propose a testing environment that will allow us to perform availability and performance tests in overload scenarios.*

Resumo. *No presente artigo avaliaremos como a proteção a sistemas computacionais e segurança da informação tem sua relevância reconhecida progressivamente. Com maior foco nos requisitos de confidencialidade e integridade, observa-se que há pouco avanço no estabelecimento de padrões de disponibilidade e desempenho. Analisaremos os requisitos de disponibilidade, desempenho e segurança da informação em editais públicos brasileiros, demonstrando uma preocupação moderada em definir padrões rigorosos de disponibilidade e desempenho. Estabelecemos definições para tais requisitos, além de argumentarmos que estes podem ser rigorosamente validados mediante ensaios sistemáticos. Propomos um ambiente de testes que nos permitirá a realização de ensaios de disponibilidade e desempenho em cenários de sobrecarga.*

1. Introdução

Na atualidade, nosso tempo é consumido cada vez mais por novas tecnologias e pela internet, ou seja, um ambiente tecnológico. Este possui um conjunto de variáveis que podem impactar positiva ou negativamente as organizações. Não sendo um local seguro, ao guardar e agregar tantas informações, nos tornaram reféns, assim como se mostrou um local encantador para os hackers, com tanta informação pessoal e de empresas, circulando sem a devida segurança [Rezende 2008].

É neste contexto que a Segurança da Informação tornou-se relevante na maioria das organizações. Seu objetivo é assegurar a continuidade dos negócios e minimizar os danos causados ao prevenir e minimizar o impacto de incidentes de segurança. É influenciada por três propriedades principais: Confidencialidade para limitar o acesso à informação

 © 2017 Sociedade Brasileira de Computação

somente aos agentes autorizados; Integridade para preservar que as informações tenham suas características fiéis à sua origem e que qualquer alteração durante o processo tenha sido realizada com autorização e controle e; Disponibilidade para garantir que a informação esteja acessível aos autorizados a todo tempo que precise ser resgatada [von Solms 1998] [Sêmola 2003].

Em se tratando de Disponibilidade podemos medi-la através do cálculo: percentual entre o tempo funcional e tempo total de uso. É requerida em acordos de níveis de serviços (SLA - *Service Level Agreement*) [Santos and et. al. 2015] .

O Desempenho é responsável pela medição e disponibilização das informações sobre aspectos dos serviços e dispositivos de rede. Esses dados são utilizados para garantir que redes e sistemas computacionais operem em conformidade com o nível de serviço acordado (SLA) com seus usuários [Santos and et. al. 2015] .

Para garantir às três propriedades principais da Segurança da Informação fazemos uso das medidas de Disponibilidade e Desempenho para Sistemas e Redes de Informação.

Uma das técnicas para inutilizar estas propriedades são os Ataques. Nesta atividade, onde um atacante utiliza um computador ou vários, conectados à rede, para remover outro computador ou vários, de operação ou fazer sua conexão a uma rede conectada à Internet ficar indisponível, é feita a tentativa de obtenção de dados e podem causar falhas relacionadas à segurança, disponibilidade e desempenho em redes e sistemas. Pode acarretar perdas importantes para as organizações. Uma falha de desempenho em um sistema hospitalar, ocasionada em função de um ataque através de um programa malicioso que consome a CPU do sistema, pode causar a demora no atendimento das pessoas. Uma falha de disponibilidade em um sistema bancário, ocasionada em função de problema em banco de dados pode gerar a perda financeira para várias pessoas. Entretanto, estes problemas são difíceis de serem detectados.

Um tipo de ataque muito utilizado é o Ataque Distribuído de Negação de Serviço (DDoS - *Distributed Denial of Service*). Estes podem ser classificados: esgotamento de recursos e esgotamento de banda [Cert.br 2016] [Somal and Virk 2014].

O protocolo BGP (*Border Gateway Protocol*) pode auxiliar na proteção de um ataque, através de configurações nos roteadores, como políticas de rotas, sessões de BGP e outras configurações [Butler et al. 2010].

Além disso, um dos problemas identificados é que os atacantes modificam constantemente suas ferramentas para contornar os sistemas de segurança. E os pesquisadores modificam suas abordagens para detectar novos ataques [Mirkovic and Reiher 2004].

Em nosso artigo, mostraremos que existe uma relação direta entre os aspectos de Segurança, Disponibilidade e Desempenho. Demonstraremos tais relações através de testes de sobrecarga. Estes nos possibilitam estabelecer medidas para identificar futuros ataques e estabelecer limites para os Sistemas Computacionais e Redes. Nosso intuito, a partir de cenários pré-estabelecidos, através das análises de editais públicos, e resultados obtidos pelos testes de sobrecarga, é mensurar e configurar as aplicações de rede, de forma que no futuro tenhamos um padrão de configuração que possam sustentar os testes.

 © 2017 Sociedade Brasileira de Computação

2. Requisitos de Segurança, Disponibilidade e Desempenho

Em nosso artigo analisaremos as informações de Governo das diversas esferas (Municipal, Estadual e Federal). Assim, nossa base para extração de informações serão Licitações e documentos publicados pelos diversos órgãos destas esferas.

2.1. Casos dos Requisitos

Em nossa análise sobre os recursos através das Licitações, escolhemos aleatoriamente quatro editais disponíveis nos sites dos respectivos órgãos publicados entre 2015 e 2017.

Os órgãos selecionados foram: Prefeitura Municipal de Monte Carmelo, Senado Federal, Câmara dos Deputados Federais e o Tribunal Regional do Trabalho da 18a Região Goiás (TRT18).

O edital da Prefeitura de Monte Carmelo [de Monte Carmelo 2017] foi solicitada a Disponibilidade de 99,5% por mês. Não houve solicitação de medidas de Desempenho. Para Segurança houve a solicitação de uma Solução de proteção de rede com características de *Next Generation Firewall* (NGFW) ou *Unified Threat Management* (UTM) para segurança de informação perimetral.

O edital do Senado Federal [Federal 2016] foi solicitada a Disponibilidade de 99,8% por mês. Não houve solicitação de medidas de Desempenho. Para Segurança, foi solicitado um Serviço de Proteção Pró-ativo Anti-DDoS, com capacidade para identificar, comunicar e mitigar quaisquer tipos de ataques que utilizem indevidamente os recursos de rede em IPv4 ou IPv6.

O edital da Câmara dos Deputados [dos Deputados Federal 2015] foi solicitada a Disponibilidade de 99,44% por mês. Houve solicitação de medidas de Desempenho para taxa máxima de utilização de CPU e Memória em 60%. Para Segurança, solicitou-se um Serviço de Proteção contra Anti-DDoS, a partir da sinalização remota de *black hole* em anúncios BGP e utilização de *blacklist* para bloqueio de tráfego.

O edital do TRT18 [18a. REGIÃO 2015] foi solicitada a Disponibilidade de 99,60% por mês. Solicitou que as medidas de Desempenho tivessem taxa máxima de utilização de CPU e Memória em 70%. O único item de Segurança solicitado foi a utilização de DNSSEC (DNSSEC - *Domain Name System Security Extensions*) para os domínios já registrados no DNS (DNS - Domain Name System) primário do TRT18.

2.2. Análise das Licitações

Podemos resumir os editais assim: todos consideram os requisitos de Disponibilidade e Segurança, entretanto, apenas dois (Câmara dos Deputados Federais e TRT18) consideram Desempenho.

Cabe ressaltar que apesar dos requisitos de Segurança constarem em todos os editais, os mesmos não possuem padrão algum. Isto ocorre em função das múltiplas tendências tecnológicas que são lançadas no mercado constantemente [Leão et al. 2017].

Um dos motivos que identificamos para falta de padrão seria a inexistência de um escritório central e com base no Governo Federal, que aborde um padrão de indicadores na Licitação. Assim temos vários órgãos gerando documentos de boas práticas e manuais com informações que sobrepõem dentro das diversas esferas de Governo (Municipal, Estadual e Federal).

 © 2017 Sociedade Brasileira de Computação

O tamanho da solução solicitada nos editais, a inexistência ou falha de um entre os aspectos de Segurança, Disponibilidade e Desempenho são vistos como fatores de risco.

3. Testes para verificação de requisitos

Para demonstrar a relação entre Disponibilidade e Desempenho em ambientes tecnológicos sob a ótica de Segurança, montamos um ambiente de redes de computadores, onde utilizamos ferramentas e dispositivos para simular testes de sobrecarga.

3.1. Ferramentas

Para nossos testes escolhemos ferramentas para simular testes de sobrecarga, coletar, medir e publicar tais resultados.

Selecionamos o T50 [Brito 2017] como simulador para testes de sobrecarga. Podemos citar como suas principais características:

- Suporte a protocolos de rede TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*), ICMP (*Internet Control Message Protocol*) entre outros;
- Simulações com mais de 1.000.000 de pacotes por segundo;
- Simulação de testes de sobrecarga, ataques com esgotamento de recursos e inundação;
- Grande facilidade de parametrização para simulação dos testes.

As ferramentas utilizadas para coleta e disponibilização dos resultados de nossos testes foram o NMON/NMONCHART. O NMON [Griffiths 2017a] é um monitor de desempenho e deve ser compilado no computador que vai receber os testes de sobrecarga. E o NMONCHART [Griffiths 2017b] que é um script escrito em Korn Shell (ksh - *Korn Shell*) para converter os arquivos coletados pelo NMON em arquivos HTML (*Hypertext Markup Language*).

3.2. Ambiente

Em nosso ambiente de testes temos três dispositivos que são: um computador (atacante) gerando testes de sobrecarga, conectado a uma rede local através de um switch, e outro computador com um servidor HTTP em LINUX (atacado), também conectado ao switch.

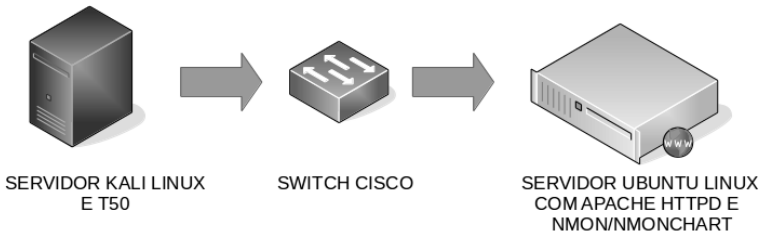


Figura 1. Dispositivos utilizados em testes de sobrecarga.

Os testes foram realizados com o software T50 configurado no servidor atacante e a coleta/disponibilização de resultados via NMON/NMONCHART no servidor atacado. Em nosso cenário temos algumas características como: conexão via porta GigabitEthernet; os testes de sobrecarga não indisponibilizaram o servidor atacado; a cada 10 segundos foi gerado tráfego com 5000000 de pacotes na porta 80; a monitoração iniciou antes e terminou após os testes.

 © 2017 Sociedade Brasileira de Computação

4. Resultados

Os resultados nas simulações foram satisfatórios e seguem abaixo.

Com configuração aplicada ao T50 verificou-se que CPU foi afetada durante o período da simulação e sua a utilização da CPU ficou superior a 50% (Figura 2), assim como a utilização de Memória, onde no período da simulação a memória livre estava em 79,9 MB passou para 68,8 MB (Figura 3).

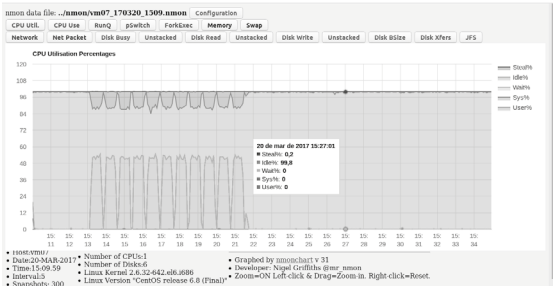


Figura 2. Medição de CPU.

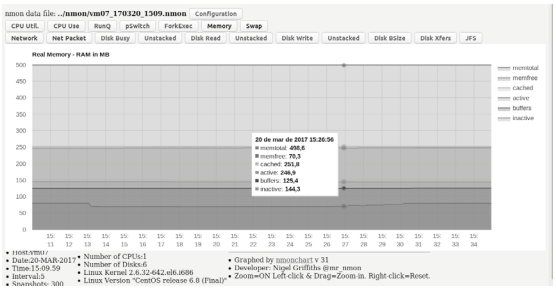


Figura 3. Medição de Memória.

Com a configuração aplicada ao T50 verificou-se que o Tráfego de Rede foi afetado durante o período da simulação e a interface de rede indicou um tráfego 9509,7 KB por segundo (Figura 4), com mais de 100000 pacotes por segundo (Figura 5).

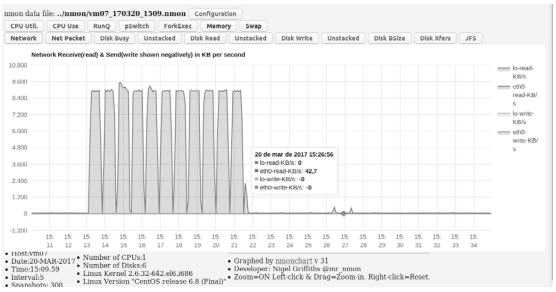


Figura 4. Medição da Tráfego de rede.

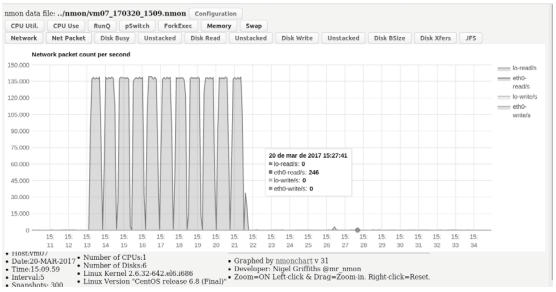


Figura 5. Medição de Pacotes por segundo.

5. Conclusão

Sobre as Licitações analisadas, apenas a Câmara dos Deputados possui uma especificação de Segurança, Disponibilidade e Desempenho que poderia ser menos afetada.

Os resultados das simulações foram satisfatórios e podemos inferir que a Disponibilidade não foi afetada, o Desempenho foi afetado e consequentemente a Segurança.

Um ponto importante é legitimar o tráfego de rede e diferenciar quais requisições são verdadeiras e quais são falsas, esta é uma das maiores dificuldades nos computadores atacados. Com nossa análise de resultados, poderemos desenvolver algoritmos que nos permitam legitimar o trafego de rede [Braga et al. 2010].

Referências

18a. REGIÃO, T. R. D. T. D. (2015). PREGÃO ELETRÔNICO N. 036/2016). Disponível em : `http://www1.trt18.jus.br/licitacao/anexos/2016PE0360002.pdf` , (Acesso em 25 Julho 2017).

 © 2017 Sociedade Brasileira de Computação

Braga, R., Mota, E., and Passito, A. (2010). Lightweight ddos flooding attack detection using nox/openflow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 408–415. IEEE.

Brito, N. (2017). T50. Página Oficial. https://sourceforge.net/projects/t50/, (Acesso em: 04 abr. 2017).

Butler, K., Farley, T. R., McDaniel, P., and Rexford, J. (2010). A survey of bgp security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122.

Cert.br (2016). Recomendações para Melhorar o Cenário de Ataques Distribuídos de Negação de Serviço (DoS). Disponível em : https://www.cert.br/docs/whitepapers/ddos/ , (Acesso em 04 Abril 2017).

de Monte Carmelo, P. M. (2017). EDITAL PREGÃO PRESENCIAL N. 34/2017. Disponível em : http://www.montecarmelo.mg.gov.br/uploads/34-Edital-Pregao-34.2017-SRP-34.2017-Contratacao-de-Internet.pdf , (Acesso em 25 Julho 2017).

dos Deputados Federal, C. (2015). EDITAL DO PREGÃO ELETRÔNICO N. 180/15). Disponível em : http://www.camara.leg.br/internet/contratos/LicitacaoArquivosDownload.asp?numSeqArquivoLicitacao=23137 , (Acesso em 25 Julho 2017).

Federal, S. (2016). PREGÃO ELETRÔNICO N. 90/2016 - Edital (Processo n. 00200.015723/2015-41). Disponível em : https://www.senado.leg.br/transparencia/liccontr/licitacoes/download.asp?COD_LICITACAO=41794 , (Acesso em 25 Julho 2017).

Griffiths, N. (2017a). nmon. Página Oficial. http://nmon.sourceforge.net/, (Acesso em: 04 abr. 2017).

Griffiths, N. (2017b). Nmonchart. Página Oficial. http://nmon.sourceforge.net/pmwiki.php?n=Site.Nmonchart, (Acesso em: 04 abr. 2017).

Leão, M. M., do Amaral, A. C. P., and Echeverria, E. L. (2017). O acordo como instrumento de garantia dos serviços de tecnologia aplicados às redes de computadores. *REVISTA FAIPE*, 4(1):19–32.

Mirkovic, J. and Reiher, P. (2004). A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53.

Rezende, D. (2008). *Planejamento Estratégico para Organizações Privadas e Públicas*. Brasport.

Santos, M. T. and et. al. (2015). *Gerência de Redes de Computadores*. "RNP".

Sêmola, M. (2003). *Gestão Da Segurança Da Informação*. ELSEVIER EDITORA.

Somal, L. and Virk, S. (2014). Classification of distributed denial of service attacks–architecture, taxonomy and tools. *Int. J. Adv. Res. Comput. Sci. Technol*, 2(2):118–122.

von Solms, R. (1998). Information security management (3): the code of practice for information security management (bs 7799). *Information Management & Computer Security*, 6(5):224–225.

 © 2017 Sociedade Brasileira de Computação

WTE – III Workshop de Tecnologia Eleitoral

© 2017 Sociedade Brasileira de Computação

Do *back-end* para o *front-end*: Uma Avaliação de Segurança de Aplicações Móveis da Justiça Eleitoral

Wellington Silva de Souza[1], **Daniel G. de Azevedo**[1], **Samuel Xavier de Souza**[1]

[1]Universidade Federal do Rio Grande do Norte - Natal/RN

wellingtonsouza@imd.ufrn.br, {daniel.galvao,samuel}@dca.ufrn.br

Abstract. *The Internet and mobile technologies for computing and communication represent the nowadays main foundation for the improvement of citizenship, promoting access to justice and social peace. However, some solutions in mobile applications at this scope have vulnerabilities with potential to cause from the unavailability of services to the inclusion, tampering and undue exclusion of information. This work presents a security assessment of mobile applications developed for the 2016 Brazilian Elections, focusing on the analysis of back-end servers faults, exploitation of communication vulnerabilities and mitigation strategies of discovered risks.*

Resumo. *A Internet e as tecnologias móveis de computação e comunicação representam, hoje, o principal alicerce para o aprimoramento da cidadania, promovendo o acesso à justiça e paz social. Verifica-se, contudo, que algumas soluções em aplicações móveis voltadas a este escopo apresentam vulnerabilidades com potencial para ocasionar desde a indisponibilidade dos serviços à inclusão, adulteração e exclusão indevida de informações. Busca-se neste trabalho apresentar uma avaliação de segurança das aplicações móveis desenvolvidas para as Eleições 2016, com enfoque na análise de falhas presentes nos servidores de back-end, exploração de vulnerabilidades de comunicação e estratégias para mitigação dos riscos encontrados.*

1. Introdução

As Eleições 2016 contaram com uma profusão de aplicações móveis destinadas ao apoio das mais diversas atividades relacionadas ao pleito eleitoral. Tais aplicações representam um novo marco na modernização do processo eleitoral, trazendo celeridade, transparência e maior participação do cidadão neste mecanismo fundamental para o exercício da democracia.

No entanto, em quaisquer contextos onde a Tecnologia da Informação é aplicada, a segurança da informação passa a ter fundamental importância, devendo, assim, ser parte intrínseca do processo. Na Justiça Eleitoral Brasileira este requisito se apresenta com ainda mais ênfase, dada a criticidade das atividades de negócio desenvolvidas. Neste ambiente, a segurança da informação representa muito mais que um processo - é um valor institucional reconhecido internacionalmente.

No tocante às aplicações móveis voltadas à Eleições 2016 verifica-se, contudo, a existência de um conjunto de riscos às segurança da informação. As vulnerabilidades encontradas têm potencial para, nos casos mais graves, permitir o acesso não autorizado à rede privativa da Justiça Eleitoral, suas bases de dados e serviços.

 © 2017 Sociedade Brasileira de Computação

Busca-se, assim, avaliar estes riscos e determinar contramedidas eficazes para eliminar ou minimizar a possibilidade de exploração das vulnerabilidades encontradas. Na seção 2 são elencadas as aplicações móveis analisadas neste estudo. A seção 3 apresenta a metodologia de avaliação de segurança e os tipos de testes realizados. Os resultados destes testes são objeto de análise na seção 4, onde também são discutidos os riscos à segurança da informação dos artefatos encontrados. Na seção 5 é mostrada uma prova de conceito, com a exploração de algumas das vulnerabilidades apresentadas. A seção 6 registra os procedimentos adotados para comunicação dos resultados aos responsáveis pelas aplicações e, por fim, a seção 7 consolida o trabalho, indicando futuros desdobramentos.

2. Aplicações móveis avaliadas

Segundo divulgado pelo Tribunal Superior Eleitoral [TSE 2016], o processo eleitoral de 2016 contou com onze aplicativos para dispositivos móveis voltados ao apoio do processo eleitoral. Destes, nove foram disponibilizados à população e dois restritos ao uso interno da Justiça Eleitoral - JE, conforme elencados a seguir:

- aplicativos públicos: Agenda JE, JE Processos, QR Code, Candidaturas, Agregador, Pardal, Onde votar ou justificar, Apuração, Mesários;
- aplicativos para uso interno da JE: QRUEL, QRCode do RUNIN.

De modo a concentrar o foco da pesquisa e tomando como critério de seleção a perspectiva de longevidade de uso regular por parte dos usuários, foram escolhidas três aplicações públicas para o processo de avaliação de segurança:

- Agenda JE: contém as datas do processo eleitoral a serem respeitadas por partidos políticos, candidatos, eleitores e pela própria Justiça Eleitoral.
- JE Processos: permite consultar e acompanhar processos físicos e eletrônicos na Justiça Eleitoral.
- Pardal: permite o envio de denúncias com indícios de práticas indevidas ou ilegais no âmbito da Justiça Eleitoral. Tais denúncias devem conter informações e evidências que ajudem a Justiça Eleitoral no combate a crimes eleitorais.

3. Metodologia de avaliação

A realização de testes de segurança visa identificar falhas de software que podem potencialmente levar a violações de segurança, bem como avaliar contramedidas que podem ser aplicadas para mitigar tais riscos [Pan 1999].

No presente trabalho, são realizados testes de segurança de penetração, por meio da metodologia *black-box* [Beizer 1995] [Nouman et al. 2016]. Neste tipo de avaliação, somente as entradas e saídas são visíveis, e a funcionalidade é determinada por meio da observação das saídas para as entradas correspondentes, sem conhecimento e acesso a estruturas internas do software (código fonte, APIs, fluxo de dados, entre outras).

A abordagem adotada caracteriza-se como composição entre testes de penetração manuais exploratórios, testes de penetração manuais sistemáticos e testes de penetração automatizados [Austin et al. 2013]. Estas técnicas, tradicionalmente utilizadas em ambientes convencionais de computação, foram adaptadas para o contexto de aplicações móveis. Os dados dos experimentos foram coletados durante o mês de setembro/2016, três semanas antes do primeiro turno das Eleições 2016, conforme detalhado a seguir.

 © 2017 Sociedade Brasileira de Computação

3.1. Testes de penetração manuais exploratórios

Os testes de penetração manuais exploratórios foram utilizados para prover uma visão geral do padrão de comunicação gerado pelas aplicações em interação com seus servidores de *back-end*.

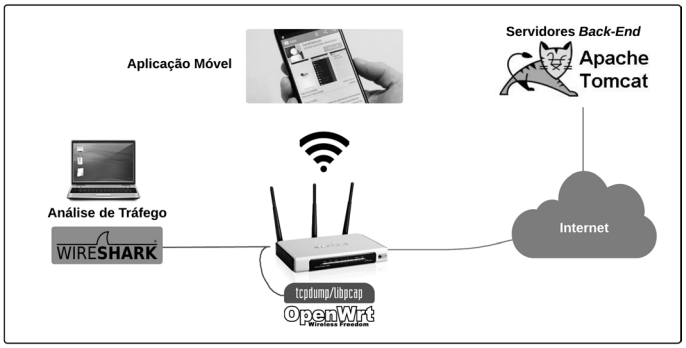
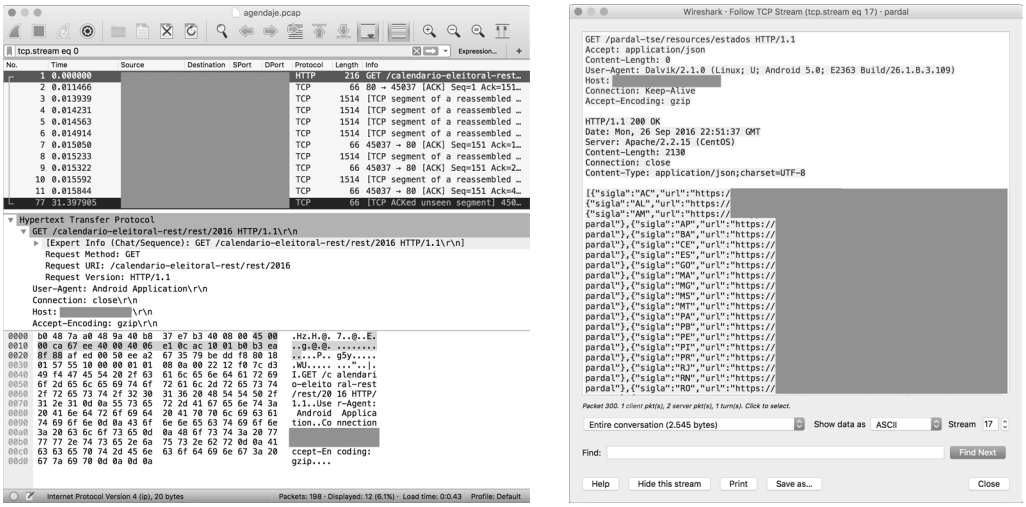


Figura 1. Plataforma de experimentação para captura de tráfego

Neste sentido, foi estabelecido uma plataforma de experimentação para captura do tráfego originado e recebido pelo dispositivo móvel, apresentado na Figura 1. Uma rede Wi-Fi monitorada é provida por um equipamento 802.11n adaptado para executar o OpenWrt [OpenWrt 2016]. O OpenWrt é uma distribuição Linux para dispositivos embarcados, utilizada especialmente em roteadores sem-fio em substituição ao *firmware* original do fabricante.

Utilizando o *sniffer* tcpdump [Tcpdump 2016] (disponível para instalação no OpenWrt), foi realizada a captura e armazenamento do tráfego relacionado à aplicação móvel. Através do Wireshark [Wireshark Foundation 2016], é possível realizar uma análise detalhada dos dados trafegados, conforme mostrado na Figura 2(a).



(a) Tráfego entre aplicação móvel e *back-end* (b) Endereços de servidores

Figura 2. Análise dos dados interceptados

Com base nos resultados encontrados, torna-se possível a aplicação de outras ferramentas e metodologias. Por exemplo, com a obtenção dos endereços dos servidores de

 © 2017 Sociedade Brasileira de Computação

back-end da aplicação Pardal (Figura 2(b)), obtida através da análise do fluxo de dados, é possível executar testes de penetração manuais sistemáticos e automatizados para avaliar a segurança dos servidores responsáveis por esta aplicação.

3.2. Testes de penetração manuais sistemáticos

Com base nas informações obtidas na etapa anterior, podemos sistematizar a metodologia de prospecção de dados. Por exemplo, é possível determinar a versão do *container* de aplicação de todos os servidores de *back-end*, conforme registrado na Tabela 1.

Tabela 1. Versões de Servidores de Aplicação

Versão do Apache Tomcat	Quantidade	Quantidade (*major*)	%
6.0.18	1	13	50
6.0.24	2		
6.0.32	3		
6.0.36	1		
6.0.37	2		
6.0.41	1		
6.0.45	3		
7.0.26	1	7	26,9
7.0.30	1		
7.0.40	1		
7.0.47	2		
7.0.59	1		
7.0.70	1		
8.0.30	1	2	7,7
8.0.32	1		
Não identificada	4	4	15,4
Total	**26**	**26**	**100**

Outro teste realizado foi a verificação da disponibilidade de acesso à ferramenta de administração do servidor de aplicação através da Internet. Para o caso do Apache Tomcat, esta aplicação é o Tomcat Manager. Este serviço é ativado na instalação e utiliza uma URL padrão bem conhecida. O acesso ao Tomcat Manager em um dos servidores de *back-end* analisados neste trabalho pode ser verificado na Figura 3.

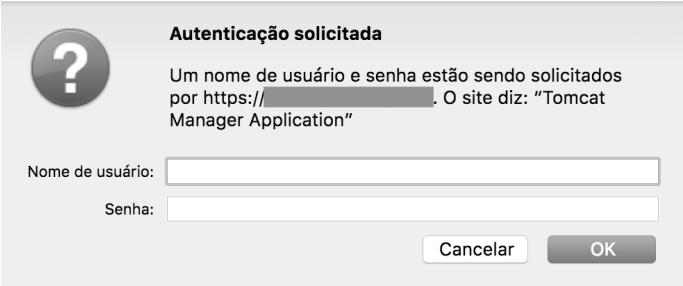


Figura 3. Ferramenta de administração acessível pela Internet

© 2017 Sociedade Brasileira de Computação

Dos 26 servidores verificados, determinamos que 3 (11,5%) estavam com o Tomcat Manager habilitado e exposto à Internet.

3.3. Testes de penetração automatizados

Os testes de penetração automatizados são realizados por meio de ferramentas que executam uma prospecção não-interativa do ambiente alvo. Estas ferramentas avaliam a resposta a diversos tipos de requisição, comparando-as com uma base de dados de vulnerabilidades conhecidas. Ao final do processo, o usuário recebe um relatório com os riscos encontrados.

Para este trabalho foi utilizado o SSL Server Test [Qualys, Inc 2016]. A escolha desta ferramenta decorreu em virtude dos servidores *back-end* estarem executando o protocolo HTTPS, o qual utiliza SSL/TLS para autenticação, integridade e confidencialidade dos dados trafegados. Abordagem semelhante é apresentada em [Cruz and Aranha 2015].

Com base em consulta-resposta, a ferramenta efetua uma triagem das configurações de segurança e sua susceptibilidade a ataques:

- proteção contra ataques bem conhecidos: DROWN, FREAK, POODLE, entre outros;
- utilização de certificados digitais públicos, válidos e confiáveis: avalia se o certificado digital utilizado pelo servidor HTTPS foi emitido por uma autoridade certificadora confiável e se o mesmo se encontra em validade junto ao seu fornecedor;
- assinatura forte do certificado digital;
- comprimento adequado de chaves criptográficas;
- negação de acesso por meio de suítes criptográficas depreciadas (não recomendadas);
- suporte a "segurança futura" (*forward secrecy*);
- suporte a HTTP *Strict Transport Security* (HSTS) [Hodges et al. 2012].

Com base na aderência e conformidade às boas práticas de configuração para servidores SSL, a ferramenta pontua o serviço em cada um dos itens avaliados, apresentando como resultado o grau de segurança do alvo em uma escala que varia de A+ (muito seguro / alta conformidade) a F (muito inseguro / baixa conformidade).

A Figura 4 apresenta o resultado do SSL Server Test aplicado a um dos servidores de *back-end*.

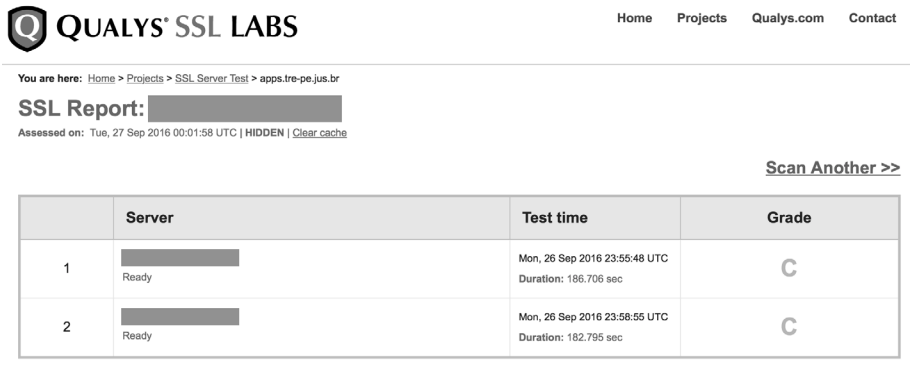


Figura 4. Resultado de uma varredura automatizada com o SSL Server Test

 © 2017 Sociedade Brasileira de Computação

4. Análise de resultados, riscos apresentados e estratégias para mitigação de vulnerabilidades

A partir nos resultados apresentados pelos testes de penetração, é realizada nesta seção uma avaliação geral de segurança do ambiente das aplicações móveis voltadas às Eleições 2016, elencando, especialmente, os riscos à segurança da informação encontrados na pesquisa.

4.1. Endereços de *back-end* em texto claro e sem autenticação

A partir dos testes manuais exploratórios, verificou-se que as aplicações móveis estabelecem comunicação com servidores de *back-end* hospedados na rede da Justiça Eleitoral. Estes são localizados por meio de consulta a *webservices*, os quais respondem com mensagens JSON contendo a lista de URLs de acesso. Foram encontrados 26 servidores, correspondentes a cada uma das unidades da federação. Percebe-se, conforme mostrado nas Figuras 2(b) e 5, que este tráfego ocorre em texto claro/plano, permitindo uma fácil localização de todos os *endpoints*.

Figura 5. Tráfego em texto claro recebido pela aplicação móvel Agenda JE

Para este caso o ideal seria a utilização de HTTPS com certificados públicos, válidos e confiáveis , de modo que as mensagens JSON fossem cifradas, dificultando a obtenção das URLs de acesso aos servidores. A utilização de HTTPS permitiria, também, a confirmação da origem da mensagem JSON, a qual deveria receber assinatura digital para confirmar sua autenticidade e integridade. Da forma como se apresenta atualmente, é possível executar um ataque de *spoofing*, forjando uma mensagem contendo dados falsos ou endereços ilegítimos de servidores *back-end*, causando apresentação incorreta de informações ao usuário, negação de serviço ou captura de dados confidenciais. Além disso, de posse da lista de servidores, é possível disparar ataques de negação de serviço à infraestrutura de suporte às aplicações móveis.

 © 2017 Sociedade Brasileira de Computação

4.2. Divulgação do nome e versão do *container* de aplicação

Os testes manuais sistemáticos permitiram a obtenção de informações importantes a respeito da infraestrutura dos servidores de *back-end*. Foi possível obter a identificação do *container* de aplicações (Apache Tomcat) bem como sua versão em 84,6% dos casos. De posse destas informações, basta a simples consulta a bases públicas de vulnerabilidades e determina-se a quais aquele serviço está exposto. Por exemplo, o Apache Tomcat 6.x.x possui uma lista de 29 falhas de segurança classificadas como importantes, 8 moderadas e 36 de baixo risco/impacto [The Apache Software Foundation 2016].

Ainda com relação à versão da plataforma de aplicações, convém observar que 50% dos servidores executam o Apache Tomcat 6.x.x, o qual terá suporte descontinuado a partir de 2017. A recomendação, para este caso, seria a migração para versões mais recentes e a inibição da apresentação do nome e versão do *container*.

4.3. Ferramenta de administração acessível através da Internet

Verificou-se ainda que três servidores, dos 26 analisados, possuíam a ferramenta de gerenciamento Tomcat Manager disponível para acesso via Internet. Existem ferramentas e métodos de ataque voltados especificamente a explorar vulnerabilidades do Tomcat Manager [Rapid7 2016] [PenTestLab 2012].

Um ataque bem sucedido a esta vulnerabilidade permitiria ao atacante implantar alguma ferramenta de acesso remoto ao servidor (um serviço de terminal, por exemplo) e, partir deste, disparar ataques a serviços internos da rede da Justiça Eleitoral, em especial aqueles voltados diretamente à recepção/contabilização de votos e divulgação de resultados. Tais ataques teriam potencial de dano ampliado se realizados no interim do pleito eleitoral.

Esta configuração expõe o ambiente a um risco desnecessário, uma vez que é possível configurar o serviço para acesso exclusivo a partir de endereços pré-configurados (*white-list*), ou mesmo desativá-lo e efetuar o gerenciamento por meio de acesso remoto via console ao servidor.

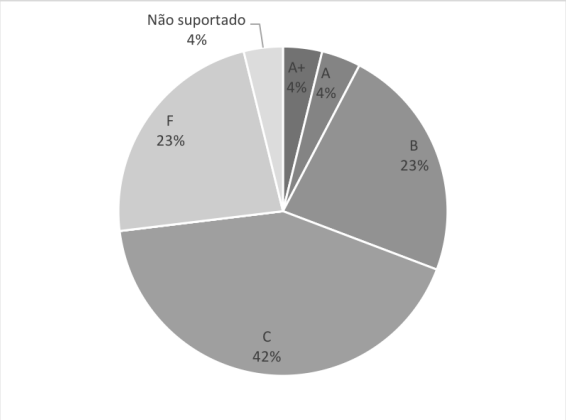


Figura 6. Classificação de segurança dos servidores de *back-end*

4.4. Vulnerabilidades SSL

Por meio dos testes de penetração automatizados (sumarizados na Figura 6) determinamos que 65% dos *back-ends* apresentam níveis inadequados de segurança (inferior a B,

 © 2017 Sociedade Brasileira de Computação

na avaliação do SSL Server Test), refletindo baixa conformidade a boas práticas recomendadas para servidores HTTPS.

Por outro lado, seis servidores atingiram a classificação B (23%) e um atingiu a classificação A (4%). Apenas uma instância atingiu a nota máxima da ferramenta, A+, representando 4% do total de endereços testados. Um dos servidores (4%) apresentou configuração não suportada pela ferramenta de avaliação (serviço HTTPS sendo executado em porta diferente da padrão).

4.4.1. Susceptibilidade a ataques SSL

Conforme apresentado na Tabela 2, 69,1% dos *back-ends* apresentaram vulnerabilidade a ataques SSL conhecidos. Para o POODLE, 16 servidores se mostraram passíveis de ataque. DROWN e FREAK registraram um servidor vulnerável, cada.

Tabela 2. Vulnerabilidades encontradas pela ferramenta SSL Server Test

Vulnerabilidade SSL	Servidores Vulneráveis	%
Assinatura de certificado fraca	4	15,4
Ataque DROWN	1	3,8
Ataque FREAK	1	3,8
Ataque POODLE	16	61,5
Certificado digital não-confiável	13	50
Chaves DH fracas	7	26,9
Criptografia RC4 disponível	17	65,4
Secure Socket Layer v2 ativo	5	19,2
Sem suporte a ”Segurança Futura”	20	76,9

Estes ataques estão associados à utilização de suítes criptográficas com falhas de segurança conhecidas, como o RC4, por exemplo, encontrado em 17 unidades pesquisadas e o SSLv2, operacional em 5 instâncias. São consideradas depreciadas e seu uso não recomendado. A atualização do *container* de aplicação e configuração adequada são suficientes para mitigar este risco. Outra possibilidade é a utilização de um *proxy*-reverso HTTP com configuração SSL adequada, colocado à frente do servidor da aplicação para receber requisições dos usuários.

4.4.2. Chaves Diffie-Hellman fracas

Chaves Diffie-Hellman são utilizadas no início da sessão SSL/TLS para a negociação de uma chave de sessão a ser utilizada na conexão HTTPS entre o servidor e cliente. Chaves com tamanho menor ou igual a 1024 bits são susceptíveis a decifragem em tempo factível (menor que dois meses) com a tecnologia computacional atual [Fried et al. 2016].

Esta situação foi encontrada em 26,9% do espaço pesquisado. O procedimento adequado é a substituição das chaves atualmente em uso por outras de maior tamanho (2048 ou 4096 bits).

 © 2017 Sociedade Brasileira de Computação

4.4.3. Certificado digital não-confiável

Metade dos servidores de *back-end* estava configurada com certificados digitais não-confiáveis. Normalmente esta situação ocorre quando são utilizados certificados auto-assinados, que não foram submetidos à validação e credenciamento em uma autoridade certificadora confiável (ICP-Brasil [ITI 2017], por exemplo).

Dessa forma, conclui-se que, no desenvolvimento das aplicações móveis que se utilizam destes *back-ends*, foi suprimida ou ignorada a verificação da cadeia de certificação do servidor HTTPS. Esta situação apresenta um risco em potencial para a aplicação móvel e seu usuário. Uma vez que não é realizada a validação do certificado, a aplicação se conectará com qualquer servidor que possua configuração HTTPS ativa, ainda que com um certificado forjado.

Esta situação é ainda mais crítica em aplicações que gravam dados no servidor remoto, como é o exemplo do Pardal. Utilizando um certificado falso, um atacante pode interceptar a comunicação da aplicação com o servidor de *back-end*, tendo condições de descobrir senhas e endereços REST que aceitam gravação de conteúdo no servidor remoto. De posse destes dados, o atacante pode comprometer a estrutura de dados da aplicação, apagando dados existentes ou inundando o servidor com solicitações de gravação, exaurindo o espaço de armazenamento.

Para estes casos, é primordial a utilização de certificados digitais válidos e confiáveis, e sua respectiva validação na aplicação móvel.

4.4.4. Falta de suporte a HSTS

Apenas dois servidores foram encontrados com o suporte a HTTP *Strict Transport Security* ativo. O HSTS garante que não haja regressão do mecanismo de acesso (HTTPS para HTTP) em conexões web. Este recurso é ativado por meio de simples configuração. É importante pois impede a utilização de *proxies* maliciosos e ataques *man-in-the-middle* que removam a criptografia da conexão.

5. Exploração de vulnerabilidades

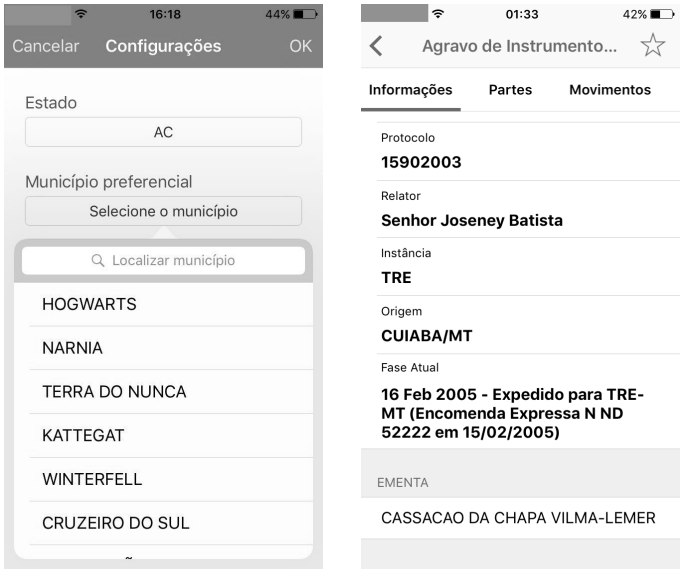
Visando validar as hipóteses levantadas, foi implementada uma prova de conceito, simulando a exploração de algumas vulnerabilidades apresentadas na seção 4. As aplicações JEProcessos e Pardal foram submetidas a uma associação dos ataques DNS *spoofing* e *man-in-the-middle* (MITM).

Segundo [Li et al. 2017] e [Peres and Weber 2003], o tipo de ataque MITM acontece de forma que o atacante comunica-se com o servidor identificando-se como o *host* cliente, ao mesmo tempo que comunica-se com o *host* cliente apresentando-se como o servidor legítimo. O atacante passa, assim, a intermediar a comunicação entre servidor e cliente.

Considerando este cenário, foi criado uma segunda plataforma de experimentação: uma rede Wi-Fi disponibilizada a partir de uma estação de trabalho com sistema operacional macOS 10.12 (Sierra). Um dispositivo móvel, executando as aplicações escolhidas, foi conectado a esta rede.

 © 2017 Sociedade Brasileira de Computação

Para a aplicação Pardal foi utilizado o *software* Burp Suite [PortSwigger Ltd. 2017], de modo a simular um *web proxy*. As requisições HTTP do cliente foram redirecionadas pelo *proxy* a um servidor Apache, em execução na mesma estação. A aplicação móvel recebeu inadvertidamente dados falsos, conforme mostrado na Figura 7(a).



(a) Pardal (b) JEProcessos

Figura 7. Exploração de vulnerabilidades com ataques MITM e DNS *spoofing*

Os testes com o JEProcessos foram realizados através de código em linguagem Python, simulando um servidor HTTP. Conjuntamente, um servidor DNS foi configurado na estação de trabalho, por meio do *software* Dnsmasq [Kelley 2017]. Recebendo respostas DNS falsificadas (DNS *spoofing*), o dispositivo móvel passou a se conectar ao servidor HTTP simulado, recebendo e apresentando dados adulterados (Figura 7(b)).

Em ambas as abordagens, as aplicações não apresentaram alterações de funcionamento. Em um cenário real, o ataque teria sido executado com sucesso, despercebido ao usuário final.

6. Comunicação dos resultados ao responsável

Uma versão preliminar do presente documento foi enviada ao Tribunal Superior Eleitoral na segunda semana de dezembro/2016, previamente a qualquer divulgação do trabalho.

Em resposta, enviada na terceira semana de janeiro/2017, o referido órgão se mostrou receptivo, agradecendo a colaboração e concordando com as assertivas elencadas. Solicitou, ainda, dados complementares para realização das intervenções propostas, e reportou que estava adotando das medidas necessárias sanar os problemas encontrados.

Foi verificado que os artefatos de segurança apresentados em 4.3 e 4.4.3 foram mitigados, por meio da indisponibilização da ferramenta de gerenciamento do *container* e substituição dos certificados digitais de todos os servidores de *back-end*, respectivamente. Testes adicionais são necessários para confirmar a efetiva supressão das demais vulnerabilidades.

 © 2017 Sociedade Brasileira de Computação

7. Conclusão e trabalhos futuros

Neste trabalho foi realizada uma análise de segurança de aplicações móveis utilizadas nas eleições brasileiras de 2016, bem como de seus servidores de *back-end*, utilizando a abordagem *black-box* em conjunto com testes de penetração. Esta combinação, tradicionalmente adotada na avaliação de sistemas convencionais, se mostrou eficaz na determinação dos riscos à segurança da informação para aplicações móveis de modo geral. Juntamente com as plataformas de experimentação, representam contribuições da presente pesquisa.

Por meio de testes de penetração, averiguou-se que 65% dos endereços apresentaram níveis inadequados de conformidade a boas práticas de segurança para servidores HTTPS. Como exemplos, temos a utilização de suítes criptográficas não-recomendadas, certificados digitais não-confiáveis e chaves criptográficas de tamanho inadequado. Foi observado, ainda, que 69,1% dos serviços são passíveis de ataques conhecidos.

Além disso, informações e serviços críticos relativos à infraestrutura das aplicações estavam acessíveis por meio da Internet, aumentando o número de vetores de ataque em potencial. Dados críticos para o funcionamento das aplicações móveis como, por exemplo, endereços de acesso aos servidores de *back-end*, são trafegados sem criptografia e desprovidos de mecanismos de assinatura digital, não possibilitando uma validação da autenticidade das informações recebidas pela aplicação.

Estratégias de mitigação de riscos para as vulnerabilidades de *back-end* e *front-end* foram apresentadas. Adicionalmente, foi executada uma prova de conceito, explorando duas aplicações móveis e validando as hipóteses apresentadas no trabalho. Com a divulgação dos resultados junto aos responsáveis pelos serviços avaliados pretende-se, como trabalho futuro, repetir a avaliação, incluindo a análise de uma quantidade maior de vetores de ataque (negação de serviço, exploração de vulnerabilidades SSL, entre outras), de modo a confirmar a tomada de medidas adequadas à proteção dos usuários e sistemas.

Referências

Austin, A., Holmgreen, C., and Williams, L. (2013). A comparison of the efficiency and effectiveness of vulnerability discovery techniques. *Information and Software Technology*, 55(7):1279–1288.

Beizer, B. (1995). *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc.

Cruz, R. and Aranha, D. (2015). Análise de segurança em aplicativos bancários na plataforma Android. In *XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (Florianópolis/SC)*.

Fried, J., Gaudry, P., Heninger, N., and Thomé, E. (2016). A kilobit hidden snfs discrete logarithm computation. Cryptology ePrint Archive, Report 2016/961. `http://eprint.iacr.org/2016/961`.

Hodges, J., Jackson, C., and Barth, A. (2012). HTTP Strict Transport Security (HSTS). RFC 6797, RFC Editor. `http://www.rfc-editor.org/rfc/rfc6797.txt`.

ITI (2017). ICP-Brasil - Instituto Nacional de Tecnologia da Informação. Disponível em: `http://www.iti.gov.br/icp-brasil`. Acessado em 01/09/2017.

 © 2017 Sociedade Brasileira de Computação

Kelley, S. (2017). Dnsmasq. Disponível em: http://www.thekelleys.org.uk/dnsmasq/doc.html. Acessado em 13/06/2017.

Li, S., Li, X., Hao, J., An, B., Feng, Z., Chen, K., and Zhang, C. (Forthcoming 2017). Defending against man-in-the-middle attack in repeated games. In *26th International Joint Conference on Artificial Intelligence (Melbourne)*.

Nouman, M., Pervez, U., Hasan, O., and Saghar, K. (2016). A survey and tutorial on white and black-box testing of c/c plus plus programs. In *2016 IEEE Region 10 Symposium (Tensymp)*, pages 111–115.

OpenWrt (2016). OpenWrt – Wireless Freedom. Disponível em: https://openwrt.org. Acessado em 21/09/2016.

Pan, J. (1999). Software testing – dependable embedded systems. Disponível em: http://www.sci.brooklyn.cuny.edu/sklar/teaching/s08/cis20.2/papers/software-testing.pdf. Acessado em 21/09/2016.

PenTestLab (2012). Apache Tomcat Exploitation — Penetration Testing Lab. Disponível em: https://pentestlab.wordpress.com/2012/03/22/apache-tomcat-exploitation. Acessado em 21/09/2016.

Peres, A. and Weber, R. F. (2003). Considerações sobre segurança em redes sem fio. In *III workshop em segurança de sistemas computacionais*.

PortSwigger Ltd. (2017). Burp Suite Editions. Disponível em: https://portswigger.net/burp/. Acessado em 23/05/2017.

Qualys, Inc (2016). SSL Server Test (Powered by Qualys SSL Labs). Disponível em: https://www.ssllabs.com/ssltest. Acessado em 21/09/2016.

Rapid7 (2016). Penetration Testing Software, Pen Testing Security — Metasploit. Disponível em: http://metasploit.com. Acessado em 21/09/2016.

Tcpdump (2016). TCPDUMP & Libpcap. Disponível em: http://www.tcpdump.org. Acessado em 21/09/2016.

The Apache Software Foundation (2016). Apache Tomcat - Reporting Security Problems. Disponível em: https://tomcat.apache.org/security.html. Acessado em 21/09/2016.

TSE (2016). Eleições Municipais 2016: Justiça Eleitoral disponibilizará 11 aplicativos para dispositivos móveis. Disponível em: http://www.tse.jus.br/imprensa/noticias-tse/2016/Junho/eleicoes-municipais-2016-justica-eleitoral-lancara-11-aplicativos-para-dispositivos-moveis. Acessado em 21/09/2016.

Wireshark Foundation (2016). Wireshark – Go Deep. Disponível em: https://www.wireshark.org. Acessado em 21/09/2016.

 © 2017 Sociedade Brasileira de Computação

Registro impresso do voto, autenticado e com garantia de anonimato

Rodrigo Carneiro Munhoz Coimbra
José Roberto Menezes Monteiro
Gladiston da Silva Costa

[1]Tribunal Superior Eleitoral (TSE)
Brasília, DF

{rodrigo.coimbra,jose.monteiro,gladiston.costa}@tse.jus.br

Abstract. *The electoral reform approved in the year 2015 brought again the need to produce a material paper record of the vote in Brazilian voting machines. The mere printing of voters' choices, without the addition of any security mechanism, while meeting the requirements of current legislation, opens the door to a series of fraud attempts at recount. This paper presents a printed voting format that allows for an easy conference of voter choices, including mechanisms for automating the counting and checking the authenticity of the vote, while preserving its anonymity. It is not proposed here to debate the effectiveness of the printing of votes by the voting machine, but rather to present a technological solution to a problem that needs to be addressed by the Brazilian Electoral Court.*

Resumo. *A reforma eleitoral aprovada no ano de 2015 trouxe novamente a necessidade de produção de um registro material em papel do voto realizado nas urnas eletrônicas brasileiras. A simples impressão das escolhas do eleitor, sem a adição de qualquer mecanismo de segurança, embora atenda aos requisitos da legislação vigente, abre espaço para uma série de tentativas de fraude no momento da recontagem. Este trabalho apresenta um formato para voto impresso, que permita uma conferência fácil de suas escolhas pelo eleitor, incluindo mecanismos para automação da contagem e verificação da autenticidade do voto, ao mesmo tempo em que seu anonimato é preservado. Não se propõe aqui fazer um debate sobre a efetividade da impressão de votos pela urna eletrônica, mas sim apresentar uma solução tecnológica para um problema que precisa ser tratado pela Justiça Eleitoral brasileira.*

1. Introdução

A Lei nº 13.165/2015 [Presidência da República 2015] institui o registro impresso do voto para conferência pelo eleitor. A lei determina:

> Art. 59-A. No processo de votação eletrônica, a urna imprimirá o registro de cada voto, que será depositado, de forma automática e sem contato manual do eleitor, em local previamente lacrado.

 © 2017 Sociedade Brasileira de Computação

> Parágrafo único. O processo de votação não será concluído até que o eleitor confirme a correspondência entre o teor de seu voto e o registro impresso e exibido pela urna eletrônica.

O novo normativo determina, portanto, que a urna deverá imprimir as escolhas do eleitor e ele não terá qualquer contato manual com o papel, que será depositado em receptáculo fisicamente seguro (lacrado). A lei demanda também que o voto só será computado pela urna após a validação do registro impresso pelo eleitor. Nesse sentido, qualquer formato adotado para o voto impresso deve priorizar a facilidade de verificação pelo eleitor.

Não há na lei qualquer sugestão de mecanismos de segurança associados ao registro impresso, tampouco há a determinação de como deve ser realizada a conferência dos votos em caso de contestação do resultado eletrônico. Esse vácuo normativo demanda que seja dada uma atenção especial a elementos da única etapa do processo que demandará intervenção manual. A recontagem de votos precisa de uma garantia de autenticidade do voto impresso, de modo a evitar fraudes associadas à troca de registros em papel ou à inserção de registros fraudulentos. A contagem também deve ser facilitada e passível de automação e validação.

A seguir é apresentada uma dinâmica de impressão do voto, associada a um formato do registro impresso, que promova uma conferência simples e ágil dos votos pelo eleitor. Em seguida é introduzido um mecanismo de assinatura digital do voto impresso, que acrescenta um elemento autenticador, ao mesmo tempo que preserva o anonimato do voto. Essa assinatura digital está associada a um mecanismo que permitirá o desenvolvimento de aplicativos independentes de apuração segura dos votos impressos. Essas propostas devem ser de fácil implementação pelo Tribunal Superior Eleitoral — TSE, que é o responsável pelo desenvolvimento do software da urna eletrônica.

2. Dinâmica de impressão do voto

O sistema brasileiro de votação consiste na apresentação individual e sequencial dos cargos que estão em disputa na eleição, de modo que o eleitor digita o número do seu candidato para o respectivo cargo e, após a confirmação do seu voto, segue para o próximo cargo até que se encerrem as escolhas possíveis.

Nas Eleições 2002, quando houve a primeira experiência brasileira de impressão de votos conferidos pelo eleitor, essa sistemática foi preservada. Contudo, após a confirmação do voto no último cargo, era produzido o registro impresso dos votos e solicitado ao eleitor que o confirmasse. Nesse momento não havia nada na tela da urna eletrônica que auxiliasse o eleitor a verificar o registro impresso, nada com que ele pudesse confrontar o registro em papel. Esse cenário fatalmente confundiu muitos eleitores, que devem ter rejeitado indevidamente o registro impresso do voto ou sequer o validaram.

Para facilitar a conferência do voto impresso pelo eleitor faz-se necessária a adição de uma nova tela ao Software de Votação — a tela resumo (Figura 1). Após a confirmação do voto no último cargo, é apresentada uma

 © 2017 Sociedade Brasileira de Computação

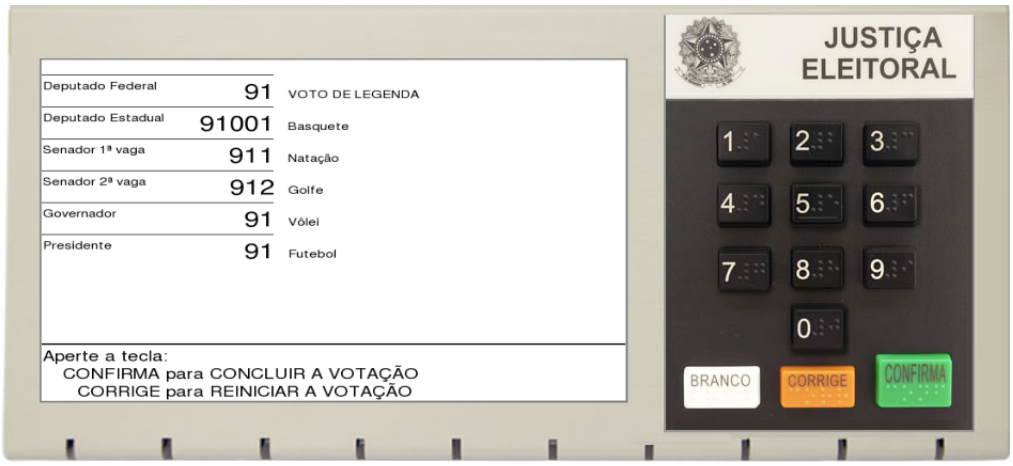


Figura 1. Exemplo de tela resumo para a conferência do voto impresso. As linhas de separação, quando alinhadas ao registro impresso auxiliam na conferência do voto.

tela com o resumo da votação do eleitor, ou seja, são listados na tela os votos em todos os cargos, de forma resumida, ao mesmo tempo em que o registro impresso é apresentado na impressora. Dessa forma, o eleitor pode confrontar o registro impresso com o registro eletrônico, sem a necessidade de recorrer à sua memória.

Nesse ponto, tanto o registro impresso do voto quanto a tela resumo devem apresentar exatamente os mesmos números digitados pelo eleitor. Caso um eleitor anule o seu voto pela digitação de um número de candidato inexistente, esse número deve ser apresentado e não somente a sua interpretação (ou seja, a marcação de voto nulo).

A apresentação da tela resumo, associada com o alinhamento da área de visualização da impressora com a tela da urna, está em conformidade com as recomendações mais modernas de usabilidade de sistemas de votação com registro impresso do voto [EAC 2015].

A partir da tela resumo, deve ser possível ao eleitor confirmar o seu voto ou cancelá-lo. Ao confirmar o seu voto, o registro impresso deve ser depositado automaticamente em receptáculo lacrado, conforme determina a Lei nº 13.165/2015. Caso contrário, o registro impresso também será depositado no mesmo recipiente lacrado, mas com marcação de "cancelado"e a votação reiniciada. Essa nova dinâmica é sintetizada pela Figura 2.

 © 2017 Sociedade Brasileira de Computação

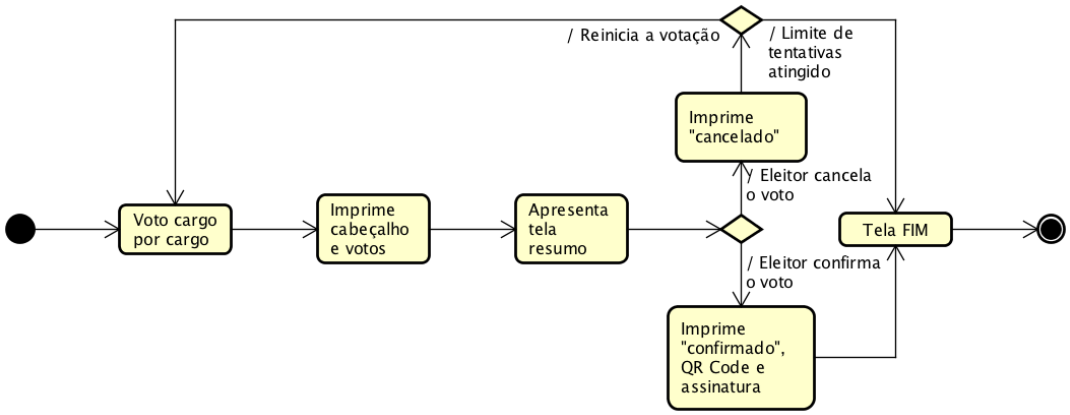


Figura 2. Novos estados do ciclo de votação. Ao cancelar um registro impresso a votação é reiniciada, mas até um determinado limite — nesse caso nenhum registro eletrônico é realizado. Ao confirmar o voto o registro eletrônico é realizado e a votação concluída.

Uma característica desse novo processo de votação é que passa a existir um registro material fidedigno da sequência de escolhas de um eleitor. Esse cenário consiste numa fragilização do anonimato do voto. Isso porque um atacante que está coagindo o eleitor a votar num determinado candidato pode exigir que ele vote numa sequência determinada de votos ou anule seus votos em cargos específicos com números previamente combinados. Numa eventual recontagem dos votos, haveria então um registro impresso que poderia ser verificado pelo atacante como sendo aquele combinado com o eleitor.

Para mitigar essa fragilização do anonimato do voto, seria necessário imprimir um registro individual para cada cargo votado pelo eleitor. Essa solução, porém, prejudica a usabilidade do sistema de votação, uma vez que aumenta significativamente o número de interações entre o eleitor e o equipamento. Outro problema é o aumento do volume de papel demandado para essa solução, uma vez que o formato apresentado a seguir precisaria ser mantido. Além dos custos envolvidos em sua própria aquisição, usar mais papel resulta em diversos custos indiretos associados ao aumento do tamanho da impressora, necessário para comportar uma bobina de papel maior.

Para que a implantação do voto impresso seja bem sucedida, é necessário que o eleitor vote com facilidade e que os custos da solução sejam equilibrados.

 © 2017 Sociedade Brasileira de Computação

3. Formato do registro impresso do voto

O registro impresso do voto, naturalmente, deve conter as escolhas do eleitor para cada cargo em disputa. Adicionalmente, devem ser incluídas informações que permitam rastrear a origem desse documento, com dados sobre a urna eletrônica, a impressora utilizada, onde a urna foi utilizada (ou seja, em qual seção eleitoral ela foi instalada) e dados sobre a eleição.

O voto impresso também precisa conter uma marca bem legível sobre a confirmação da sua validade pelo eleitor. Em seguida, pode-se incluir um QR Code contendo todos os dados que já foram impressos, além de outros que podem ser usados para a assinatura digital, e a própria assinatura. Essa mesma assinatura pode ser impressa após o QR Code. Como será visto adiante, essa assinatura será grande demais, além de ser impressa num momento que se seguirá uma ejeção rápida do papel, o que inviabilizará qualquer tentativa de memorização pelo eleitor.

4. Assinatura do voto impresso

Sob o ponto de vista de segurança, o maior problema é quanto à fraude do voto impresso. O registro impresso deve ter as escolhas do eleitor, um QR Code e um campo de autenticação. Por ser sujeito a um processo de recontagem manual, alguma pessoa pode fraudar o processo fazendo uma inclusão, exclusão, troca ou alteração de algum voto impresso.

Para resolver as disputas, além da autenticação e do QR Code, pode-se usar outros meios adicionais, como o registro digital do voto — RDV gerado pela urna eletrônica e o caderno de votação para a conferência dos eleitores que foram votar na seção.

Especificamente, o procedimento de exclusão vai gerar disputas que só podem ser resolvidas com os meios adicionais (RDV e caderno de votação). Um juiz pode resolver usar o RDV, se o número de assinantes do caderno de votação confere com o de votos do RDV, e é superior ao de votos impressos, então o RDV pode ser usado para obter votos excluídos.

Um procedimento de alteração poderá ser resolvido através do QR Code e do campo de autenticação. Mas também pode-se usar os meios adicionais.

Uma troca consiste em uma exclusão e uma inclusão. Assim, para ser efetiva, é preciso fraudar uma inclusão, e isso será mostrado a seguir. Uma inclusão será evitada através do campo de autenticação. Para inserir um voto fraudado, é preciso criar um campo de autenticação correto.

O campo de autenticação será basicamente uma assinatura feita sobre as escolhas do eleitor, que ficará registrada no voto impresso e poderá ser lida, de forma automática, com o auxílio do QR Code.

Esse último procedimento de fraude será discutido a seguir, sob o aspecto de geração das assinaturas e sua autenticação.

 © 2017 Sociedade Brasileira de Computação

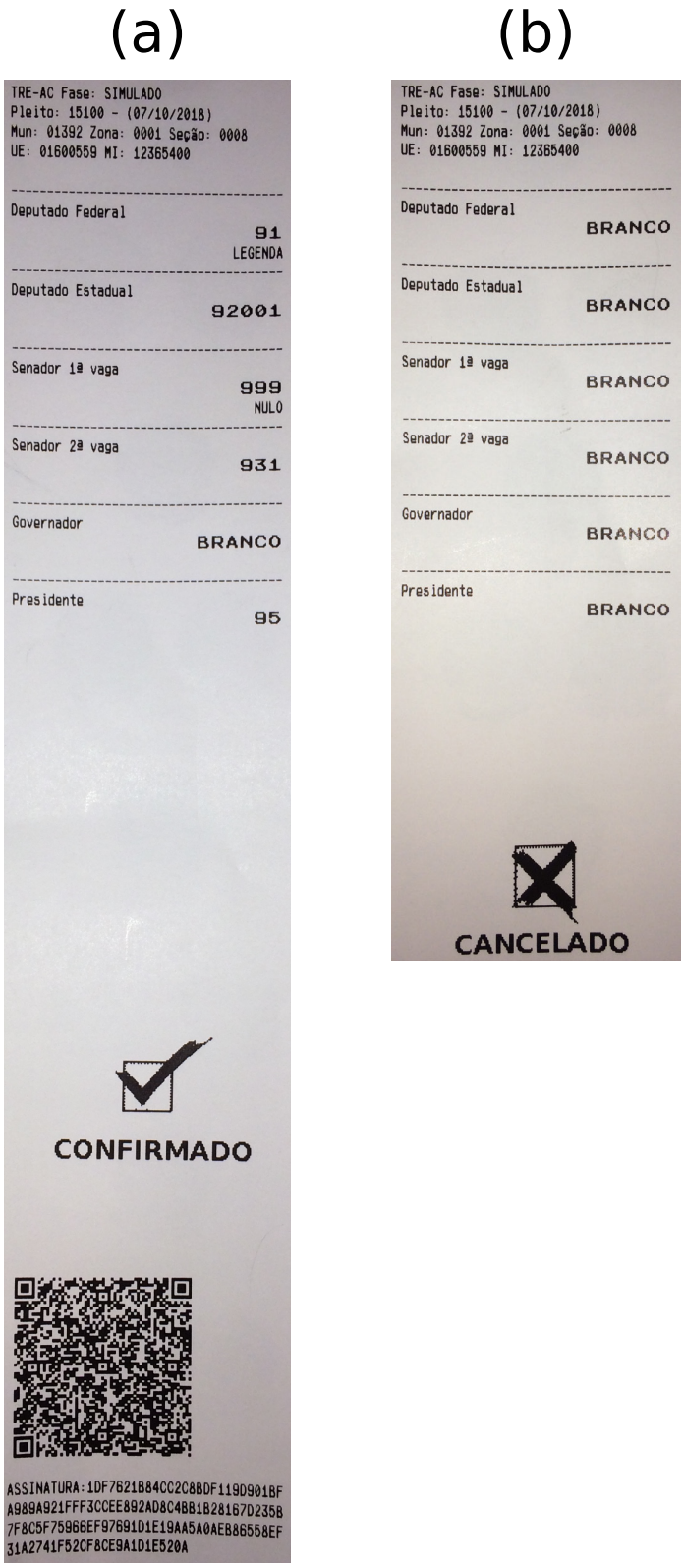


Figura 3. Exemplos de votos impressos. (a) confirmado pelo eleitor. É incluída uma marca gráfica indicando a confirmação do voto, seguida do QR Code e da assinatura digital. (b) cancelado pelo eleitor. Nesse caso o QR Code não precisa ser impresso e é incluída uma marca gráfica bem clara de que o voto foi cancelado. As linhas de separação, quando alinhadas à tela resumo, auxiliam na conferência do voto.

© 2017 Sociedade Brasileira de Computação

4.1. Dados a serem assinados e especificação da assinatura

O conjunto de dados contido no registro impresso do voto que devem ser assinados são:

1. identificadores da urna eletrônica ($IdUE = Nue \| ImpV \| RecV \| Corr$):
   (a) número interno da urna eletrônica (Nue)
   (b) identificação da impressora de votos ($ImpV$)
   (c) identificação do receptáculo de votos ($RecV$)
   (d) código de carga da urna ($Corr$)
2. identificadores da eleição ($IdEl = IdProc \| IdPl \| DtE$):
   (a) processo eleitoral ($IdProc$)
   (b) pleito ($IdPl$)
   (c) data do pleito (DtE)
3. identificadores da seção eleitoral ($IdSe = UF \| Mun \| Zona \| Sec$):
   (a) sigla da UF (UF)
   (b) município (Mun)
   (c) zona ($Zona$)
   (d) seção (Sec)
4. identificador de cada cargo seguido do voto do eleitor ($Votos = (IdP, VotoP) \| (IdG, VotoG) \| (IdS_1, VotoS_1, IdS_2, VotoS_2) \| (IdD_F, VotoD_F) \| (IdD_E, VotoD_E) \| (IdC_n, VotoC_n) \| ...$):
   (a) Presidente ($IdP, VotoP$)
   (b) Governador ($IdG, VotoG$)
   (c) Senadores ($IdS_1, VotoS_1, IdS_2, VotoS_2$)
   (d) Deputado Federal ($IdD_F, VotoD_F$)
   (e) Deputado Estadual/Distrital ($IdD_E, VotoD_E$)
   (f) Cargo N ($IdC_n, VotoC_n$)

Esses campos encontram paralelo naqueles já utilizados pelo TSE no QR Code do boletim de urna [TSE 2016b]. Com isso, não deve haver dificuldade para que a equipe do TSE implemente a proposta apresentada.

Nenhum dado aleatório será incluído na assinatura. A geração de uma assinatura sem aleatórios sobre esses dados implica em dizer que ao se repetir o mesmo voto, na mesma seção e mesma urna, será produzida a mesma assinatura. Essa propriedade é importante, pois decisão do Supremo Tribunal Federal — STF impede que seja associado um número único ao registro impresso do voto [STF 2014].

Assim, pode-se gerar a seguinte assinatura para um voto impresso usando um algoritmo assimétrico para assinatura $E_{pri}()$ com um par de chaves $(priVI, pubVI)$:

$$hash_{VI} = SHA512(IdUE \| IdEl \| IdSe \| Votos) \quad (1)$$

$$Sig_{VI} = E_{priVI}(hash_{VI}) \quad (2)$$

A verificação da assinatura Sig_{VI} pode ser feita por qualquer pessoa que possua a chave $pubVI$, o voto e conheça o algoritmo de verificação $E^{-1}_{pub}()$.

 © 2017 Sociedade Brasileira de Computação

Uma característica desse procedimento de assinatura é que ele não distingue criptograficamente um mesmo voto quando dado por pessoas diferentes. Mais uma vez, essa característica é necessária para atendimento a decisão STF. Dessa forma, é garantido o anonimato do voto: um registro impresso com votos que são únicos já é um registro único e associá-lo a uma assinatura única não altera a sua distinguibilidade; um registro impresso com os mesmos votos de outro, ao conter a mesma assinatura desse outro, não adiciona nenhum elemento que o torne distinguível de outro registro, o que preserva o seu anonimato.

Por outro lado, a indistinguibilidade de votos iguais dá espaço para ataques de inclusão de votos válidos replicados. Esse cenário pode ser minimizado com o reforço de proteção física sobre o receptáculo dos votos impressos e maior controle no processo de contagem dos votos. Outra possibilidade é criar uma associação direta entre RDV e registro impresso, fazendo com que os votos no RDV sejam agrupados da mesma forma que no registro impresso, incluindo a sua assinatura. Dessa forma, disputas ocasionadas por inclusões inválidas por replicação poderiam ser resolvidas considerando como válidos os votos contidos no RDV. Contudo, alterações no formato do RDV não serão tratadas neste trabalho.

Para se usar um padrão de assinatura que possa ser conferido por qualquer pessoa, de forma transparente, propõe-se o uso da assinatura EdDSA [Bernstein et al. 2012], baseada na *Curve25519*, que já tem sido utilizada pelo TSE [TSE 2016b, TSE 2016a].

A assinatura EdDSA não é da forma (r, s), na qual r é um "aleatório"e s a assinatura. Com EdDSA é satisfeito o requisito de que na mesma seção e mesma urna, se produza a mesma assinatura para os mesmos dados, pois o valor r é gerado de forma determinística por meio de um hash. O valor r é gerado com base na chave privada e nos dados a serem assinados [Bernstein et al. 2012].

Assim, para gerar a assinatura do Voto Impresso (VI) usa-se o algoritmo $EdDSA_{pri}()$ com o par de chaves $(priVI, pubVI)$, e com o aleatório r gerado pelo próprio algoritmo. Então a urna executará as seguintes operações:

$$\begin{aligned} hash_{VI} &= SHA512(IdUE \| IdEl \| IdSe \| Votos) \\ s &= EdDSA_{priVI}(hash_{VI}) \\ Sign_{VI} &= (r, s) \end{aligned} \tag{3}$$

A verificação da assinatura $Sign_{VI}$ pode ser feita por qualquer pessoa que possua a chave $pubVI$, os dados de D e conheça o algoritmo de verificação $EdDSA^{-1}_{pub}()$. O tamanho da assinatura é de 512 bits, que podem ser representados por 128 dígitos hexadecimais. Esse comprimento de assinatura pode ser impresso ao final do registro em papel do voto, pois é impraticável a sua memorização pelo eleitor.

 © 2017 Sociedade Brasileira de Computação

O método é bem rápido e seguro, pelo uso do SHA512 e do EdDSA, e já foi usado anteriormente pelo TSE nos QR Codes dos boletins de urna [TSE 2016b, TSE 2016a].

5. Automação segura da apuração

Embora a Lei nº 13.165/2015 não tenha estipulado o que fazer com os registros impressos após a conclusão da votação, certamente era vontade do legislador que, em alguma escala, os votos impressos fossem contados e sua apuração confrontada com o resultado eletrônico, de modo a auditar as urnas eletrônicas. Sabidamente, processos manuais de contagem são propensos a erro. Por isso, a disponibilização de um mecanismo que propicie a automação da apuração dos votos impressos agregará segurança ao processo de auditoria das eleições.

A inclusão de um QR Code ao final do registro impresso do voto, contendo todos os campos que são assinados digitalmente, além da própria assinatura, permitirá o desenvolvimento de aplicativos que rapidamente digitalizem o voto impresso e verifiquem a sua assinatura. Aproveitando a experiência adquirida pelo TSE com o QR Code no boletim de urna, para o voto impresso pode ser adotado o mesmo formato já empregado no boletim, assim como a infraestrutura para publicação de chaves.

A seguir são apresentados os campos que devem estar codificados no QR Code (Tabela 5), assim como um exemplo de QR Code para o voto impresso (Figura 4).

Tabela 1. Tabela de campos do QR Code do voto impresso.

PROC:nnnnn	Número do processo eleitoral.
DTPL:aaaammdd	Data do pleito.
PLEI:nnnnn	Número do pleito.
TURN:n	Número do turno (1 – primeiro turno; 2 – segundo turno).
FASE:x	Fase dos dados (O – oficial; S – simulado; T – treinamento).
UNFE:xx	Sigla da UF.
MUNI:nnnnn	Número do município.
ZONA:nnnn	Número da zona eleitoral.
SECA:nnnn	Número da seção eleitoral.
IDUE:nnnn...	Número de série da urna.
IMIE:nnnn...	Número de série da impressora de votos.
IRVI:nnnn...	Número de série do receptáculo de votos.
IDCA:nnnn...	Código de identificação da carga (24 dígitos).
cc:vvvvv	Código do cargo (cc) seguido do voto (vvvvv).
HASH:xxxxxx...	$hash_{VI}$ codificado em hexadecimal.
ASSI:xxxxxx...	$Sign_{VI}$ codificada em hexadecimal.

 © 2017 Sociedade Brasileira de Computação

Figura 4. Exemplo de QR Code para o voto impresso.

6. Conclusões

O voto impresso é uma necessidade para as Eleições 2018 imposta por lei. A sua introdução traz desafios de usabilidade e segurança ao processo eleitoral brasileiro. A adição de uma tela resumo para conferência dos registros eletrônico e impresso certamente promoverá uma experiência mais adequada para o eleitor, se comparado às Eleições 2002 quando esse mecanismo não existia. A inclusão de uma assinatura digital baseada em EdDSA agrega um forte mecanismo de autenticação, ao mesmo tempo que preserva o anonimato do voto. Finalmente, a adição de um QR Code ao registro impresso viabilizará a construção de aplicativos que automatizem a apuração dos votos numa eventual auditoria das eleições. As principais propostas apresentadas devem ser implementadas sem grande dificuldade pelo TSE, uma vez que se baseiam em tecnologias já experimentadas no boletim de urna.

Futuramente, após a implantação da proposta apresentada, é necessário avaliar eventuais melhorias de usabilidade e de segurança que podem ser agregadas ao registro impresso do voto.

Referências

Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B.-Y. (2012). High-speed high-security signatures. *Journal of Cryptographic Engineering*, pages 1–13.

EAC (2015). Voluntary voting system guidelines. `https://www.eac.gov/assets/1/28/VVSG.1.1.VOL.1.FINAL1.pdf`. United States Election Assistance Comission.

 © 2017 Sociedade Brasileira de Computação

Presidência da República (2015). Lei nº 13.165 de 2015. hhttp://www.planalto.gov.br/ccivil_03/_ato2015-2018/2015/lei/l13165.htm. Altera as Leis nº 9.504, de 30 de setembro de 1997, nº 9.096, de 19 de setembro de 1995, e nº 4.737, de 15 de julho de 1965 - Código Eleitoral, para reduzir os custos das campanhas eleitorais, simplificar a administração dos Partidos Políticos e incentivar a participação feminina.

STF (2014). ADI 4543 — ação direta de inconstitucionalidade. http://www.stf.jus.br/portal/processo/verProcessoAndamento.asp?numero=4543&classe=ADI&origem=AP&recurso=0&tipoJulgamento=M. Relatora: Min. Cármen Lúcia.

TSE (2016a). QR Code no boletim de urna: instruções adicionais para a verificação de assinatura digital e exemplos de código. http://www.justicaeleitoral.jus.br/arquivos/tse-qr-code-no-boletim-de-urna-verificacao-assinatura-digital.

TSE (2016b). QR Code no boletim de urna: manual para criação de aplicativos de leitura. http://www.justicaeleitoral.jus.br/arquivos/tse-manual-para-a-criacao-de-aplicativos-de-leitura.

 © 2017 Sociedade Brasileira de Computação

Urna eletrônica de terceira geração: Um protótipo para eleições auditáveis

Fernando Teodoro de Lima[1], **Mario A. Gazziro**[1], **Antonio de Abreu Batista Jr** [2]
Paulo Matias[3], **João V. C. Costa**[1]

[1]Universidade Federal do ABC (UFABC)
Santo André – São Paulo – Brasil

[2]Departamento de Informática – Universidade Federal do Maranhão (UFMA)
São Luís – Maranhão – Brasil

[3]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
São Carlos – São Paulo – Brasil

{fernando.lima,mario.gazziro}@ufabc.edu.br, antonio.batista@ufma.br

matias@ufscar.br, custodiovjoao@gmail.com

Abstract. *In this work, we developed a third generation electronic voting machine prototype that meets several proposed functional requirements by the international scientific community, which established – over the last decades – the guidelines for the latest generation of electronic voting machines. In addition to the voting module, we implemented the various modules needed to run an election, such as setup and verification modules. Finally, we report the use of this prototype in a real election occurred in a Brazilian university.*

Resumo. *Nesse trabalho foi desenvolvido um protótipo de urna eletrônica de terceira geração, atendendo a vários dos requisitos funcionais propostos pela comunidade científica internacional, a qual estabeleceu – ao longo das últimas décadas – as diretrizes para a última geração de urnas eletrônicas. Foram implementados, além do módulo de votação, os diversos módulos necessários à execução de uma eleição, como módulos de cadastro e apuração. Por fim, relatamos o uso desse protótipo em uma eleição oficial, ocorrida dentro de uma universidade brasileira.*

1. Introdução

Muitos sistemas de votação eletrônica, entre eles o atual sistema brasileiro, funcionam como uma caixa preta [van de Graaf 2017]. Eles não permitem que o eleitor verifique que o seu voto foi incluído na contagem nem testemunhe uma correta apuração deles, obrigando o eleitor a confiar na boa fé das pessoas. Se tais sistemas algum dia cair sob o controle de pessoas desonestas, elas poderão eleger quem desejarem. Entretanto, o que se espera de um bom sistema eleitoral é que ele permita – dentre outras funcionalidades – que qualquer pessoa possa acompanhar e conferir uma apuração, independentemente de possuir algum conhecimento técnico em informática, obtendo acesso ao conteúdo do voto, e ainda assim, garantindo resguardado o seu sigilo.

Ao longo dos últimos anos diversas máquinas de votação eletrônica foram propostas [Karima et al. 2014, Culnane et al. 2015, Puri et al. 2017]. Todas elas buscando

 © 2017 Sociedade Brasileira de Computação

aprimorar e/ou facilitar os procedimentos de auditoria, tanto do registro do voto como de sua apuração e totalização. A última geração delas, as urnas eletrônicas de terceira geração, são caracterizadas por unir de forma tecnológica os dois tipos de registro do voto: o físico e o digital [Brunazo Filho and Gazziro 2014]. Entre as várias formas de se fazer isso, duas delas se destacam:

1. O uso de códigos de barras *QR* – que nada mais são do que registros digitais impressos em meio físico e que podem ser lidos (escaneados) pela maior parte dos celulares que possuem câmeras fotográficas; e
2. O uso de *tags* de RFID embutidas na cédula de votação. Tal solução foi utilizada nas urnas de terceira geração da Argentina. Nelas o voto era gravado em chips de rádio frequência embutidos nas cédulas de votação e impresso por fora delas. É importante ressaltar que os chips RFID adotados na cédula argentina foram responsáveis por várias das vulnerabilidades descobertas naquele sistema [Oro 2015], sendo a impressão em papel suficiente para mitigar vários ataques, como leitura à distância e gravação com celular.

Entretanto, no Brasil, o uso de urnas de terceira geração ainda está distante de aplicações sérias, sendo que os principais desafios são prover uma solução segura e ganhar a confiança dos eleitores em usá-la [Wang et al. 2017]. Atualmente, o Brasil é o único país do mundo a usar urnas de 1^a geração, em que os votos são gravados apenas digitalmente, não oferecendo possibilidade de auditoria por outros meios [Filho et al. 2015]. Deste modo, a confiabilidade do resultado publicado fica totalmente dependente da confiança no software instalado no equipamento.

Neste artigo, nós apresentamos as soluções técnicas elaboradas para um protótipo de urna eleitoral eletrônica de terceira geração que atende a vários dos requisitos funcionais propostos pela comunidade científica internacional [Adida and Rivest 2006, Ali and Murray 2016]. Apesar da versão atual de nosso protótipo não prover a propriedade de verificação fim-a-fim do voto, eleitores e fiscais podem acompanhar cada parte do processo, que é de fácil compreensão mesmo para um cidadão não-técnico.

2. O protótipo desenvolvido

O protótipo de urna desenvolvido é constituído de diversos módulos, estando o código fonte de todos eles disponível no endereço: `https://github.com/mariogazziro/UrnaEletronica3G`. O projeto foi desenvolvido em linguagem *Python* e faz uso de bibliotecas livres como *PySide* para controle de janelas e *SQL Alchemy* para a base de dados de cadastros. O compilador utilizado foi o GCC da GNU, gratuito e de código aberto, usado e desenvolvido por muitos usuários no mundo inteiro. A seguir, na Tabela 1, apresentamos um comparativo entre o protótipo desenvolvido e a urna eletrônica em uso nas eleições brasileiras, com relação aos requisitos funcionais propostos por [Brunazo Filho and Gazziro 2014].

2.1. Módulos construídos

Os módulos devem ser executados na ordem estabelecida abaixo, seguindo o fluxo natural de uma eleição.

1. Ajuste de Urna: Gera um par de chaves de encriptação para cada seção eleitoral. Uma das chaves do par, a chave pública, é distribuída entre as máquinas de votação

 © 2017 Sociedade Brasileira de Computação

de uma seção eleitoral e usada por elas para encriptar cada voto. A outra chave do par, a chave privada, é distribuída entre as máquinas de verificação de votos e de apuração que a utiliza para verificação de integridade do voto, impedindo que cédulas geradas em outras seções eleitorais sejam apuradas na seção atual.

2. Ajuste de Eleição: Realiza o cadastro dos partidos (ou chapas, no caso de eleições não políticas), dos cargos e dos candidatos. O sistema permite cadastrar mais de um voto possível por cargo, o que permite a um eleitor, por exemplo, votar em 2 candidatos a senador.
3. Votação: O módulo de votação propriamente dito, no qual o eleitor deve necessariamente votar utilizando-se de um número pré-definido para identificação dos candidatos. É permitido ao eleitor, ainda, escolher a ordem de votação desejada, assim, este primeiro escolhe o(s) cargo(s) para o qual deseja votar. Caso o número de candidatos seja pequeno, recomendamos manter uma listagem com os códigos dos candidatos impressa e colada dentro da cabine de votação. No caso de um número muito grande de candidatos, como em eleições nacionais, o eleitor necessita adentrar à seção já sabendo o número de seu candidato, similar às eleições brasileiras.
4. Verificação: Módulo de verificação do voto, a ser usado pelo eleitor para conferir se o conteúdo do código de barras *QR* impresso é igual ao conteúdo escrito por extenso em sua cédula de votação. Idealmente deve ser executado em outra máquina na seção eleitoral, para garantir que não haja comunicação entre a máquina que gerou o voto e a máquina de verificação.
5. Apuração: O módulo de apuração totaliza cada voto apenas uma vez, com o auxílio de um número aleatório de até 10 casas decimais embutido no código *QR*, junto ao conteúdo do voto. O voto é encriptado. Com isso, garante-se que votos de outras eleições ou outras seções eleitorais gerem alertas e não sejam contabilizados. Caso seja constatada divergência entre o registro textual (legível por um humano) e o registro digital (código *QR*) presentes em uma mesma cédula, deve-se considerar o registro digital errôneo e proceder à contagem manual dos votos.

3. Protocolo aplicado ao voto

Com relação ao processo completo adotado em um sistema de eleição utilizando preceitos de terceira geração, listamos nessa seção, em detalhes, todas as fases envolvidas no processo.

Fase de inicialização

A comissão eleitoral executa os procedimentos listados abaixo.

1. Criação de chaves:

   1.1 execute o módulo de Ajuste de Urna. O módulo retornará um par de chaves de encriptação - pública k_{pub} e privada k_{pri}.

   1.2 insira uma cópia da chave pública k_{pub} em cada uma das máquinas de votação e uma cópia da chave privada k_{pri} nas máquinas de verificação e apuração.
2. Cadastros:

   1.1 São cadastrados os Partidos (ou Chapas) pela comissão eleitoral, atribuindo um número decimal de dois dígitos.

 © 2017 Sociedade Brasileira de Computação

Tabela 1. Tabela comparativa, extraída e adaptada de [Brunazo Filho and Gazziro 2014]

Princípio da Publicidade	Urna brasileira	Urna 3G UFABC
Gera voto impresso conferível pelo eleitor	NÃO	SIM
Eleitor pode conferir o conteúdo da gravação digital do voto antes de sair do local de votação	NÃO	SIM
Fiscal externo pode verificar a igualdade entre os diversos registros do voto	NÃO	SIM
Fiscal externo pode acompanhar e verificar a contagem dos votos de cada seção eleitoral	NÃO	SIM
Princípio da Independência do Software		
Uma modificação ou erro não detectado no software pode causar um erro indetectável no resultado da apuração	SIM	NÃO
Outros Conceitos Desejáveis		
Tempo para publicação na Internet dos resultados por Seção, para fiscalização da Totalização	72h (2012)	1,5 hora por seção com 400 votos
Conferência da assinatura digital do software feita em equipamento sob controle do fiscal	NÃO	SIM
Distribuição matricial de urnas e mesas: o eleitor pode escolher uma urna livre para votar, sem ter que esperar que um eleitor anterior complete seu voto. Menores filas.	NÃO	SIM
Eleitor pode escolher a ordem dos cargos a votar	NÃO	SIM
Adaptação para plebiscitos e outras consultas - disponibilidade de opções “sim”, “não”, ou outras mais específicas	NÃO	SIM
Troca de equipamento defeituoso	LENTA	RÁPIDA
Necessidade de recuperação de dados	SIM	NÃO

 © 2017 Sociedade Brasileira de Computação

1.3 Realização do cadastro dos cargos, com a possibilidade de definição do número de votos por cargo, a exemplo do caso do cargo a senador, no qual cada eleitor pode votar em até dois candidatos.

1.3 São cadastrados todos os candidatos e seus respectivos partidos (ou chapas)e cargos, com a possibilidade de inclusão de foto.

Fase de votação

Primeiramente, os eleitores se identificam aos mesários, que verificam se estes estão aptos a votar. E, caso estejam aptos, as suas assinaturas também são colhidas por eles. Uma vez que um eleitor entra na cabine, o procedimento é como segue:

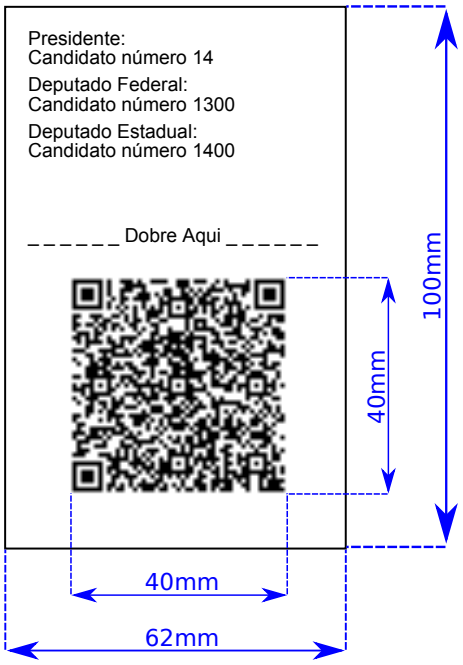


Figura 1. Cédula de votação gerada pela urna de terceira geração, com suas dimensões. Registro físico e digital unidos no mesmo tipo de mídia, o papel.

1. O eleitor entra com a sua escolha de candidatos na máquina de votação.
2. Em resposta às entradas do eleitor, a máquina imprime uma cédula como a da Figura 1. Nela há um código de barras *QR* contendo o registro digital do voto, além de um registro textual legível por um humano.
3. Após receber a cédula, o eleitor dirige-se a máquina de verificação (diferente da que votou) para conferência do voto. Caso detecte alguma divergência com relação à sua escolha de candidatos, o eleitor descarta a cédula e retorna ao primeiro passo.
4. O eleitor deposita a cédula em urna previamente lacrada para garantir o total sigilo do voto.

O registro digital do voto é um código de barras *QR* contendo a concatenação da strings $S_1 = RSA_{K_{pub}}(voto)$ e S_2. S_1 é o voto encriptado e S_2 um número pseudo aleatório gerado pela máquina de votação. RSA é o algoritmo de chave pública RSA [Rivest et al. 1978] e K_{pub} a chave de encriptação copiada para a máquina de votação.

 © 2017 Sociedade Brasileira de Computação

Embaralhamento das cédulas

Cada eleitor vota em um cabine e deposita a sua cédula de votação em uma urna previamente lacrada. A urna deve ser sacudida sempre que uma nova cédula foi incluída nela, após o eleitor ter conferido o seu voto. Este procedimento operacional é importante para garantir ao eleitor o sigilo de seu voto, com relação a ordem de votação dos eleitores em suas seções eleitorais.

Fase de verificação da cédula

O eleitor dirige-se a máquina de verificação para conferência do voto. Ele, então, mostra a sua cédula de votação para ela.

1. A máquina:
   1 lê S_1 a partir do código de barras *QR*;
   2 computa $voto' = RSA_{K_{pri}}(S_1)$; e
   3 imprime $voto'$.
2. O eleitor, então, confere se o conteúdo do código de barras *QR* impresso é igual ao conteúdo escrito por extenso em sua cédula de votação $voto' = voto$; e
3. Caso detecte alguma divergência com relação à sua escolha de candidatos $voto' \neq voto$, o eleitor descarta a cédula e vota novamente.

Fase de transporte das cédulas

A comissão eleitoral lacra as urnas e as leva para área de apuração de votos.

Fase de contagem de votos

O presidente da seção eleitoral deve proceder como segue:

1. Para cada urna lacrada deve-se romper o lacre e retirar as cédulas, uma a uma, enquanto existirem cédulas de votação nela. Cada cédula deve ser processada pelo módulo de apuração.

   A máquina de apuração procede como segue para cada cédula:

1. caso, a *String* aleatória, S_2 lida a partir do código QR, já tenha sido processada antes; uma mensagem de alerta é exibida, indicando que aquela cédula em questão ja foi computada, e nada mais é feito.
2. caso contrário, o conteúdo do voto é decriptado $voto' = RSA_{K_{pri}}(S_1)$ e;
3. o voto é contabilizado para o candidato, e exibido na tela do equipamento, a qual pode ser ligada a um projetor para facilitar a auditoria dos fiscais presentes.

Conforme cada cédula vai sendo lida, o resultado da totalização apresentado na tela vai se atualizando, exibindo sempre a contagem atual de votos para cada candidato. Quando todas as cédulas tiverem sido lidas, o presidente da seção eleitoral aciona o botão *GERAR BOLETIM*. A máquina de votação, então, emite o resultado final da apuração, de forma impressa. Finalmente, o presidente da seção verifica se o número de assinaturas colhidas corresponde ao total de votos contabilizados pela máquina.

 © 2017 Sociedade Brasileira de Computação

4. Probabilidade de colisão de números aleatórios em um seção eleitoral

Cada cédula contém um número aleatório para garantir que, quando escaneado eletronicamente, um voto nunca seja contabilizado mais de uma vez em uma seção.

Porém, caso a máquina acidentalmente escolha para uma cédula um número que já tenha sido sorteado antes e já tenha sido atribuído a um outro eleitor da mesma seção eleitoral, o voto de um deles não seria contabilizado. No entanto, a probabilidade desse evento acontecer é muito pequena. No caso de uma auditoria, poderia-se verificar se a frequência com que essa situação ocorre condiz com a probabilidade esperada teoricamente.

Essa probabilidade é estimada por meio do paradoxo de aniversário. Seja X o número de colisões, k o número de eleitores de uma seção eleitoral e $[1, n]$ o intervalo dentro do qual a máquina sorteia os números aleatórios. Abaixo, calculamos $P\{X = 0\}$, a probabilidade de que não haja nenhuma colisão, e $P\{X \geq 1\}$, a probabilidade de que haja uma ou mais delas.

$$P\{X = 0\} = \prod_{i=0}^{k}(1 - \tfrac{i}{n}) \quad (1)$$

$$P\{X \geq 1\} = 1 - P\{X = 0\} \quad (2)$$

A máquina gera números pseudo aleatórios de até 10 casas decimais. Impor esse comprimento máximo permite que câmeras de baixo custo, em função de suas limitações óticas e baixas resoluções (em geral, em torno de 640×480 pixels) leiam os códigos *QR*. O gráfico da Figura 2 mostra a probabilidade de colisão de números aleatórios em seções eleitorais com diferentes números de eleitores. Nós estamos considerando que a máquina sorteia números em um espaço com $n = 2^{30}$ elementos.

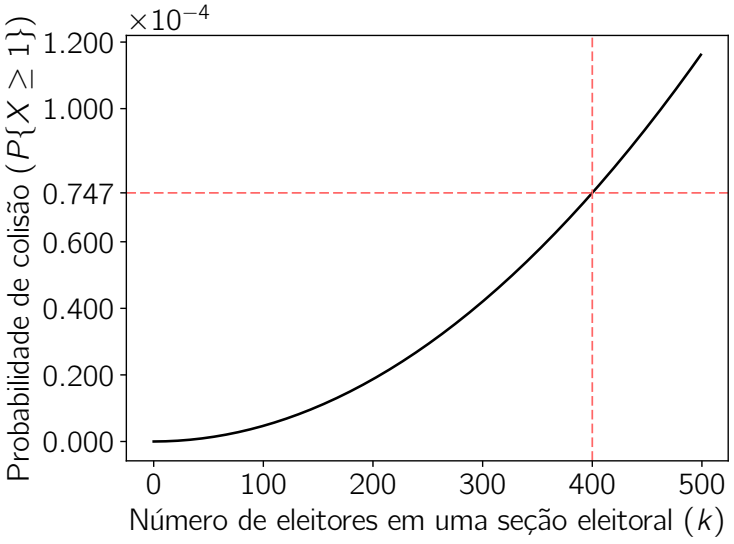


Figura 2. Probabilidade de colisão de números aleatórios em uma seção eleitoral.

© 2017 Sociedade Brasileira de Computação

5. A eleição realizada

O protótipo da urna foi testado em ambiente real, em uma votação referente à portaria n° 25 de 23 de junho de 2016 do CMCC, da instituição de ensino UFABC. A votação ocorreu dia 20 de julho de 2016 no período das 10h00 às 13h00 e das 15h00 às 19h00, nos *campi* de Santo André e São Bernardo da instituição. A eleição disponibilizou 12 vagas no total, 2 para representantes discentes de graduação, 2 para representantes da pós-graduação e 2 para representantes dos servidores técnico-administrativos, bem como os respectivos suplentes em ambas as categorias para composição do Conselho do CMCC, um dos centros da mesma universidade. A apuração da eleição aconteceu dia 21 de julho de 2016 das 13h20 às 13h47 (tal como à ata n° 02/2016 da Comissão Eleitoral para Conselho de Centro do CMCC). Nela, foram constatados 64 eleitores votantes com um total de 63 assinaturas – devido a uma falha dos mesários em colher uma assinatura de um dos eleitores. Tal discrepância foi desconsiderada pelo então presidente da comissão, uma vez que a diferença entre um eleitor e uma assinatura não alteraria o resultado final da eleição. **No dia 22 de julho de 2016 o seu resultado foi divulgado e a eleição foi dada como conclusiva**. Este foi o primeiro uso de uma urna de terceira geração em uma instituição pública brasileira.

5.1. Tempo de apuração

A apuração eletrônica levou 15 minutos, com mais alguns minutos despendidos na elaboração da ata. Ao todo, foram apurados 64 votos, de forma digital, a uma taxa de $\approx$15 segundos por voto, sendo esse o tempo utilizado em retirar a cédula de votação da urna, apresentá-la ao sistema de apuração eletrônico – constituído de um dos módulos de nossa urna eletrônica – e apresentação da tabela de resultados atualizada a uma comissão de fiscais presente, utilizando um *datashow*. Os fiscais também puderam examinar o conteúdo impresso em cada cédula de votação.

Para apurar uma seção eleitoral típica, com 400 eleitores, o tempo estimado é de uma hora e meia.

6. Considerações Finais

Neste trabalho, apresentou-se um protótipo de urna eletrônica de 3^a geração em que se descreveu as suas soluções técnicas. Além disso, mostrou-se também que o protótipo atende a várias das recomendações estabelecidas pela comunidade acadêmica para esta nova geração de urnas eletrônicas.

Dentre as melhorias previstas no desenvolvimento futuro dessa urna eletrônica está incluir em seu projeto o fato de que o voto deve ser incondicionalmente seguro [van de Graaf 2017] além de utilizar encriptação homomórfica com o objetivo de prover mecanismos para verificação fim-a-fim do voto [Rivest 2002].

Referências

Adida, B. and Rivest, R. L. (2006). Scratch & vote: Self-contained paper-based cryptographic voting. In *Proceedings of the 5th ACM Workshop on Privacy in Electronic Society*, WPES '06, pages 29–40, New York, NY, USA. ACM.

Ali, S. T. and Murray, J. (2016). An overview of end-to-end verifiable voting systems. *CoRR*, abs/1605.08554.

 © 2017 Sociedade Brasileira de Computação

Brunazo Filho, A. and Gazziro, M. A. (2014). Critérios para avaliação de sistemas eleitorais digitais. *XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Ed. por GRAAF, Jeroen van de, NOGUEIRA, José Marcos*, pages 599–610.

Culnane, C., Ryan, P. Y. A., Schneider, S., and Teague, V. (2015). vvote: A verifiable voting system. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30.

Filho, A. B., Carvalho, M. A. M., Teixeira, M. C., Jr, M. A. S., and Fernandes, C. T. (2015). Auditoria especial no sistema eleitoral 2014. *XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 511–522.

Karima, D., Victor, T., and Faycal, R. (2014). An improved electronic voting machine using a microcontroller and a smart card. In *IDT*, pages 219–224. IEEE.

Oro, I. A. B. (2015). `https://ivan.barreraoro.com.ar/vot-ar-una-mala-eleccion`. Acessado em 9 de Outubro, 2017.

Puri, T., Singh, J., and Kaushal, H. (2017). Prototyping of Indian electronic voting machine. *International Journal of Engineering Research and Development*, 13(5):44–51.

Rivest, R. (2002). Voting, homomorphic encryption. `http://web.mit.edu/6.857/OldStuff/Fall02/handouts/L15-voting.pdf`. Acessado em 9 de Setembro, 2017.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

van de Graaf, J. (2017). Long-term threats to ballot privacy. *IEEE Security & Privacy*, 15(3):40–47.

Wang, K.-H., Mondal, S. K., Chan, K., and Xie, X. (2017). A review of contemporary e-voting: Requirements, technology, systems and usability. *Data Science and Pattern Recognition*, 1(1):31–47.

 © 2017 Sociedade Brasileira de Computação

Variations on Wombat: end-to-end voting on a shoestring

Camilo Gómez[1], João Moreno[2], Jeroen van de Graaf[2], Alejandro Hevia[1]

[1]Universidad de Chile

[2]Universidade Federal de Minas Gerais

{cjgomez,ahevia}@dcc.uchile.cl, {joaomorenorf,jvdg}@dcc.ufmg.br

Abstract. *The motivation of this paper is providing end-to-end, cryptographically secure voting systems which are inexpensive, practical, and easy to deploy. To this end we review and discuss one particular family of dual electronic voting systems: those that derive from the Wombat voting system. Systems in this family support both paper as well as electronic tallying using QR codes, employ well documented cryptographic tools, and can be easily deployed in standard off-the-shelve equipment (smartphones, tablets, PCs, printers and scanners). In this paper, after describing the protocols in detail, we discuss some existent designs and implementations which provide improvements. We argue that these systems are desirable for small and medium-size organizations which often struggle to convince unsophisticated users that electronic elections are as secure as paper elections. We do not claim great novelty. Yet, we believe that bringing these prototypes and their properties to the forefront is important to understand how a new generation of voting systems with E2E security might look in practice.*

1. Introduction

The Wombat electronic voting system, first presented in 2011, offers *duality*: it combines a paper-based voting system with a mixnet-based, end-to-end cryptographic scheme. Moreover, the system provides a nice balance between cutting-edge cryptographically secure integrity and privacy guarantees, as well as physical, intuitive, administrative security [Benaloh 2008]. In the former, tools like public key encryption and zero-knowledge proofs help us to provide confidenciality of the votes, and integrity of the tally. In the later, physical requirements (e.g. transparent boxes), and procedural checks (each voter deposits one and only one piece of paper to the box) provide local poll workers with the feedback needed to convince themselves that the system was properly executed. Certainly, a system that appropriately combined both approaches is more likely to be adopted and used by non-technical communities.

More recently, in [van de Graaf 2017], the system was combined with a simplified version of Helios Voting [Adida 2008] to instantiate an easy to use, simple QR-code based, booth voting system. The resulting protocol is revisited here with additional diagrams in order to highlight several interesting properties. In particular, we point out that these protocol are perhaps the most suitable for low-budget, small and medium-size organizations where physical copies of the ballots are required in order to gain users' confidence. There are existent implementations for these protocols which run on reasonable inexpensive hardware such as standard PCs, smartphones, cameras, and basic printers.

We start by reviewing the Wombat system, and then we describe improvements and variants.

 © 2017 Sociedade Brasileira de Computação

2. Wombat voting

2.1. The original Wombat scheme

At the 2011 Dagstuhl Seminar Verifiable Elections and the Public, and in a more formal publication a year later at EVOTE [Ben-Nun et al. 2012], Ben-Nun et al. proposed a *dual* electronic voting system which combines the simplicity of paper voting with the cryptographic guarantees of mixnet-based, end-to-end voting. In a dual electronic voting system – first suggested by Benaloh [Benaloh 2008] – two voting systems are used in parallel: one electronic and another based on paper like those used in traditional elections in many countries.

High level description: Wombat carefully combines both schemes by having the voter produce a printed copy of her chosen candidate on the same piece of paper but in two different formats: in plain text (i.e. the name of the candidate), and in the form of a barcode of the corresponding encryption of the vote. The encrypted vote is then fed into an electronic system by requiring the voter to scan the barcode. In parallel, the ballot with the vote in plaintext is deposited in a traditional ballot box after detaching the barcode, which is returned to the voter as a receipt. Once all encrypted ballots have been cast, ballots are anonymized using a re-encryption mixnet, and then decrypted and counted to produce the election tally. The plain text ballots in the ballot box can be manually counted to verify the electronic tally. A simplified version of the system is described next.

In this system, it is assumed that election authorities perform an initial set up where private keys for some asymmetric encryption scheme (eg. ElGamal or Paillier) are jointly generated. Also, once the election started, poll workers are assumed to first verify the voter's identity and eligibility. Only then, the now anonymous voter is allowed access to the voting machine. The procedure is as follows:

The basic Wombat protocol

Step 1 The voter enters her choice into the voting machine. For simplicity we assume it is a choice between two options, encoded respectively as 0 and 1.

Step 2 The voting machine prints a ballot consisting of two parts: v, the voter's vote in plain text; and e, the same vote in encrypted form using a barcode. The encryption algorithm used is probabilistic so it requires a additional random bitstring which must be honestly generated (for example using smartcards [Ben-Nun et al. 2012]). So we have $e = E(v, r)$, where v is the vote, and r is an additional random string generated by the machine.

Step 3 The voter chooses: either to **audit** this ballot or to **cast** it.

Step 4 If the voter chooses to **audit** the ballot, the voting machine voids the ballot by printing audit information, namely the value r used for encryption, also in barcode format. This voided ballot, now containing the triple (v, e, r), is then taken to another equipment which scans v and r, computes $e' = E(v, r)$, and verifies whether $e' \stackrel{?}{=} e$. Today's smartphones are sufficiently powerful to perform this check. If the test fails then this is an indication that the voting machine produced an incorrect ballot, and an investigation is started. If the test passes, the voter returns to Step 1 to print a new ballot.

 © 2017 Sociedade Brasileira de Computação

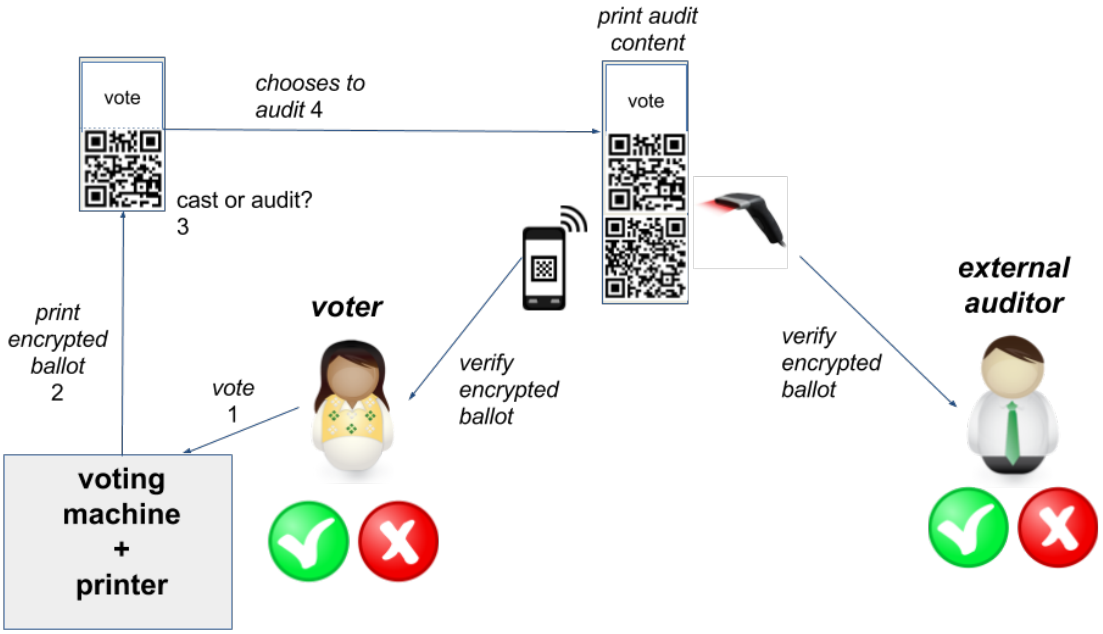


Figure 1. Wombat voting in case the voter chooses to audit the ballot. The numbers correspond to protocol steps.

Step 5 If the voter chooses to **cast** the ballot, she folds it over in a way to shield the plain text v printed on it, and hands it over to a poll worker. In front of the voter the bar code representing e is scanned. Then the ballot is divided in two, the folded part containing v is deposited in the ballot box, and the bar code containing e is returned to the voter as a receipt.

Step 6 After the election has ended, the authorities publish a list with all the receipts values e_i, i.e. the encrypted ballots received. This constitutes the digital equivalent of a ballot box. Conceptually this list looks as follows:

Number	Encrypted ballots
1	$e_1 = E(v_1, r_1)$
2	$e_2 = E(v_2, r_2)$
$\vdots$	$\vdots$
n	$e_n = E(v_n, r_n)$

Step 7 Once all the votes are cast and published, the authorities are able to tally the election results, while maintaining all the ballots encrypted. All ballots go through a re-encryption mixnet, and then the resulting vector is threshold decrypted by the authorities, who output a proof of correctness of the decryption process.

Step 8 (Optional) The authorities count the printed ballots v_i deposited in the ballot box.

 © 2017 Sociedade Brasileira de Computação

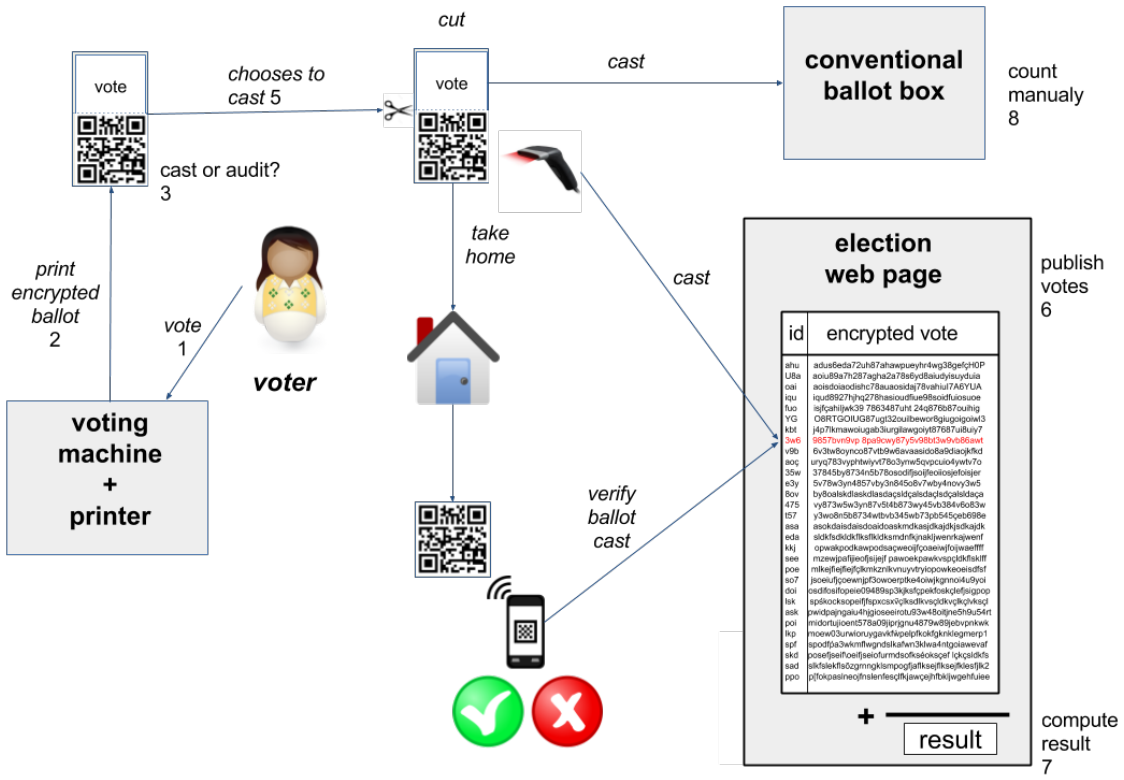


Figure 2. Wombat voting in case the voter chooses to cast the ballot. The numbers correspond to protocol steps.

2.2. Improvements and variants

In [van de Graaf 2017] a simplified voting system based on Wombat while preserving end-to-end security is described. The system relies on *homomorphic encryption*:

$$E(m_1) \cdot E(m_2) = E(m_1 + m_2).$$

This can be achieved, for example, using exponential ElGamal (see for instance [van de Graaf 2017]) or Paillier with a threshold shared key. Also, each ballot's correctness is proved using proofs of knowledge (eg. [Cramer et al. 1997] for ElGamal). Using these tools, the authorities can compute the election result in encrypted form. Then, they decrypt this value together, while providing a proof of correctness of this decryption process. Notice that, in this case, mixnets are no longer required.

3. Properties obtained

We review the main security properties satisfied by the system described above. This section is taken almost verbatim from [van de Graaf 2017].

Duality: This election system implements duality: the vote is printed on paper and also processed in electronic format. Obviously the tally of Step 7 should match the tally of Step 8. The advantage of paper is that it provides physical evidence of the voter's intention, therefore allowing a recount. The electronic format has the advantage

 © 2017 Sociedade Brasileira de Computação

of speed. There are various strategies to compare the two results. For instance, the more cumbersome counting of paper votes can be executed only if the election turns out to be tight. In addition, the hand counting can be executed in some randomly chosen precincts.

Individual verifiability: So how does this system accomplish individual verifiability? First, because of the audit possibility in Step 3 and 4, the voter can be reasonably sure that his vote is faithfully encrypted in e. Second, she receives this e as a receipt and, when the election is over, she can verify that it indeed appears on the election web page. Or she may delegate this task by showing or giving the receipt to somebody else she trusts, who performs this verification on her behalf. Assuming that any randomness used to generate the encrypted ballots was honestly generated (or it is jointly generated with the voter), if the receipt does not appear in the list published, this is strong evidence that a ballot went astray (especially if the receipt was digitally signed), and the owner of the receipt can go public about this and complain with the authorities. This not being the case, the voter can be convinced that her encrypted ballot is included in the list of ballots published by the authorities.

Universal verifiability: Observe that the tallying of the votes is publicly verifiable: any person, be it a voter or an outsider, can copy the encrypted ballots from the election web page, and either (a) verify the mixnet re-encryption and threshold decryption, or (b) reproduce the homomorphic tallying steps by computing the product and verifying that the authorities decrypted and published the correct result. This is what is meant by public verifiability: any interested person can verify the tally computed from the set of encrypted ballots. Note also that the election authorities cannot change the election result. Given the values of the second column it is mathematically impossible to change the election result without being caught.

Software independence: To emphasize the last point, we have achieved a property which is also known as software independence: the correctness of the election result no longer depends on the correctness of the software or hardware, or on the honest behaviour of election workers. As long as honest randomness is provided to the voter and the auditors certify the election outcome, the result must be correct. And this statement remains true even if the underlying software is incorrect or if the election authorities conspire. They could violate the privacy of the voters, but they cannot alter the election result. This is an important advantage over voting machines of earlier generations since they often consists of hundred of thousands of lines of source code, making it impossible to assert its correct functioning with certainty.

Privacy: As long as the encryption scheme is not broken, the information on the receipt and copied to the web page does not reveal the vote. We could even imagine that the serial numbers in the list are assigned to specific voters, allowing a closer control of who showed up to vote. These statements hold true provided that the authorities, that is, the trustees of the threshold shared master key K_{pr}, do not abuse their power. K_{pr} is only intended to decrypt e_T, but could, in principle, be used to decrypt any of the encrypted ballots. One way to avoid this is by destroying K_{pr} after successful decryption of e_T. Verifiably destroying cryptographic keys is not easy but is possible, requiring carefully elaborate procedures in which smart cards might be of help.

 © 2017 Sociedade Brasileira de Computação

3.1. Discussion

As far as we know, Wombat voting was the first voting system to use QR codes. The main difference with the system in [van de Graaf 2017] described above is the cryptographic back-end: above we use homomorphic tallying as proposed by [Cramer et al. 1997] (and used in the newer versions of Helios), whereas Wombat uses mixing. Mixing has the advantage that much more flexible ballot layouts are possible. On the other hand, the mixing takes a lot of processing time, whereas with homomorphic tallying most processing is done before the vote is cast while post-processing is modest. Wombat is currently being used at a specific university election, but otherwise it seems that no new development has taken place since 2012.

4. A concrete instantiation: the UFMG system

A basic prototype of Wombat was started at UFMG, implemented in Python. It uses Exponential ElGamal to encrypt the votes and compute the tally. The implementation is composed of five independent modules:

1. Voting booth: It assumes the voter have already been identified. It asks the desired vote, sends the result to the printer and then prompts the voter to cast or audit the encrypted ballot. If she wants to cast, it prints the receipt. If not, it prints the random value r used.
2. Audit ballot: This module receives the QR code for the ballot and the audit QR code, and then verifies the validity of the corresponding ballot by recomputing $E(v, r)$.
3. Scan ballot: This module scans the ballot QR code and receipt QR code and, if they are valid, casts the vote in the electronic ballot box.
4. Tally votes: It receives the electronic ballot box, computes the encrypted result using homomorphic tallying, and decrypts it using the shared private key.
5. Audit tally: Giving the list of encrypted votes, this module recomputes the encrypted result and verifies the official tally published.

4.1. Extension with digital signatures

A problem with the current system is that the receipts given to the voter and scanned by the system are not authenticated. We therefore envision the following straightforward extension of Wombat, in which both the Voting Booth and the Ballot Scanner produce a digital signature:

- In a ceremony before the elections, the election authorities create the private key used to encrypt the ballots. The private key is shared between various parties, whereas the public key is exported in the form of a self-signed certificate as a QR code, *QRcode*$(cert_{Elec})$. This actually is not an extension, but a crucial initialization step omitted in the protocol description above.
- Before the election starts, each Voting Machine scans *QRcode*$(cert_{Elec})$. The paper with the QR code can be displayed to the voter.
  The voting machine also produces a public-key/private key pair in order to sign each ballot, and two copies of *QRcode*$(cert_{VM})$ are printed, one to be displayed next to the VM; the second to be scanned by the Ballot Scanner and to be put on display there. After scanning has ended, *QRcode*$(cert_{VM})$ can go in the Conventional Ballot Box.

 © 2017 Sociedade Brasileira de Computação

- Before the election, each Ballot Scanner imports $\textit{QRcode}(cert_{VM})$ by scanning it, thus certifying the Voting Machine. The paper with the QR code can be put on display for the voter. In addition, the Ballot Scanner creates a public-key/private key pair and exports $\textit{QRcode}(cert_{BS})$.
- The Election Server must know $\textit{QRcode}(cert_{BS})$ so that it can verify the signatures produced by the Ballot Scanner, but this can be the same communication channel through which the encrypted ballots are sent, either an on-line internet connection, or some off-line connection using either CD-Rom or USB sticks.

Now he following additional verification steps are included:

- In Step 2 the Voting machine will digitally sign the ballot printed.
- In Step 4, when the voter audits the ballot, she will also check that the digital signature on the ballot holds.
- In Step 5, when the ballot is scanned, the Ballot Scanner verifies the digital signature on the ballot and only accepts ballots coming from machines that it certified. If it accepts the ballot, the Ballot Scanner creates a digital signature of the encrypted. This signature can be printed on one-sided adhesive sticker, which is then glued onto the receipt before it is handed back to the voter. (A unique ID on both receipt and sticker can be added to avoid voters peeling off the adhesive part and switch them.)

These additional measures enhance the security of the overall system in various respects:

- A ballot signed by the Voting Booth is irrefutable evidence that the ballot was produced by that machine, which is useful if the audit of Step 4 fails. On the other hand (and maybe more important) it makes it impossible to forge invalid ballots not produced by the Voting Machine, thus eliminating malicious attempts to undermine the system's credibility.
- The digital signature on the ballot implies a cryptographic verification that only votes will be cast coming from machines which were certified, making it impossible to include false ballots.

E-Voting as a Service: A very interesting feature of this improvement is its flexibility: the cryptographic equivalent of an election is the private key with which the ballots are encrypted. Associated to each election can be any number of Voting Machines and Ballot Scanners. For instance, if users take more time to vote than to scan, there could be polling stations with three Voting Machines and one Ballot Scanner.

Note also that Voting Machines and Ballot Scanners can serve various elections in parallel; this is just a matter of configuration. This could be a way to deal with multiple races, if it is decided that these should go on separate ballots. In cases where logically different polling stations are located in the same building, this can also help to pool resources, which can reduce waiting time and expenses on hardware.

Yet another variation is Bring-Your-Own-Ballot: dispose of the Voting Machine but have the voter bring a print-out of the ballot. This is essentially a paper-based equivalent of Helios, so the same precautions apply: the Ballot Scanner must verify the correctness of the format of the encrypted ballot, and voters can be subject to coercion. In particular, a voter could mark his vote by using a special kind of paper, causing this variation to be of little practical value.

 © 2017 Sociedade Brasileira de Computação

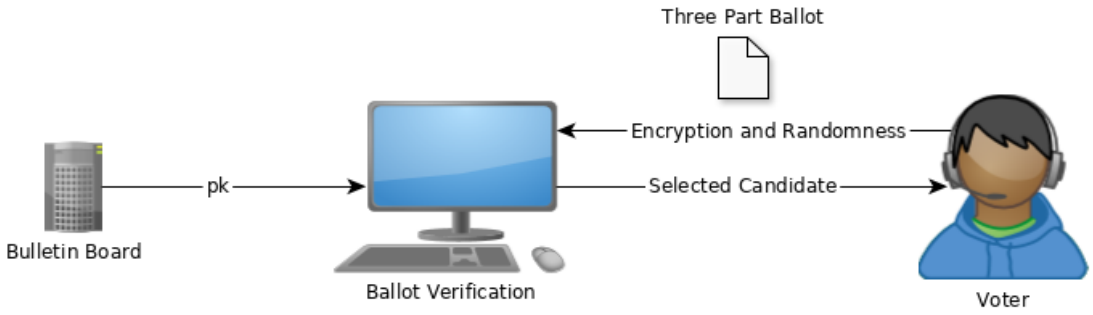


Figure 3. In the UChile voting system, the voter is able to verify the correctness of the encrypted ballot by adding a QR code with the randomness used for encryption. This randomness is verified by the audit machine, and then discarded before depositing the ballot.

5. Further extensions

Another extension based on Wombat was developed in a separate effort at UChile. Conceptually, it is closely related to the system described in Section 2, but with several minor differences:

1. The encrypted ballot is digitally signed *by the voter* before it is cast. In this stage, communication between the voting booth and the voter's smartphone takes place through reading and displaying QR codes. Figure 4 depicts this process.
2. Auditing the encryption of the ballot is mandatory. All ballots are printed with three parts: plain text vote, encrypted vote, and randomness used to encrypt. All voters must go to the auditing machine to verify that the encryption is correct. For this reason the auditing machine is very simple, currently being implemented on a Raspberry Pi. This is depicted in Figure 3. Ballot casting is a two step process: first the electronic (encrypted) version of the ballot is sent, then the physical paper version is deposit on the ballot box. This is shown in Figure 5. This machine could be implemented by several organizations (political parties, NGOs, government, etc), and the voter could choose the machine which was developed by an organization he trusts.
3. The system is implemented mainly in Java and for Android. There are several mobile applications implemented in order to help deployment and improve user experience of all actors: authorities, administrators, voters and external observers. The implementation relies on Paillier encryption scheme.

More details on this implementation are available online [Gómez 2017] including a short video demonstration.

6. Observations and conclusions

This paper presents Wombat voting, and some variations based on this protocol. Our goal is to describe and implement a system which is end-to-end, flexible, and which needs no specific hardware, implying that any organization with a modest budget can implement fair elections.

 © 2017 Sociedade Brasileira de Computação

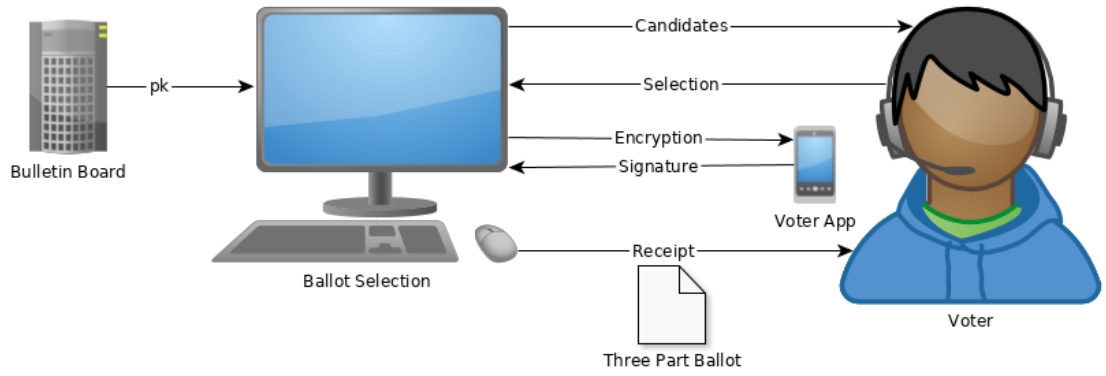


Figure 4. In the UChile voting system, the voter signs her vote using her smartphone to capture the data to sign (the encrypted ballot) and displaying a QR-code representation of the signed encrypted ballot.

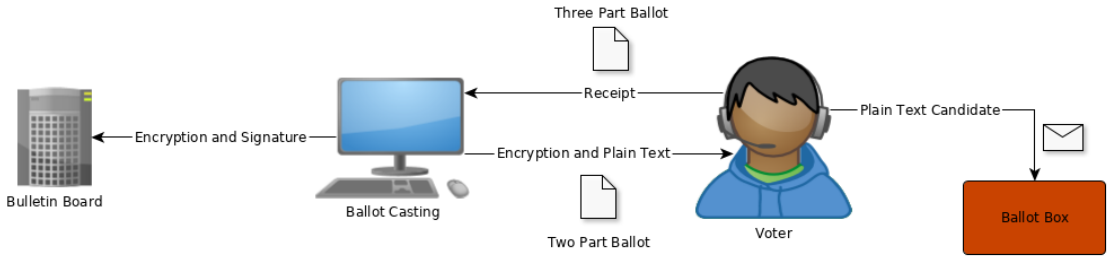


Figure 5. In the UChile voting system, the voter cast her ballot in a two step process: first it submits her signed ballot to the ballot casting machine which checks her signature. Then it detaches the encryption QR code and deposits only her plaintext ballot in the ballot box.

 © 2017 Sociedade Brasileira de Computação

We are aware that all the ideas presented here have been incorporated in STAR-Vote (see [Benaloh et al. 2013]), but we believe that a simplified system will be useful in itself, as well as help in understanding the more sophisticated features of STAR-Vote.

References

Adida, B. (2008). Helios: Web-based open-audit voting. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348. USENIX Association.

Ben-Nun, J., Fahri, N., Llewellyn, M., Riva, B., Rosen, A., Ta-Shma, A., and Wikström, D. (2012). A new implementation of a dual (paper and cryptographic) voting system. In *EVOTE 2012*, pages 315–329.

Benaloh, J. (2008). Administrative and public verifiability: can we have both? *EVT*, 8:1–10.

Benaloh, J., Byrne, M. D., Eakin, B., Kortum, P. T., McBurnett, N., Pereira, O., Stark, P. B., Wallach, D. S., Fisher, G., Montoya, J., Parker, M., and Winn, M. (2013). STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. In *EVT/WOTE*.

Cramer, R., Gennaro, R., and Schoenmakers, B. (1997). A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 103–118. Springer.

Gómez, C. (2017). The Moca QR Voting System. `http://cjgomez.cl/moca`.

van de Graaf, J. (2017). Long-term threats to ballot privacy. *IEEE Security & Privacy*, 15(3):40–47.

 © 2017 Sociedade Brasileira de Computação

WTICG – XI Workshop de Trabalhos de Iniciação Científica e de Graduação

© 2017 Sociedade Brasileira de Computação

A integração do Criptossistema El Gamal Limiar ao Sistema de Votação via Internet CIVIS

André A. Silva Neto, Roberto Araújo

[1]Faculdade de Computação – Universidade Federal do Pará (UFPA)
Belém – PA – Brasil

andre.neto@icen.ufpa.br, rsa@ufpa.br

Abstract. *Modern digital voting systems typically employ threshold cryptosystems to ensure secure voting. These cryptosystems help ensure the secrecy of the vote and bring as a main benefit the decentralization of the decryption process. In this way, they prevent a single voting authority from having full control over that process. In this context, this work presents an implementation of an El Gamal threshold cryptosystem and its integration to the Internet voting system CIVIS.*

Resumo. *Sistemas modernos de votação digital normalmente empregam criptossistemas limiares a fim de garantir votações seguras. Esses criptossistemas ajudam a preservar o sigilo do voto e trazem como principal benefício a descentralização do processo de apuração. Dessa forma, eles evitam que uma única autoridade de votação tenha total controle sobre esse processo. Neste contexto, este trabalho apresenta uma implementação de um criptossistema El Gamal limiar e sua integração ao sistema de votação via Internet CIVIS.*

1. Introdução

Um dos requisitos de segurança de sistemas para votação é garantir o sigilo dos votos. Uma das primitivas criptográficas utilizadas para esse fim é um criptossistema de chave pública. Ele permite que os votos sejam encriptados pelos votantes e, posteriormente, decriptados durante a apuração.

O emprego de criptossistemas de chave pública convencionais em sistemas de votação, no entanto, possui como desvantagem a confiança em uma única autoridade de votação. Em caso de corrupção, essa autoridade poderia facilmente comprometer o sigilo dos votos. Devido a problemas desse tipo, criptossistemas limiares tornaram-se ferramentas fundamentais em sistemas modernos de votação. Nesse tipo de criptossistema, a geração do par de chaves (pública e privada) é realizada de forma distribuída. Existe uma única chave pública e a chave privada correspondente é composta de segredos compartilhados entre um conjunto de n participantes. De forma a decriptar um texto cifrado, é necessário que qualquer subconjunto de t participantes utilizem seus segredos para realizar essa operação.

Em sistemas de votação via Internet, particularmente, criptossistemas limiares reduzem o risco de ocorrer a decriptação dos votos antes da apuração tendo em vista que é necessário um conjunto mínimo de autoridades para realizá-la. Dessa forma, esses criptossistemas reduzem o problema da confiança em uma única autoridade causado pelo uso de criptossistemas convencionais.

 © 2017 Sociedade Brasileira de Computação

Neste contexto, este trabalho apresenta uma implementação do criptossistema El Gamal limiar e sua integração ao sistema de votação via Internet CIVIS. Esse sistema implementa um protocolo que possibilita aos votantes resistirem a ataques coercivos, i.e., ataques onde o votante é forçado a votar em uma determinada opção. Ele tem como um de seus requisitos um criptossistema limiar.

2. Trabalhos Relacionados

Na literatura, existem diversas implementações de esquemas de criptografia limiar para os mais diversos tipos de aplicações. Em Mohammadi [Mohammadi and Jahanshahi 2009], por exemplo, o conceito é aplicado a um sistema de e-tendering. Ou seja, um sistema de criação de contratos via Internet. O trabalho apresenta uma implementação do sistema de e-tendering, que utiliza um esquema de criptografia limiar para distribuir a encriptação dos documentos do contrato entre as partes envolvidas no acordo.

O trabalho de Ertaul e Chavan [Ertaul and Chavan 2007], por sua vez, apresenta implementações do criptossistema RSA limiar e do criptossistema de curvas eliptícas El Gamal limiar. A ideia é que estes criptossistemas sejam utilizados no envio seguro de dados em MANET (Redes Móveis Ad Hoc). A utilização de um criptossistema limiar permite que cada nó da rede possa participar da encriptação destes dados.

Davis et al. [Davis et al. 2008] implementou um criptossistema limiar e o empregou no sistema de votação eletrônica Civitas. A criptografia limiar também foi utilizado no sistema de votação eletrônica Helios [Cortier et al. 2013]. O trabalho Cortier et al. destaca a realização de votações com o resultados mais confiáveis, a partir da utilização de um criptossistema limiar. Assim como apresentado neste trabalho, o objetivo destes trabalhos foi distribuir o processo de encriptação dos votos entre um grupo de autoridades da eleição.

3. Fundamentos Teóricos

Esta seção apresenta as principais tecnologias empregadas nesse trabalho.

3.1. O Criptossistema El Gamal Limiar

Em um criptossistema de chave pública (t, n)-limiar a chave privada é formada por um conjunto de segredos compartilhados por n participantes. Para decriptar uma mensagem é necessário que pelo menos t participantes utilizem seu segredo. Define-se criptossistema El Gamal limiar como uma versão limiar do criptossistema El Gamal convencional. O criptossistema apresentado nesta seção é utilizado como base da implementação e integração realizadas neste trabalho.

Tal criptossistema requer a aplicação de um esquema de compartilhamento seguro de segredos. Este esquema visa dividir um segredo S em n partes $S_1, ..., S_n$ de forma que quaisquer t (onde t é definido previamente) ou mais S_i possam facilmente recuperar S e quaisquer $t - 1$ ou menos S_i não revelem nenhuma informação sobre S [Shamir 1979]. Este trabalho utiliza o termo fração de segredo [1] para referenciar a informação S_i entregue a cada membro sobre o segredo compartilhado neste esquema.

[1] O termo "fração de segredo"é uma tradução de "secret share"citado por Shamir em [Shamir 1979].

 © 2017 Sociedade Brasileira de Computação

É utilizado ainda o termo parte de chave secreta como referência a informação D_i pertencente a cada membro, que poderá ser utilizada no processo de decriptação. No criptossistema empregado as frações de segredo são objetos utilizados na geração e distribuição das chaves do grupo, enquanto as partes de chave secreta são objetos utilizados na decriptação (mais precisamente, cada D_i é uma combinação de S_i's).

A decriptação ocorre quando é recebido um texto encriptado E_{pkG} com a chave pública do grupo pkG. Neste processo, cada membro pode utilizar D_i e E_{pkG} para gerar um texto parcial T_i, ou seja, uma parte do cálculo de decriptação. Caso t ou mais T_i forem combinados, estes retornarão o texto decriptado.

O criptosistema El Gamal limiar utilizado neste trabalho foi proposto por Pedersen [Pedersen 1991]. No entanto, o protocolo de decriptação aplicado é o de Cramer et al. [Cramer et al. 1997]. O criptossistema tem como base o esquema de compartilhamento de segredos de Shamir [Shamir 1979]. O esquema de Shamir, por sua vez, utiliza o método de interpolação polinomial de Lagrange [Wagstaff Jr 2002].

Considerando que os parâmetros que determinam um grupo seguro estão disponíveis (e.g. um grupo cíclico G de ordem p e um gerador g onde o problema de decisão de Diffie-Hellman [Boneh 1998] é difícil), a seguir são apresentados os algoritmos necessários ao criptossistema.

Geração de chaves. Seja um grupo de n membros, cada membro gera um par de chaves El Gamal convencional $(x_i, h_i = g^{x_i})$, onde x_i é a chave privada e h_i é a chave pública. Esses pares de chaves são utilizados para obter sigilo no processo de geração do par de chaves limiar. As chaves de todos os membros são utilizadas na geração da chave pkG e das partes D_i de cada membro;

Compartilhamento de Chaves Privadas. Cada membro compartilha sua x_i em n frações S_i e as entregam aos outros membro do grupo. Para isso utilizam o esquema de Shamir. A entrega das S_i deve ocorrer de forma secreta;

Verificação das Frações de Segredo. Os membros verificam a consistência de cada S_i recebida utilizando dados publicados pelos outros membros;

Geração das Partes de Chave Secreta. Cada membro pode, então, gerar sua parte D_i, combinando as frações S_i recebidas de cada membro. D_i é mantida em segredo pelo membro do grupo;

Geração da Chave Pública do Grupo. pkG é calculada a partir das chaves h_i de cada membro do grupo;

Decriptografia Distribuída. Pelo menos t membros do grupo devem gerar textos parciais T_i (e provas de conhecimento zero atestando a validade de T_i) utilizando suas respectivas D_i. Assim que houver t ou mais T_i com provas válidas, estes podem ser combinados, resultando no texto decriptado.

3.2. O Sistema de Votação via Internet CIVIS

O CIVIS é um sistema de votação *Web* seguro que implementa o protocolo de votação de Araújo et al. [Araújo et al. 2008]. Esse sistema foi desenvolvido na linguagem de programação Python [Van Rossum and Drake 2003], utilizando o framework Django [Holovaty and Kaplan-Moss 2009]. O CIVIS utiliza-se de uma série de mecanismos criptográficos (redes de misturadores, provas de conhecimento zero, etc) a fim de garantir a segurança de eleições. Uma das principais características desse sistema é a resistência

 © 2017 Sociedade Brasileira de Computação

a ataques coercivos. Ou seja, ataques em que os votantes são forçados a votar em uma determinada opção.

Neste sistema, uma eleição é conduzida por autoridades, que são usuários responsáveis pelas etapas do sistema. Cada eleição possui três autoridades: a autoridade de eleição (AE), responsável pelas configurações gerais da eleição; a autoridade de registro (AR), responsável por autorizar os votantes a participar da eleição; e a autoridade de apuração (AP), responsável pela verificação, criptografia e contabilização dos votos enviados a eleição.

Uma eleição no CIVIS utiliza-se de um quadro público para disseminar toda informação pública referente a eleição. Ele contém, por exemplo, todos os votos encriptados enviados a eleição, além das chaves públicas de AP e AR e outros dados públicos. Cada eleição no sistema é dividida em quatro fases: configuração, registro, votação e apuração.

Na fase de configuração são definidos os parâmetros iniciais da eleição, como título, disputas, votantes, entre outros. Nesta fase, AP e AR geram seus respectivos pares de chave El Gamal. Estas chaves serão utilizadas com diferentes propósitos no decorrer da eleição. A Figura 1 apresenta página inicial do sistema.

Figura 1. Página inicial do sistema CIVIS contendo as eleições em andamento.

Após a configuração, ocorre a fase de registro. Nesta fase, AR é responsável pela autorização dos votantes que participarão de uma eleição. Posteriormente, na fase de votação, os votantes devem enviar seus votos para o sistema. No CIVIS, os votos são encriptados pelo votante antes do envio ao sistema. Para isso, até recentemente, era utilizado o criptossistema El Gamal convencional. O votante utiliza a chave pública de AP (gerada na fase de configuração) para criptografar seu voto.

Na fase de apuração, os votos são autorizados e decriptados. Nesta fase, cada voto encriptado passa por uma série de verificações, atestando sua validade para a eleição. Os votos considerados válidos serão decriptados pela AP utilizando sua chave privada. A AP então publica no quadro público o resultado final da eleição.

4. O Criptossistema El Gamal limiar e sua Integração ao CIVIS

A partir da introdução do criptossistema El Gamal limiar e do sistema de votação CIVIS apresentados na seção anterior, a seguir são apresentados a implementação desse criptossistema e sua integração ao sistema CIVIS.

 © 2017 Sociedade Brasileira de Computação

4.1. A Implementação do Criptossistema El Gamal Limiar

O criptossistema foi implementado utilizando a linguagem de programação JavaScript [Flanagan 2006] para possibilitar a realização de operações localmente, como a geração dos segredos. Além disso, também foi utilizado o formato para troca de dados JSON [Bray 2014], que torna mais fácil a integração do criptossistema a outras aplicações.

A implementação segue os protocolos de Pedersen e de Crammer et al. para realização de dois processos principais do criptossistema: a seleção e distribuição das chaves e a decriptação de textos cifrados.

Arquitetura do Criptossistema

O criptosistema foi implementado de forma modular. Para isso, ele foi dividido em quatro módulos: Pedersen, Shamir, Lagrange e Crammer. Cada módulo representa um método ou protocolo implementado no criptossistema. A Figura 2 ilustra a interação entre esses módulos.

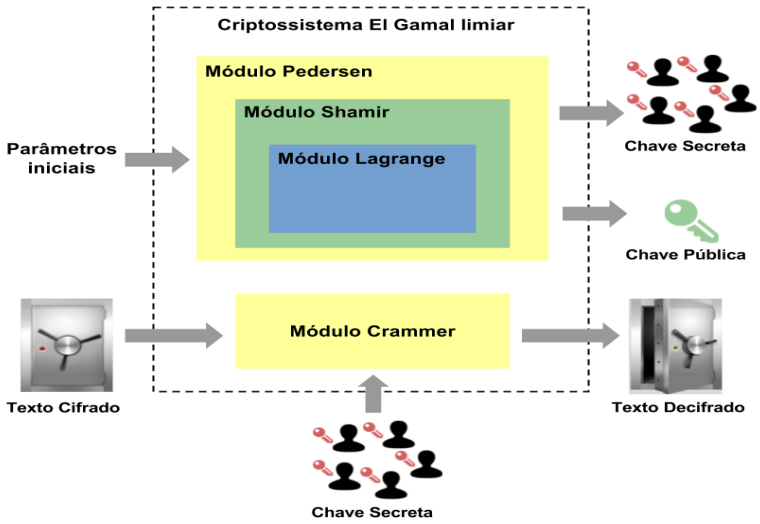


Figura 2. Visão geral do criptossistema implementado.

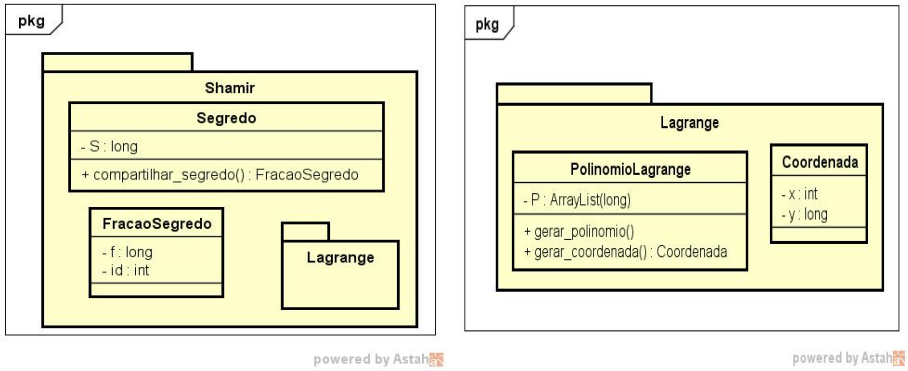
Os módulos Pedersen, Shamir e Lagrange recebem como entrada parâmetros iniciais referentes a definição do grupo seguro. O módulo Lagrange implementa o método de interpolação polinomial de Lagrange. Neste módulo são implementados métodos matemáticos, como polinômios e coordenadas geométricas. Tais métodos auxiliam no compartilhamento de segredos de Shamir, implementado no módulo Shamir [Wagstaff Jr 2002].

O módulo Pedersen, por sua vez, realiza a geração das chaves do grupo (i.e. as partes de chave secreta e a chave pública do grupo). Ele requer o módulo Shamir na sua execução. O módulo Crammer realiza a decriptação distribuída de textos cifrados. A seguir são apresentadas as classes presentes em cada um destes módulos.

A Figura 3(b) ilustra as classes que implementam o método de interpolação polinomial de Lagrange. As classes deste módulo representam ferramentas matemáticas

 © 2017 Sociedade Brasileira de Computação

utilizadas pelo módulo Shamir, são elas: *PolinomioLagrange* e *Coordenada*. O módulo Shamir (ver Figura 3(a)), por sua vez, utiliza diretamente estas classes para realizar o compartilhamento do segredo (classe *Segredo*) em frações de segredo (classe *FracaoSegredo*). Ambas classes *Segredo* e *FracaoSegredo* fazem parte do módulo Shamir.



(a) Diagrama do módulo Shamir. (b) Diagrama do módulo Lagrange.

Figura 3. Diagrama de classes dos módulo Shamir e Lagrange.

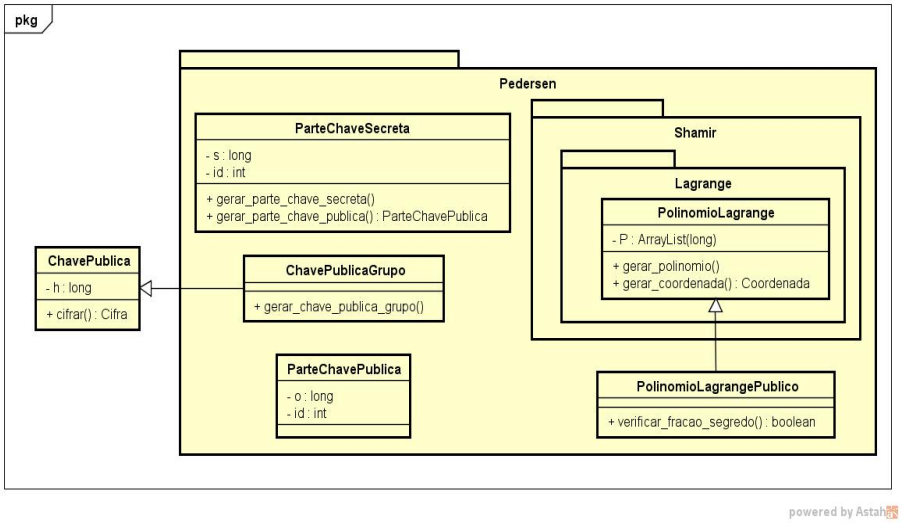


Figura 4. Diagrama de classes do módulo Pedersen.

No módulo Pedersen (ver Figura 4), além dos módulos Shamir e Lagrange, ocorrem as operações mais importantes para geração das chaves do grupo. Estão inclusas neste módulo as classes *ChavePublicaGrupo*, *ParteChaveSecreta*, *ParteChavePublica* e *PolinomioLagrangePublico*. A classe *ChavePublicaGrupo* é fundamental na representação da chave utilizada para o envio de mensagens criptografadas ao grupo. A classe *ParteChaveSecreta* representa as partes de chave secreta mantidas pelos membros do grupo. As classes *ParteChavePublica* e *PolinomioLagrangePublico* têm uma função mais específica para a verificação das frações de segredo.

Por fim, o módulo Crammer (ver Figura 5) apresenta as classes: *TextoParcialmenteDecifrado* e *ProvaTextoParcialmenteDecifrado*. Estas representam, respectivamente, os

 © 2017 Sociedade Brasileira de Computação

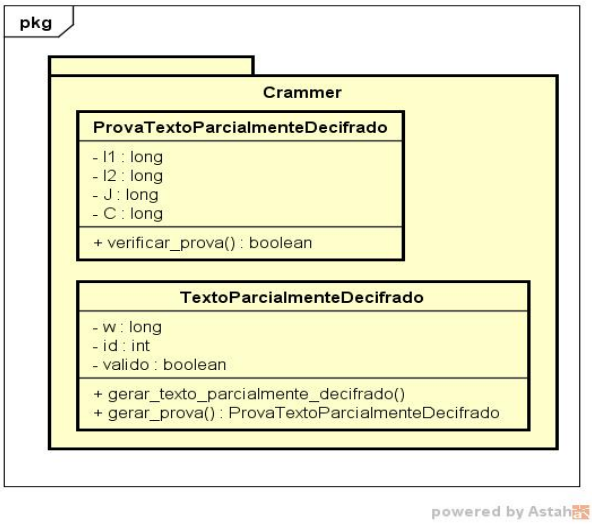


Figura 5. Diagrama de classes do módulo Crammer.

textos parciais enviados por cada membro para decriptação e suas provas de conhecimento zero correspondentes, que atestam que o texto parcial foi gerado de forma correta. Como descrito na Seção 3.1, no protocolo de Crammer et al., os textos parciais são parte da decriptação gerados pelos membros do grupo.

4.2. A Integração ao sistema CIVIS

A partir da implementação do criptossistema apresentado na seção anterior foi possível sua integração com sistema CIVIS. O objetivo dessa integração é compartilhar a responsabilidade pela criptografia dos votos da eleição por um grupo de apuradores (AP) possibilitando uma apuração distribuída dos votos. Como resultado, reduziu-se os riscos relativos a APs maliciosas. Além disso, a integração possibilitou complementar o sistema CIVIS tendo em vista que o criptossistema implementado é um dos requisitos do protocolo implementado por ele.

Com a integração, as funções do criptossistema são executadas no decorrer das páginas do CIVIS. Inicialmente foi planejada uma sequência de execução do criptossistema, o que permitiu uma visão sobre as páginas em que cada função poderia ser inserida no sistema. Com isso, houveram mudanças aplicadas a estas páginas no sistema.

Além disso, a estrutura do banco de dados do CIVIS também foi adaptada para receber as informações geradas pelo criptossistema. Informações públicas (que devem ser conhecidas pelos membros do grupo) são salvas no banco de dados do CIVIS e podem ser acessados pelos membros a qualquer momento após a publicação.

Antes da execução do criptossistema limiar, ocorre a definição dos parâmetros t e n. Eles são escolhidos pela AE na página da criação da eleição (ver Seção 3.2). Mais adiante, a AE também define as APs que farão parte do grupo.

A geração das chaves ocorre posteriormente, na fase de configuração da eleição, com a participação de todas as APs definidas. A Figura 6 apresenta a página *Web* onde as autoridades colaborativamente selecionam as chaves. Cada AP deve executar quatro etapas: Gerar Par de Chaves (1); Distribuir Chave Secreta (2); Verificar Frações de Segredo

 © 2017 Sociedade Brasileira de Computação

(3); e Gerar Parte de Chave Secreta (4). Cada etapa é liberada assim que todas as APs no grupo tiverem realizado a etapa anterior.

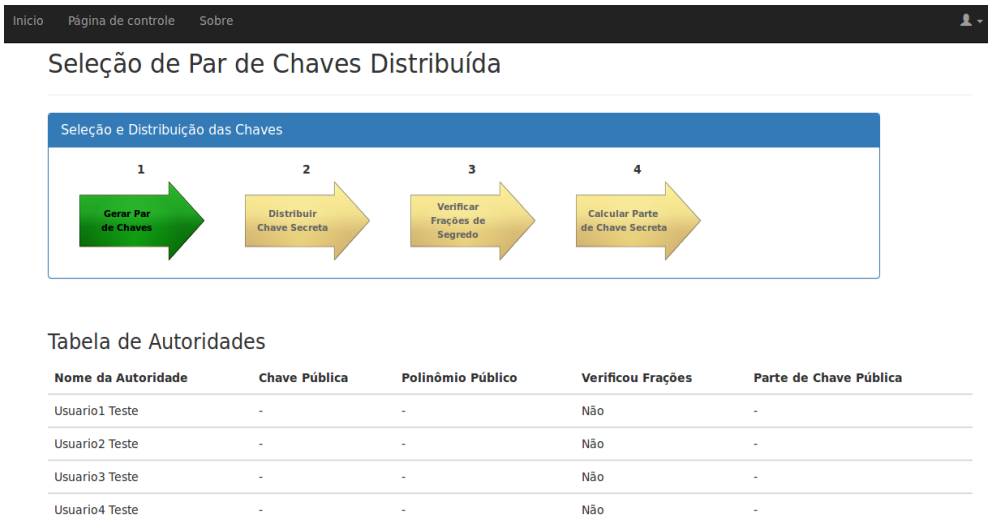


Figura 6. Página para geração das chaves distribuída no CIVIS.

Na etapa 1, a AP gera o seu próprio par de chaves El Gamal, mantendo em segredo sua chave privada e publicando sua chave pública para os outros membros. Como destacado na Seção 3.1 a partir deste par de chaves próprio de cada membro (neste caso AP) são geradas a chave pública do grupo e as partes de chave secreta de cada membro. Além disso, destaca-se que este par é utilizado na integração para a troca confidencial de frações de segredo entre as autoridades no grupo.

Na etapa 2, a autoridade aplica o esquema de Shamir para compartilhar sua chave privada entre os membros. A autoridade deve entregar, de forma encriptada, a cada membro uma fração de sua chave. Além disso, a autoridade publica outras informações para fins de verificação das frações geradas.

A etapa 3 consiste na verificação da consistência das frações recebidas. Para isso a AP deve, inicialmente, decriptar as frações recebidas utilizando sua chave privada. Após isso, a autoridade deve verificar a consistência de cada fração recebida utilizando os dados publicados pela respectiva autoridade geradora.

A última etapa consiste no cálculo da parte de chave secreta. Nesta etapa a AP deve utilizar todas as frações válidas recebidas para realizar seu cálculo. Na Figura 6, abaixo das etapas, é possível observar a Tabela de Autoridades. Nesta tabela, existente no quadro público, são apresentados todos os dados públicos produzidos pelas autoridades como chaves públicas e polinômios públicos. Com a finalização da etapa 4 por todas as autoridades, o sistema deve encerrar a geração das chaves e calcular a chave pública do grupo.

A fase de registro da eleição ocorre sem influência do criptossistema limiar. Na fase de votação, a única mudança será a troca da chave pública utilizada para encriptação os votos (com a integração, a chave pública do grupo será utilizada).

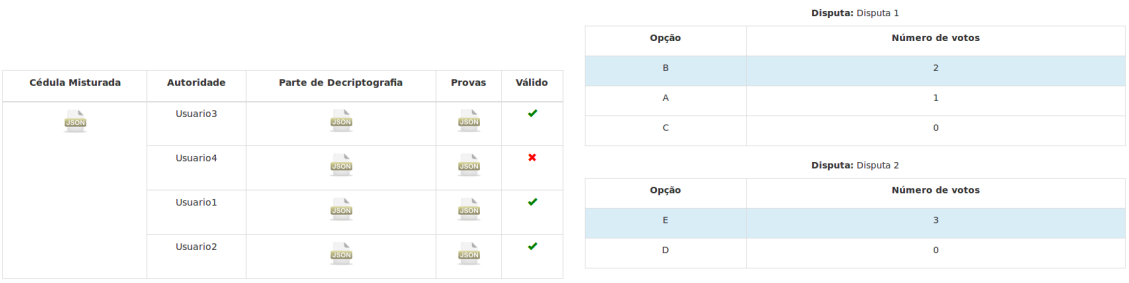
Na fase de apuração ocorre a apuração distribuída dos votos. As APs que desejarem participar deste processo devem acessar a página apresentada na Figura 7. Nesta

 © 2017 Sociedade Brasileira de Computação

Figura 7. Página para decriptar votos distribuidamente no CIVIS.

página a autoridade deverá inserir sua parte de chave secreta para o cálculo de seus textos parciais e respectivas provas. Cada AP deve gerar um texto parcial e uma prova que empregou corretamente seu segredo para cada voto válido da eleição.

Ambos textos parciais e provas são publicados no quadro público da eleição. Os resultados da verificação da provas (realizada pelo sistema) também são publicados nesse quadro. Assim que houver um número limiar de textos parciais com provas válidas para cada voto, o sistema realiza o cálculo definido no protocolo de Crammer et al. para decriptar os votos. Os resultados serão então publicados também no quadro público da eleição. Na Figura 8 são exibidas as páginas do quadro público. Na Figura 8(a) são exibidos os textos parciais (coluna Parte de Decriptografia) e suas provas para um único voto. Destaca-se a existência de um texto parcial com a prova inválida (Usuário 4). Na Figura 8(b) é exibida a página com os resultados finais da eleição.



(a) Textos parciais e provas publicados.

(b) Resultados da eleição.

Figura 8. Páginas do quadro público de uma eleição no CIVIS.

5. Considerações Finais

Um dos principais requisitos de sistemas de votação digital é garantir o sigilo dos votos. Para isso, são necessários o uso de mecanismos criptográficos para este fim. Este trabalho apresentou a implementação de um criptossistema El Gamal limiar e sua integração ao sistema de votação CIVIS.

A utilização desse criptossistema em sistemas de votação distribui a apuração dos votos entre as autoridades evitando que uma única autoridade seja responsável por isso. A

© 2017 Sociedade Brasileira de Computação

implementação do criptossistema produziu como resultado uma biblioteca que pode ser aplicada a diversos sistemas.

A integração do criptossistema ao CIVIS possibilitou um aumento de segurança nesse sistema. Ele permitiu a adição de múltiplas autoridades ao sistema e a decriptação de votos de forma distribuída.

Como trabalho futuro, pretende-se avaliar o desempenho da implementação considerando uma grande quantidade de votos, votantes e autoridades no sistema CIVIS.

Referências

Araújo, R., Foulle, S., and Traoré, J. (2008). A practical and secure coercion-resistant scheme for remote elections. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Boneh, D. (1998). The decision diffie-hellman problem. In Buhler, J., editor, *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer.

Bray, T. (2014). The javascript object notation (json) data interchange format.

Cortier, V., Galindo, D., Glondu, S., Izabachene, M., et al. (2013). A generic construction for voting correctness at minimum cost-application to helios. *IACR Cryptology EPrint Archive*, 2013:177.

Cramer, R., Gennaro, R., and Schoenmakers, B. (1997). A secure and optimally efficient multi-authority election scheme. *Transactions on Emerging Telecommunications Technologies*, 8(5):481–490.

Davis, A. M., Chmelev, D., and Clarkson, M. R. (2008). Civitas: Implementation of a threshold cryptosystem.

Ertaul, L. and Chavan, N. (2007). Elliptic curve cryptography based threshold cryptography (ecc-tc) implementation for manets. *IJCSNS*, 7(4):48.

Flanagan, D. (2006). *JavaScript: the definitive guide*. "O'Reilly Media, Inc.".

Holovaty, A. and Kaplan-Moss, J. (2009). *The definitive guide to Django: Web development done right*. Apress.

Mohammadi, S. and Jahanshahi, H. (2009). A secure e-tendering system. In *Electro/Information Technology, 2009. eit'09. IEEE International Conference on*, pages 62–67. IEEE.

Pedersen, T. (1991). A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT'91*, pages 522–526. Springer.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

Van Rossum, G. and Drake, F. L. (2003). *Python language reference manual*. Network Theory.

Wagstaff Jr, S. S. (2002). *Cryptanalysis of number theoretic ciphers*. CRC Press.

 © 2017 Sociedade Brasileira de Computação

Avaliação de assinaturas baseadas em *hash* para a Internet das Coisas

Jéssica C. Carneiro[1], **Leonardo B. Oliveira**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

{jessicacarneiro,leonardo.barbosa}@dcc.ufmg.br

Abstract. *With the advent of quantum computing, traditional signatures such as RSA, DSA, and ECDSA will become unsafe thanks to Shor's algorithm. Thus, cryptosystems resilient to quantum computing must be studied, specially for devices in the Internet of Things which limitations may hinder the use of secure cryptosystems. Alternatives are the hash-based signatures (HBS) which are post-quantum and provide a trade-off between performance and security level. This paper presents an implementation of HBS schemes on an Arduino Due platform and a performance evaluation.*

Resumo. *Com o advento da computação quântica, assinaturas tradicionais como RSA, DSA e ECDSA tornarão-se inseguras graças ao algoritmo de Shor. Dessa forma, criptossistemas resilientes à computação quântica devem ser estudados, especialmente para dispositivos da Internet das Coisas cujas limitações podem dificultar a utilização de criptossistemas seguros. Uma alternativa são as assinaturas baseadas em* hash *(*hash-based signatures - *HBS) que são pós-quânticas e proveem um compromisso entre desempenho e nível de segurança. Este trabalho apresenta a implementação de esquemas HBS sobre a plataforma Arduino Due e uma avaliação de desempenho.*

1. Introdução

Assinaturas digitais são uma forma de associar informações a uma entidade, seja ela uma pessoa ou um dispositivo [Menezes et al. 1996]. Elas fornecem autenticação, integridade e irretratabilidade de informações trocadas em uma rede. Atualmente, assinaturas digitais são amplamente utilizadas em serviços na Web, computadores pessoais, dispositivos IoT, etc. Alguns esquemas de assinatura tais como RSA [Rivest et al. 1978], DSA [ElGamal 1985] e ECDSA [Johnson et al. 2001] estão entre os criptossistemas de chave pública mais conhecidos e utilizados [Buchmann et al. 2009].

Os criptossistemas citados têm sua segurança pavimentada na intratabilidade de problemas matemáticos tais como fatoração de números inteiros (RSA), logaritmos discretos (DSA) e logaritmos discretos em curvas elípticas (ECDSA). No entanto, a descoberta por Shor [Shor 1994] de algoritmos polinomiais quânticos para resolução de tais problemas os tornaria inseguros. Consequentemente, alternativas devem ser avaliadas para o período pós-quântico.

É especialmente necessário avaliar criptossistemas pós-quânticos adequados para a Internet das Coisas pois espera-se que o número de dispositivos cresça significantemente[1]. Dispositivos para a IoT possuem recursos computacionais restritos o que impõe

[1]http://www.gartner.com/newsroom/id/2636073

 © 2017 Sociedade Brasileira de Computação

uma limitação a quais criptossistemas são viáveis para tais plataformas. As assinaturas baseadas em *hash* (*hash-based signatures* - HBS) são alternativas possíveis para a IoT pois dependem apenas de funções *hash* e não são necessárias operações caras como no esquemas RSA e ECDSA, por exemplo.

As HBS permitem um compromisso entre desempenho e nível de segurança. Também, caso a função *hash* empregada seja considerada insegura, o criptossistema pode continuar sendo utilizado apenas trocando-se a função, o que fornece uma maior flexibilidade às assinaturas. Além disso, sua segurança depende somente da propriedade de resistência à colisão da função escolhida [Buchmann et al. 2009].

A escolha pelas HBS se deu pelo fato de elas serem rápidas, possuírem chaves menores e sua segurança ser melhor entendida em relação a outras alternativas pós-quânticas como Criptografia baseada em Reticulados, Criptografia baseada em Códigos Corretores de Erro e Criptografia Multivariada [Hülsing et al. 2015].

As contribuições deste trabalho são:

1. Uma explicação didática das HBS na seção 2;
2. Implementação dos esquemas Winternitz (W-OTS) e Merkle *Signature Scheme* (MSS) na seção 3;
3. Avaliação de desempenho em termos de armazenamento, processamento e comunicação na seção 4;
4. Reunião de outros trabalhos sobre esquemas pós-quânticos e sobre HBS em dispositivos restritos na seção 5.

2. Fundamentação Teórica

As HBS foram criadas por Ralph Merkle em [Merkle 1989]. Elas ganharam atenção como alternativas no cenário pós-quântico aos esquemas usados atualmente. Sua construção requer somente a existência de funções *hash* criptográficas e sua segurança depende da propriedade de resistência à colisão da função usada [Buchmann et al. 2009].

Dentre as principais vantagens das HBS estão: (i) a simplicidade de sua implementação; (ii) sua segurança é bem entendida [Buchmann et al. 2011]; (iii) suas chaves e assinaturas são menores que outras opções pós-quânticas (exemplo: Criptografia baseada em Reticulados); (iv) o tempo gasto na geração de assinaturas e verificação é competitivo mesmo em relação a esquemas tradicionais [Hülsing et al. 2012]. Algumas desvantagens são o número limitado de assinaturas que podem ser geradas com um par de chaves e a impossibilidade de utilizá-las para encriptação.

2.1. Tipos de HBS

Os esquemas HBS são divididos em dois tipos principais: *One-Time Signatures* (OTS) e *Multi-Time Signatures* (MTS). O primeiro permite a geração de uma única assinatura por par de chaves. Portanto, as OTS não são úteis na prática. No entanto, elas são utilizadas para a construção de esquemas MTS que permitem a assinatura de um número maior de mensagens.

A figura 1 mostra como é gerada uma assinatura no esquema LD-OTS, a mais simples OTS. Na figura 1, n representa o tamanho do resumo criptográfico produzido pela função *hash* f em bits, SK é a chave privada, PK a chave pública e b_i representa

 © 2017 Sociedade Brasileira de Computação

o i-ésimo bit do resumo criptográfico da mensagem a ser assinada para $0 \leq i < n$. sk_i é uma sequência de bits de tamanho n selecionada aleatoriamente. Obtém-se PK aplicando-se f sobre cada sequência de SK. A assinatura é gerada através da seleção da sequência $sk_{i,0}$ de SK caso o i-ésimo bit de b seja 0 ou $sk_{i,1}$ caso seja 1. Como é possível perceber, um esquema OTS revela parte de sua chave privada, portanto, uma segunda assinatura poderia revelar a um adversário toda a SK. É importante salientar que o nível de segurança das HBS apresentadas é definido como $n/2$ [Buchmann et al. 2011].

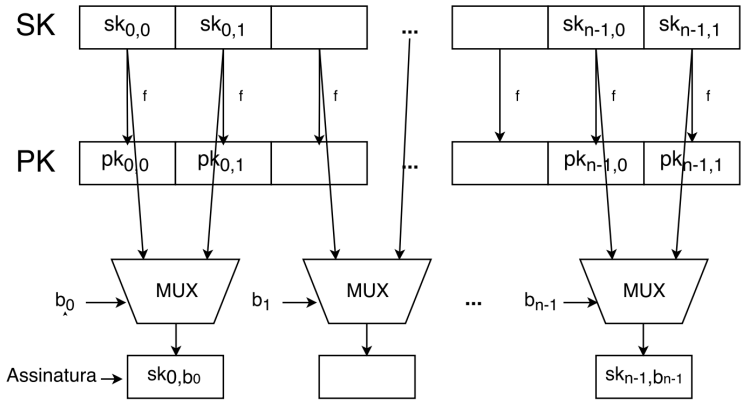


Figura 1. Geração de assinatura em LD-OTS. Fonte: PQCRYPTO17[2]

Para a verificação de uma assinatura LD-OTS, um processo similar ao de sua geração é feito. No entanto, para cada bit da mensagem a ser verificada, obtém-se a respectiva sequência (0 ou 1) de PK ao invés de SK. Então, f é calculado para todas as sequências sk_i, n da assinatura e esse resultado deve ser comparado ao do passo anterior. Como $f(SK)$ = PK, os dois resultados serão iguais se a assinatura for válida para a mensagem correspondente.

2.2. W-OTS

É possível notar que as chaves de LD-OTS possuem um tamanho consideravelmente grande ($2n^2$). Isso ocorre pois cada bit da mensagem é assinado individualmente. O esquema W-OTS diminui o tamanho da chave permitindo a assinatura de mais bits simultaneamente. O parâmetro w define o número de bits assinados em conjunto. Essa melhoria pode ser feita às custas de um aumento no número de avaliações da função f, isto é, no tempo de processamento. A partir de w e n outros parâmetros são definidos:

$$t_1 = \left\lceil \frac{n}{w} \right\rceil, t_2 = \left\lfloor \frac{\lceil log_2 t_1 \rceil + 1 + w}{w} \right\rfloor, t = t_1 + t_2 \quad (1)$$

A chave privada SK é composta por t sequências de bits de tamanho n. A chave pública PK é gerada aplicando-se f $2^w - 1$ vezes à SK. Para a geração da assinatura é calculado o resumo criptográfico da mensagem $f(M) = B = (b_0, b_1, ..., b_{n-1})$. B é então dividido em t_1 sequências de tamanho w tal que $B = b_{t-1}||...||b_{t-t_1}$, onde $||$ denota concatenação.

[2] https://huelsing.files.wordpress.com/2017/06/20170619_pq_summer_school.pdf

 © 2017 Sociedade Brasileira de Computação

A seguir, cada b_i é considerado como um inteiro de valor entre 0 e $2^w - 1$ e um *checksum* C é calculado.

$$C = \sum_{i=t-t_1}^{t-1} (2^w - b_i) \tag{2}$$

Em seguida, C é dividido em t_2 sequências de bits de tamanho w tal que $C = b_{t_2-1}||...||b_0$. Caso o tamanho de B e C não seja divisível por w, zeros são acrescentados no início da sequência. Por fim, a assinatura SIG é calculada utilizando $B||C$:

$$SIG = (f^{b_{t-1}}(sk_{t-1}), ..., f^{b_1}(sk_1), f^{b_0}(sk_0)) \tag{3}$$

A verificação da assinatura é feita calculando-se o $checksum$ novamente conforme descrito anteriormente e comparando o resumo da assinatura calculado abaixo com a chave pública.

$$(f^{2^w-1-b_{t-1}}(sig_{t-1}), ..., f^{2^w-1-b_0}(sig_0)) = (pk_{t-1}, ..., pk_0) \tag{4}$$

2.3. MSS

O esquema MSS utiliza uma árvore binária de *hashes* (árvore de Merkle) para assinar até 2^h mensagens onde h é a altura da árvore. A figura 2 mostra uma árvore que permite a assinatura de até 8 mensagens. São geradas 2^h chaves OTS e o *hash* das chaves públicas OTS $PK_0, ..., PK_{2^h}$ são armazenados nas folhas da árvore. Cada nó acima das folhas é construído gerando-se o *hash* da concatenação de seu filho esquerdo com o direito. A chave pública de um esquema MSS é a raiz da árvore. Portanto, para validar uma assinatura, o verificador deve receber, além da chave pública OTS, o índice da chave utilizada e todos os nós da árvore que permitam o cálculo da raiz. Uma assinatura MSS é válida se a verificação da chave OTS resultar em sucesso e a reconstrução da árvore gerar a mesma raiz recebida.

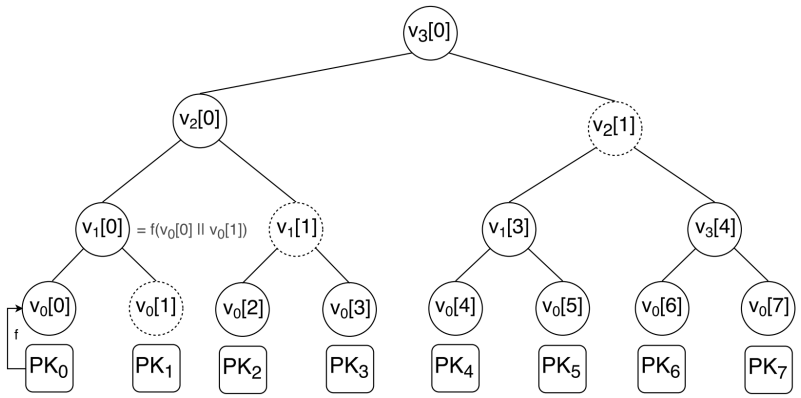


Figura 2. MSS h=3. Os nós de bordas tracejadas são usados no cálculo da raiz em uma assinatura com $v_0[0]$.

3. Implementação

A implementação dos esquemas apresentados (W-OTS e MSS) foi feita em C com a biblioteca criptográfica RELIC[3]. Os esquemas foram implementados seguindo as especificações contidas em [Buchmann et al. 2009] e descritos na seção 2 deste trabalho.

[3] `https://github.com/relic-toolkit`

 © 2017 Sociedade Brasileira de Computação

Para a geração das chaves foi utilizado um pseudo-gerador de números aleatórios implementado pela RELIC. Desta forma, as chaves do esquema MSS não precisariam ser armazenadas, apenas a semente do gerador seria suficiente para gerar as chaves à medida que fossem utilizadas.

A implementação foi feita de maneira a permitir que a função $hash$ utilizada seja trocada facilmente. O trecho abaixo mostra uma função genérica para calcular o $hash$ de uma mensagem.

```
1  void hash_msg(uint8_t out[MD_LEN], uint8_t *msg, size_t size)
2  {
3  #if MD_MAP == SHONE
4    md_map_shone(out,msg,size);
5  #elif MD_MAP == SH224
6    md_map_sh224(out,msg,size);
7  #elif MD_MAP == SH256
8    md_map_sh256(out,msg,size);
9  #elif MD_MAP == SH384
10   md_map_sh384(out,msg,size);
11 #elif MD_MAP == SH512
12   md_map_sh512(out,msg,size);
13 #endif
14 }
```

4. Avaliação e Resultados Experimentais

4.1. Avaliações Analíticas

As avaliações analíticas foram feitas de acordo com as definições em [Buchmann et al. 2009]. A tabela 1 apresenta como se dá o compromisso entre memória e processamento no esquema W-OTS de acordo com a escolha do parâmetro w. O tamanho da chave diminui linearmente enquanto o número de avaliações da função *hash* aumenta exponencialmente. Portanto, é preciso encontrar o compromisso que satisfaça às limitações do dispositivo.

 © 2017 Sociedade Brasileira de Computação

Tabela 1. Compromisso entre tamanho da assinatura e chave e o tempo de processamento

n	w	**Tamanho assinatura/ chave (bytes)**	**Avaliações de f**
256	2	4256	399
	4	2144	1005
	8	1088	8670
	16	576	1179630
512	2	16768	786
	4	8384	1965
	8	4224	16830
	16	2176	2228190

A tabela 2 apresenta a avaliação analítica do esquema MSS considerando $w = 4$. O tamanho da assinatura muda em relação à tabela 1 pois os nós da árvore de Merkle para a validação da chave pública também devem ser considerados.

Tabela 2. Avaliação do esquema MSS para w = 4

h	**n**	**Tam. chave (bytes)**	**Tam. assinatura (bytes)**	**Máximo de assinaturas**
14	256	2144	2,592	16,384
	384	4752	5,424	
	512	8384	9,280	
16	256	2,144	2,656	65,536
	384	4752	5,520	
	512	8384	9,408	
18	256	2,144	2,720	262,144
	384	4752	5,616	
	512	8384	9,536	

4.2. Avaliações Experimentais

As avaliações experimentais foram feitas na plataforma Arduino Due que possui processador ATSAM3X8E 84 MHz, 96kB de SRAM e 512kB de memória Flash. Os experimentos foram executados 100 vezes para cada operação (geração de chaves, assinatura da mensagem e verificação da mensagem). Os gráficos 3(a) e 3(b) apresentam os tempos por operação utilizando as funções SHA256, SHA384 e SHA512 implementadas na biblioteca RELIC. Os valores escolhidos para w foram 2 e 4 pois estes possibilitam um tamanho de chave aceitável para o Arduino Due sem que os tempos de execução tornem-se impraticáveis.

 © 2017 Sociedade Brasileira de Computação

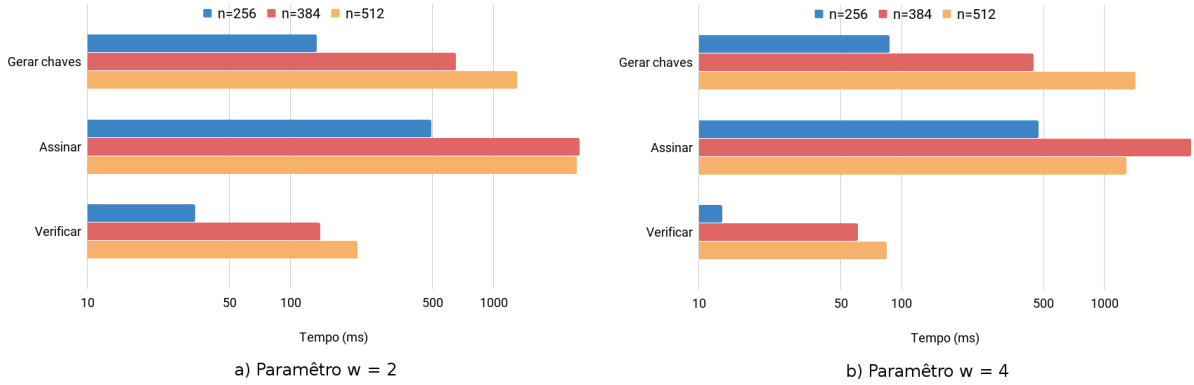


Figura 3. Tempo por operação em W-OTS

O gráfico 4 apresenta o tempo por operação para o MSS usando o W-OTS. O maior custo do MSS é a geração de chaves pois é preciso gerar todas as chaves OTS para calcular a PK. No entanto, para valores de h pequenos é possível gerá-las no próprio dispositivo. As operações de geração e verificação da assinatura têm como a maior parte de seu custo ligados ao esquema OTS. Isso decorre do fato de que gerar e validar o caminho de autenticação é barato pois apenas h nós são utilizados.

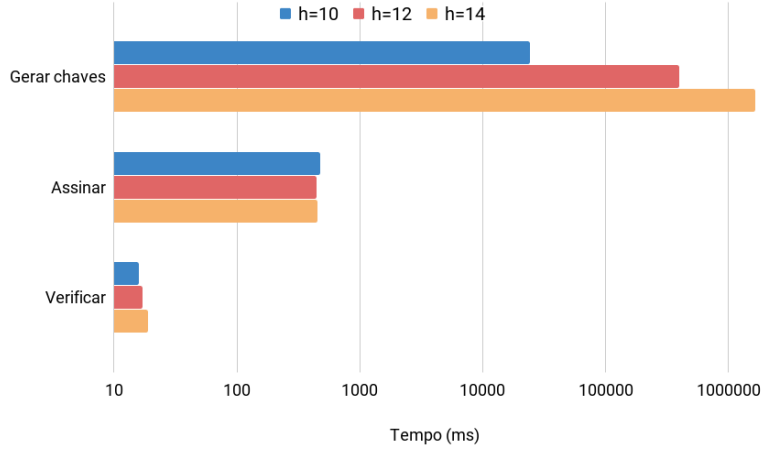


Figura 4. Tempo por operação em MSS

5. Trabalhos Relacionados

O trabalho de [Barreto et al. 2014] apresenta um panorama da criptografia pós-quântica. Exitem trabalhos que implementam esquemas pós-quânticos citados na seção 1 para dispositivos restritos. Em [Maia et al. 2010] é apresentada uma implementação de um criptossistema baseado em sistemas quadráticos multivariados para redes de sensores sem fio. No trabalho de [Biasi et al. 2014] é feita a implementação de um criptossistema baseado em códigos corretores de erro.

Na literatura, existem também trabalhos que implementam e avaliam HBS em dispositivos limitados como *smart cards* com microprocessadores 8-bit [Rohde et al. 2008] e

 © 2017 Sociedade Brasileira de Computação

16-bit [Hülsing et al. 2012], microprocessadores AVR ATxmega [Eisenbarth et al. 2013] e microcontroladores ARM Cortex M3 [Hülsing et al. 2016]. No entanto, ainda existem poucos trabalhos tratando deste assunto.

Em [Rohde et al. 2008], o esquema MSS com o W-OTS foram implementados em *smart cards* com microprocessadores 8-bit da família AT90SCxxx utilizando a cifra de bloco AES. Os autores mostraram que sua implementação para *smart cards* possui código menor se comparado a implementações dos esquemas RSA e ECDSA para a mesma plataforma, além de ser mais eficiente que o RSA e comparável ao ECDSA. No entanto, neste trabalho as chaves não são geradas diretamente no dispositivo devido às suas limitações.

O trabalho de [Hülsing et al. 2012] apresenta uma versão baseada no esquema XMSS à qual dão o nome de XMSS$^+$ e sua implementação para *smart cards* 16-bit Infineon SLE78. Neste trabalho, o problema de geração das chaves no próprio dispositivo foi solucionado através da redução do tempo de geração das chaves em XMSS$^+$.

No trabalho de [Eisenbarth et al. 2013], uma nova abordagem para a computação do caminho de autenticação para o MSS foi sugerida para evitar aumentar a resiliência ao vazamento das chaves sob o custo de maior armazenamento necessário. A implementação desta nova abordagem para o MSS foi feita para microprocessadores AVR ATxmega.

Em [Hülsing et al. 2016], os autores implementaram o esquema SPHINCS em um microcontrolador ARM Cortex M3. Este esquema não armazena estados, isto é, ao contrário do esquema avaliado neste trabalho, o SPHINCS não lembrar-se da última chave utilizada. A desvantagem de esquemas *stateless* é que eles possuem chaves significantemente maiores que os esquemas *stateful*. No entanto, os autores demonstraram ser possível implementar um esquema *stateless* em um dispositivo limitado.

6. Conclusão

Este trabalho mostrou a aplicabilidade das HBS em dispositivos IoT como uma alternativa aos esquemas tradicionais no período pós-quântico. As HBS devem ser consideradas como possíveis substitutas do RSA, DSA, ECDSA e outros. O bom entendimento do nível de segurança e facilidade de implementação das HBS tornam esses criptossistemas vantajosos em relação a outros esquemas pós-quânticos.

Para trabalhos futuros, serão implementados outros esquemas HBS para avaliação e comparação com os já implementados. Além disso, outras métricas podem ser utilizadas nas avaliações tais como consumo de energia, tamanho de código e número de ciclos.

Agradecimentos

Esse trabalho foi parcialmente financiado pelo CNPq, FAPEMIG, CAPES e RNP.

Referências

Aranha, D. F. and Gouvêa, C. P. L. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

Barreto, P. S., Biasi, F. P., Dahab, R., López-Hernández, J. C., de Morais, E. M., de Oliveira, A. D. S., Pereira, G. C., and Ricardini, J. E. (2014). A panorama of post-quantum

 © 2017 Sociedade Brasileira de Computação

cryptography. In *Open Problems in Mathematics and Computational Science*, pages 387–439. Springer.

Biasi, F. P., Barreto, P. S., Misoczki, R., and Ruggiero, W. V. (2014). Scaling efficient code-based cryptosystems for embedded platforms. *Journal of Cryptographic Engineering*, 4(2):123–134.

Buchmann, J., Dahmen, E., and Szydlo, M. (2009). Hash-based digital signature schemes. In *Post-Quantum Cryptography*, pages 35–93. Springer.

Buchmann, J. A., Dahmen, E., Ereth, S., Hülsing, A., and Rückert, M. (2011). On the security of the winternitz one-time signature scheme. *Africacrypt*, 11:363–378.

Dods, C., Smart, N. P., and Stam, M. (2005). Hash based digital signature schemes. In *IMA International Conference on Cryptography and Coding*, pages 96–115. Springer.

Eisenbarth, T., Von Maurich, I., and Ye, X. (2013). Faster hash-based signatures with bounded leakage. In *Selected Areas in Cryptography*, pages 223–243.

ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472.

Hülsing, A., Busold, C., and Buchmann, J. A. (2012). Forward secure signatures on smart cards. In *Selected Areas in Cryptography*, volume 7707, pages 66–80. Springer.

Hülsing, A., Gazdag, S.-L., Butin, D., and Buchmann, J. (2015). Hash-based signatures: An outline for a new standard.

Hülsing, A., Rijneveld, J., and Schwabe, P. (2016). Armed sphincs: Computing a 41 kb signature in 16 kb of ram.

Johnson, D., Menezes, A., and Vanstone, S. (2001). The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63.

Maia, R. J. M., Barreto, P. S. L. M., and de Oliveira, B. T. (2010). Implementation of multivariate quadratic quasigroup for wireless sensor network. In *Transactions on computational science XI*, pages 64–78. Springer.

Meier, A. V. (2005). The elgamal cryptosystem.

Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC press.

Merkle, R. C. (1989). A certified digital signature. In *Proceedings on Advances in cryptology*, pages 218–238. Springer-Verlag New York, Inc.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Rohde, S., Eisenbarth, T., Dahmen, E., Buchmann, J., and Paar, C. (2008). Fast hash-based signatures on constrained devices. In *International Conference on Smart Card Research and Advanced Applications*, pages 104–117. Springer.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE.

 © 2017 Sociedade Brasileira de Computação

Benchmarking tools for verification of constant-time execution

Arthur Costa Lopes[1], **Diego de Freitas Aranha**[1]

[1]Institute of Computing – University of Campinas (UNICAMP)

Abstract. *Developing secure implementations of cryptography, in particular those protected against side-channel attacks, is a challenging research and engineering problem. In the case of timing attacks, the task can be facilitated by employing verification tools to check constant-time behavior. Given this concern, this work explores different verification tools and their distinct forms of analysis with application to verifying cryptographic libraries. A simple syntax for describing implementations was designed and a benchmarking database was constructed to validate such tools, indicating that the combination of static and dynamic analysis is required to fully verify the timing behavior of a cryptographic implementation.*

Resumo. *Desenvolver implementações seguras de criptografia, em particular protegidas contra ataques de canal lateral, é um problema desafiador de pesquisa e engenharia. No caso de ataques de temporização, a tarefa pode ser facilitada ao se empregar ferramentas de verificação para determinar execução em tempo constante. Dado esse problema, este trabalho expora diferentes ferramentas de verificação e suas formas distintas de análise, com aplicação em bibliotecas criptográficas. Uma sintaxe simples para descrever as implementações foi projetada e uma base de implementações foi construída para comparar e validar tais ferramentas, indicando que uma combinação de análise dinâmica e estática é requisito para verificar completamente o comportamento de uma implementação criptográfica do ponto de vista de tempo de execução.*

1. Introduction

With a high demand for computational systems to store, transfer and process sensitive data, the Information Security community is responsible for designing and deploying methods that must remain secure even when widely used. The application of cryptographic techniques is a common approach to protect part of the attack surface of modern systems, but naturally even cryptography may suffer from flaws and vulnerabilities. Given this goal, there is a substantial amount of research dedicated to explore different ways to attack such systems, retrieve sensitive data and expose cryptographic keys.

Side-channel attacks became popular after their success in retrieving secret information protected by a cryptosystem, using methods that attack implementations of algorithms instead of tackling the more challenging underlying hardness assumptions such as integer factoring, discrete logarithm computation or exhaustive search in the key space [Kocher 1996]. There are many types of side-channel attacks, involving execution time, branch prediction rate, power consumption and acoustic emanations. The attacks can be passive and rely on monitoring of side-channel information only, or more invasive

 © 2017 Sociedade Brasileira de Computação

and require injection of faults in the processing hardware. Passive attacks that work remotely against connected devices are among the most convenient for an attacker to mount.

In the past years, many timing attacks against cryptographic implementations came up in the literature, compromising entire cryptosystems and exposing sensitive data by using timing characteristics leaked by an insecure implementation [Bernstein 2004, Percival 2005, Aciiçmez et al. 2007]. Those attacks consist in measuring the execution/response time of an insecure implementation, which will be different for distinct inputs and may be correlated with secret information. Critical variances in execution time can be caused by careless programming practices, but also by more subtle cache-timing behavior, branch prediction/resolution effects and variable-time instructions within the processor microarchitecture. If a system is secure against timing attacks, it is called *constant time* because it produces the answer with the same latency independently of the inputs. On the other hand, if the running time is not constant, meaning that different inputs yield different running times, this system is called time-variant and may pose a vulnerability that can be exploited by a remote attacker.

Timing attacks can be extremely complex to exploit and require the combination of sophisticated techniques and observation of network traffic, as discussed above, but they can also be very simple. For example, suppose an algorithm that checks if a given string matches a certain secret password or its hash value, as shown below in Listing 1. It can be easily implemented by comparing each character from both strings and stopping the comparison immediately if a difference is found, or returning that the password is correct contrariwise. Now, given this specific implementation, consider an attack in which the opponent sends a random word, but with the first letter being *s* and measures the response time. If the first letter of the password is in fact *s*, then the execution time would be greater than the case in which the first letter was wrong, since the code will try to match the second letter. Now the attacker has a simple way to check each letter separately, reducing the complexity of the brute force attack from exponential to linear.

```
1  /*If pw == in returns 1, else returns 0 */
2  int compVar (char *pw, char *in, int len) {
3     for (int i = 0; i < len; i++) {
4        if (pw[i] != in[i]) {
5           return 0;
6        }
7     }
8     return 1;
9  }
10
11 int main(void) {
12    int result, comp=1;
13    char pw[] = "secret", in[7];
14    scanf ("%s", in);
15    result = compVar(pw, in);
16    printf ("%d", result);
17 }
```

Listing 1. Example of a time variant implementation of string comparison.

 © 2017 Sociedade Brasileira de Computação

A constant-time version of the same string comparison function with the same interface can be found below in Listing 2.

```
1 /*If pw == in returns 1, else returns 0 */
2 int compConst (char *pw, char *in, int len) {
3     int r = 0;
4     for (int i = 0; i < len; i++) {
5         r |= (pw[i] ^ in[i]);
6     }
7     return (r == 0);
8 }
```

Listing 2. Example of a constant time implementation of string comparison.

Although constant time implementations appear to solve all the timing attack vulnerabilities, there are a few more steps that need to be taken in order to assure such protection. First the programmer must check if the high-level code executes in constant time across all code paths, possibly with the help of static analysis. Even though static analysis tools may check the code and rate it as secure, the programmer still must use other tools to verify dynamic behavior. The usage of both methods is required because in the transition from source code (i.e. C/C++ source file) to a final executable there are many processing steps performed by the compiler, which can generate a final version with side-channel protection disabled during the optimization or containing insecure variable-time instructions inserted during code generation.

This work presents two main contributions. The first one is CTBench, a large database containing several implementations of cryptographic functions, ready to be tested by any static or dynamic analysis tools. We evaluated the tools FlowTracker, *dudect* and *ctgrind* in terms of accuracy when analyzing if the implementations are constant time or time-variant. Another contribution was the design of a XML file format useful to annotate critical information in cryptographic code for static analysis.

This paper is organized as follows. Section 2 discusses the two analysis techniques, as well as a few examples of tools that are available in the research literature. Section 3 presents the representation format and how the benchmarking database is organized. Section 4 discusses the results obtained by the tools. Section 5 concludes the paper and discusses future work.

2. Related work

Previous attacks against implementations of AES were able to recover the full key almost in real time just by observing encryption of a few KBs of data [Gullasch et al. 2011], or by analyzing network response time [Bernstein 2004]. Others were able to recover Diffie-Hellman exponents in naive implementations of the square-and-multiply exponentiation operation [Kocher 1996] and even factor RSA keys [Schindler 2000] by measuring precisely the time required to perform some operations like encryption and multiplications, sometimes remotely [Brumley and Boneh 2005].

A more specific class of timing attacks is called cache-based attacks, which uses leakage from hits and misses in cache memory to extract information about a secret parameter (i.e. cryptographic keys or plaintext data). Cache memory provides faster

 © 2017 Sociedade Brasileira de Computação

memory access to the processor for small amounts of data which are frequently used, reducing the delay to retrieve these values from the main memory. Since cache memory typically only has up to 128 KBs of capacity in its fastest L1 level, sometimes a process may not find a memory address loaded on it, and therefore the processor must retrieve the data from slower cache levels or the main memory, yielding a cache miss and consequently a delay. The attack can also work across different processing cores if there is a shared cache level. Measuring this kind of delay during the encryption or decryption of a given implementation can allow the attacker to infer properties of the key being used. This type of attack works against insecure implementations of AES, as shown by [Bernstein 2004, Percival 2005]; and the RSA cryptosystem [Yarom and Falkner 2014, Yarom et al. 2017].

There are several works in the literature that detected and exploited implementations of cryptographic algorithms using variable-time instructions or compiler interference. For example, integer multiplication instructions in ARM processors that finish early when the precision of the operands is low can be explored in side-channel attacks [Großschädl et al. 2009]. Another work targeting this approach is [Kaufmann et al. 2016], which studied attacks against the Curve25519 key exchange protocol by analyzing the 32-bit compiled version of an otherwise 64-bit secure implementation. This happened due to the Windows runtime library which added an insecure branch in order to optimize long integer multiplication. There are some microbenchmarks specifically created for different architectures to clarify programmers which instructions are safe and, more importantly, which are not secure and should be avoided or used with some additional precautions [Pornin 2017].

Given the power of timing attacks in the literature and practice of cryptography, the importance of studying how to prevent vulnerabilities and verify the security of cryptographic implementations in terms of side-channel resistance is clear. Many approaches tried to prevent such vulnerabilities by creating tools to be used by the programmer to improve the code for critical applications. The main types of analysis employed by these tools are *dynamic* and *static*, with their own features and limitations, as discussed in detail on the sections below.

2.1. Dynamic Analysis

A first method to detect if an implementation leaks timing information is to analyze its behavior while executing given some input. This kind of method is called dynamic since it relies on an experiment using the compiled version of the code during execution, and not just the source code itself.

A recently proposed tool called *dudect* [Reparaz et al. 2017] was created using the concept. It tests an implementation against several inputs, trying to find a different behaviour that yields a variant running time. In order to detect such behaviour, it uses some statistical tests, such as the Welch's t-test for detecting variations in the collected samples. An advantage of dynamic analysis is detecting low-level timing variances introduced by the instructions themselves, such as early-abort multipliers available in the ARM architecture[Großschädl et al. 2009]; or by the compiler through insecure runtime libraries [Kaufmann et al. 2016].

Another tool for this kind of analysis is *ctgrind* [Langley 2010], based on the pop-

 © 2017 Sociedade Brasileira de Computação

ular *Valgrind* tool [Nethercote and Seward 2007] for memory management and leakage analysis. The *ctgrind* tool changes the behavior of *Valgrind* to check if secret data was accessed and then concludes that a branch instruction depends on secret information, ultimately marking the implementation as insecure. The modified behaviour can be used through special functions provided by *ctgrind* to treat secret parameters as uninitialized data, triggering the warnings.

The main limitation of these tools is that they cannot check if a given implementation is in fact constant-time, because this would imply that every possible input was tested and that the average execution time of every sample has the same value, which is impossible given the number of possibilities. Furthermore, unlikely inputs carefully crafted by an attacker to produce variable time behavior may not be explored in a small number of samples. However, they can still provide some supporting evidence of timing behavior and help the programmer fix his/her implementation.

Each tool has a different method to be used which can vary according to the functions executed by the program. Both dynamic tools studied use a similar representation process to test a given implementation. The *dudect* tool requires a C source file with a function to generate the random input and calculate the execution time of a sample, and a wrapper for each function the user wants to check. The user must just initialize any required variables and use the randomly generated data as input. Another parameter that can be changed is the number of samples taken, usually ranging from then thousand to one million. The tool will output two possible results:

- *For the moment, maybe constant time*, which means that so far every sample taken has a measurement very close to each other, so it cannot confirm that the function is indeed variant or constant time.
- *Probably not constant time*, which means that the mean from two or more samples have enough difference to indicate that the function is not constant-time.

The *ctgrind* tool requires a C source file containing the function to be tested and the special initialization of the secret data. To compile the code, the user must also use some special flags provided in the Makefile in order to be able to use the *Valgrind* software to check the binary.

2.2. Static Analysis

Another type of analysis, known as static analysis, is widely applied to find vulnerabilities involving sensitive variables used in a cryptosystem. This kind of analysis receives the moniker static because it works based on an intermediate representation of the code, instead of the execution of a compiled binary.

Static analysis tools can detect several types of problems within an implementation without compiling and executing the software itself. To perform this kind of analysis the user simply executes the tool using an auxiliary file containing information about the code that he wants to check and the source file. The user will usually receive several warnings describing how the secret information flows within the application. Some recent tools like FlowTracker[Rodrigues et al. 2016] and *ct-verif* [Almeida et al. 2016] use techniques for analyzing programs in intermediate representation languages.

The *ct-verif* tool requires the source file to be annotated using a special syntax and converts a compiled program represented in the LLVM intermediate representation to a

 © 2017 Sociedade Brasileira de Computação

special representation in the Boogie language which allows to extract provable guarantees of constant-time behavior. This allows the user to detect branch instructions using secret information as one of the parameters, therefore being vulnerable since the running time will depend on those parameters. Unfortunately, the tool appears rather preliminary for general use and under development, so it was not possible to benchmark it in this work.

FlowTracker [Rodrigues et al. 2016] is a static tool developed to detect and analyze the implicit flow of information within an implementation. It can be used to keep track of sensitive input, such as cryptography keys, and determine if its value may have some influence in the running time, for example by being the parameter for a branch instruction or the index of a vector possibly stored in cache memory. The main limitation of these tools is that it cannot check all dependencies from a given code, therefore vulnerabilities will not be found if insecure functions or instructions are used.

FlowTracker has a distinct kind of input since it will not compile and execute the code. As explained in Section 3 the input consists of an XML file with precise information about the functions being tested. The output is a little bit more involved. There are two file formats generated by the tool.

- *Subgraph file*: Indicates the nodes in which secret information was propagated, from the declaration to a branch.
- *ASCII file*: indicates the lines in the code that propagate sensitive information to a branch or array index. These can be used by the programmer to detect and fix the vulnerability.

Table 1 below shows the main advantages and limitations of each tool.

Table 1. Comparison of different tools for analyzing constant-time behavior.

Tool name	Type	Limitations	Advantages
dudect	Dynamic	Cannot prove that an implementation is constant time. Takes a long time to run for covering a large portion of inputs.	Usability and detection of microarchitecture effects.
ctgrind	Dynamic	Cannot prove that an implementation is constant time or test all possible inputs.	Usability (reasonably easy to set up and run).
ct-verif	Static	Preliminary and under development.	Provides formal constant-time guarantees at high level, but does not detect microarchitecture effects.
FlowTracker	Static	Cannot check all dependencies automatically or detect microarchitecture effects.	High efficiency and coverage of all possible inputs through evidence collected from information flow analysis.

 © 2017 Sociedade Brasileira de Computação

3. Benchmarking database

In this section, we describe how our benchmarking database was constructed, by starting from the annotation process. Because each dynamic or static tool has specific characteristics, they also require a representation file containing a compilation of the main properties that the user must give in order to perform the analysis. Given that most of information from these files is the same and can be shared among different tools, such as name of the function, parameters and some others, we decided to create a standard XML file to be used by different tools. With this general file, the user can easily analyze the code being written by various tools, expanding the verification and therefore being able to find a larger number of vulnerabilities. Another advantage of this representation is that the user can create this file once and use several times in different tools by just changing a few tags, facilitating the automation of the verification process and detection of potential regressions due to changes in the code or compiling toolchain.

The XML created to describe an implementations consists of a main tag representing each function to be tested (tag `function`), including the name and all parameters, separated in two sections: `public` and `secret`. Also, the user can use a special tag `return` to determine if the return from a given function must be treated as a secret by assigning the tag to *true*. Just like any XML file, the user can use comments to specify the meaning of each parameter received by the function. For illustration purposes, the XML file below refers to the representation of Algorithm 1 presented in Section 1.

The FlowTracker static analysis tool was already to designed to receive an input XML with annotations in a similar format, although less usable. Some illustrative examples can be found in the project website[1]. The FlowTracker format required the programmer to refer to sensitive parameters only by using their number in the prototype list of arguments. We performed all the necessary modifications to adapt FlowTracker parsing in order to integrate the new representation file with the tool. The entire benchmarking database is publicly available on GitHub[2].

```
<functions>
   <sources>

      <function>
         <name>compVar</name> <!--Function to be analyzed-->
         <return>false</return> <!--Return value is not critical-->
         <public>
            <parameter>in</parameter> <!--Input String-->
         </public>
         <secret>
            <parameter>pw</parameter> <!--Password-->
         </secret>
      </function>

   </sources>
</functions>
```

Listing 3. XML file used to annotate code in Listing 1.

[1] `http://cuda.dcc.ufmg.br/flowtracker/example.html`
[2] CTBench - `https://github.com/arthurlopes/ctbench`

 © 2017 Sociedade Brasileira de Computação

Table 2. List of algorithms in our benchmarking database, containing implementations from the BearSSL and NaCl cryptographic libraries and examples from the *dudect* dynamic analysis tool.

Library	Algorithm Category	Constant	Variant
BearSSL	Symmetric	18	4
	MAC	1	2
	Hash	3	5
	RSA	3	4
	ECC	0	4
dudect examples	AES	1	1
	ECC	1	1
	Others	1	1
NaCl	Authenticated encryption	6	0
	Hash	4	0
	Curve25519	1	0

Another contribution proposed by this paper was the creation of a benchmarking database containing several cryptograpihc functions used to analyze the tools. To create such a database we used implementations from the cryptographic libraries BearSSL[3], NaCl[4] [Bernstein et al. 2012] and examples provided by the *dudect* tool. In total the database has about 60 implementations detailed in Table 2 below.

More precisely, the database contains three libraries, the first one is the *dudect* examples, consisting of two AES implementations, two comparison algorithms and two Elliptic Curve Cryptography (ECC) implementations. The second one, BearSSL, contains about twenty implementations of the AES and four of the DES block ciphers. Regarding hash functions, there are implementations of both MD5 and SHA. There are also three implementations of Message Authentication Codes (MAC) and a few of ECC algorithms. The last library is NaCl, containing six algorithms related to Authenticated encryption, two hash implementations (SHA256 and SHA512) and one implementation of scalar multiplication in a specific prime curve (Curve25519) [Bernstein 2006].

The groundtruth classification in constant or variable-time implementation was obtained through the official documentation of each project. To complete the database, all representation files used by the FlowTracker, as well as every C source file required by *dudect* were created in order to run the tools correctly.

4. Experimental results

The results were obtained by evaluating the static and dynamic verification tools across our entire benchmarking database.

We started by analyzing the database and rating each implementation as constant or variable time with the static tool FlowTracker. After the static analysis was finished we

[3]BearSSL - A smaller SSL/TLS library - `https://www.bearssl.org`

[4]NaCl: Networking and Cryptography library - `https://nacl.cr.yp.to`

 © 2017 Sociedade Brasileira de Computação

moved to the dynamic approach.

Next, the *dudect* tool was used to check all implementations classified as constant time by FlowTracker, which as expected produced the output *probably constant-time implementation*, since every sample taken had the same mean, producing an inconclusive result. To obtain a reliable result we executed most implementation about ten million times. For some implementations that required more time to run this value was reduced to five million or increased to twenty million, always running for at least fifteen minutes.

Only one implementation showed different behaviour due to an insecure function used (*memcmp* with early-abort comparison). FlowTracker classified this implementation as constant because it did not analyze the code for this function, but fortunately *dudect* detected the vulnerability. This implementation is a comparison function from the *dudect* examples, the variant *cmpmemcmp* using `memcmp()` implemented in variable time.

After using the *dudect* to check all implementations, we executed the same tests with the *ctgrind* tool. The original *ctgrind* patch was adapted to the latest release 3.13 of Valgrind since the GitHub version was created in 2010 and was not compatible with recent versions. The tools produced equivalent results, an expected outcome given that both use similar analysis (dynamic).

Finally we studied the behavior of implementations classified as variable time by FlowTracker. Both dynamic tools were able to find vulnerabilities in every implementation. For those implementations, FlowTracker was able to provide the corresponding subgraph indicating where the secret information is being used, along with the affected branch instructions. The tools *dudect* was able to find samples with different means and *ctgrind* found traces of uninitialized memory being used. Considering all those results we can conclude that the code indeed does not run in constant time regarding the input and the tools behave as intended.

5. Conclusions and future work

Given the amount of code analyzed from the libraries, those tools can indeed be used by a programmer to improve the code and mitigate timing attacks. The usage of those tools is a recommended practice that can be easily applied by developers to help them create more secure implementations, avoiding the leakage of sensitive and crucial data. We observed that a combination of static and dynamic analysis must be used to fully characterize the timing behavior of a cryptographic implementation, since the binary compiled version can use insecure functions or instructions that the static analysis is not able to detect.

We plan to extend the benchmarking database to include more libraries and examples to be used by several tools, analyzing their performance, effectiveness and the correctness. With a large database we will able to find more examples of implementations that result in different results when using dynamic and static analysis.

Additionally, we will study how those analysis tools could be integrated to commonly used development tools, in order to facilitate the usage by developers who are not fully aware about side-channels attacks work against insecure implementations. In terms of usability, our representation format will be extended to include information about external dependencies such that static analysis tools can produce a warning in case a specific portion of code is not available for analysis.

 © 2017 Sociedade Brasileira de Computação

6. Acknowledgements

We thank Intel and FAPESP for financial support though process 14/50704-7.

References

Aciiçmez, O., Koç, Ç. K., and Seifert, J. (2007). Predicting secret keys via branch prediction. In *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 225–242. Springer.

Almeida, J. B., Barbosa, M., Barthe, G., Dupressoir, F., and Emmi, M. (2016). Verifying constant-time implementations. In *USENIX Security Symposium*, pages 53–70. USENIX Association.

Bernstein, D. J. (2004). Cache-timing attacks on AES. URL: `http://cr.yp.to/papers.html#cachetiming`.

Bernstein, D. J. (2006). Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer.

Bernstein, D. J., Lange, T., and Schwabe, P. (2012). The security impact of a new cryptographic library. In *LATINCRYPT*, volume 7533 of *Lecture Notes in Computer Science*, pages 159–176. Springer.

Brumley, D. and Boneh, D. (2005). Remote timing attacks are practical. *Computer Networks*, 48(5):701–716.

Großschädl, J., Oswald, E., Page, D., and Tunstall, M. (2009). Side-channel analysis of cryptographic software via early-terminating multiplications. In *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 176–192. Springer.

Gullasch, D., Bangerter, E., and Krenn, S. (2011). Cache games - bringing access-based cache attacks on AES to practice. In *IEEE Symposium on Security and Privacy*, pages 490–505. IEEE Computer Society.

Kaufmann, T., Pelletier, H., Vaudenay, S., and Villegas, K. (2016). When constant-time source yields variable-time binary: Exploiting curve25519-donna built with MSVC 2015. In *CANS*, volume 10052 of *Lecture Notes in Computer Science*, pages 573–582.

Kocher, P. C. (1996). Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer.

Langley, A. (2010). ImperialViolet: Checking that functions are constant time with Valgrind. `https://www.imperialviolet.org/2010/04/01/ctgrind.html`.

Nethercote, N. and Seward, J. (2007). Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI*, pages 89–100. ACM.

Percival, C. (2005). Cache missing for fun and profit. In *Proceedings of BSDCan*.

Pornin, T. (Accessed in July 2017). BearSSL - Constant-Time Mul. `https://www.bearssl.org/ctmul.html`.

 © 2017 Sociedade Brasileira de Computação

Reparaz, O., Balasch, J., and Verbauwhede, I. (2017). Dude, is my code constant time? In *DATE*, pages 1697–1702. IEEE.

Rodrigues, B., Pereira, F. M. Q., and Aranha, D. F. (2016). Sparse representation of implicit flows with applications to side-channel detection. In *CC*, pages 110–120. ACM.

Schindler, W. (2000). A timing attack against RSA with the chinese remainder theorem. In *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 109–124. Springer.

Yarom, Y. and Falkner, K. (2014). FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732. USENIX Association.

Yarom, Y., Genkin, D., and Heninger, N. (2017). CacheBleed: a timing attack on OpenSSL constant-time RSA. *J. Cryptographic Engineering*, 7(2):99–112.

 © 2017 Sociedade Brasileira de Computação

Estudo do mecanismo de descoberta de vizinhança do IPv6 para a realização do ataque *man in the middle*

Matheus Dias Queiroz[1], **André Leon S. Gradvohl**[1]

[1]Faculdade de Tecnologia – Universidade Estadual de Campinas (UNICAMP)
R. Paschoal Marmo, 1888. Jd. Nova Itália. Limeira-SP, Brasil.

madaqz@gmail.com, gradvohl@ft.unicamp.br

Abstract. *Due to the increasing demand for a more connected world, the need to transition from IPv4 to IPv6 protocols has become a problem. With IPv6 traffic on the networks, the security of this version of the protocol becomes a target of study, since its predecessor carries with it security flaws. One of the flaws is in its neighborhood discovery mechanism, which allows a "man in the middle" sort of attack. This kind of attack provides improper access to the information that flows on the network. This research analyzed this mechanism of IPv6 to verify if the protocol also brings with it failures that allow this type of attack. For this, we perform experiments that showed that this security flaw persists in IPv6.*

Resumo. *Devido à crescente demanda de um mundo mais conectado, a necessidade de transição do protocolo IPv4 para o protocolo IPv6 tornou-se um problema. Com o tráfego IPv6 nas redes, a segurança desta versão do protocolo torna-se alvo de estudo, já que seu antecessor carrega consigo falhas de segurança. Uma delas está em seu mecanismo de descoberta de vizinhança, que permite um ataque do tipo man in the middle. Essa modalidade de ataque provê acesso indevido às informações que trafegam na rede. Essa pesquisa analisou esse mecanismo do IPv6 para verificar se o protocolo também traz consigo falhas que permitem esse tipo de ataque. Para isso, foram realizados experimentos que mostraram que essa brecha de segurança persiste no IPv6.*

1. Introdução

A rápida expansão da Internet e seu ritmo constante de crescimento e evolução implicou em necessidades de mudanças em um dos principais protocolos responsáveis por seu funcionamento, o *Internet Protocol* (IP). O IP versão 4 (IPv4), surgiu em 1979 e está em operação até hoje. Na época de seu surgimento não era possível imaginar o quanto a Internet cresceria e a quantidade de endereços possíveis do protocolo, em torno de 4 bilhões, parecia ser mais do que suficiente.

Porém, segundo Mota Filho (2013), em 2011 a *Internet Assigned Numbers Authority* (IANA), organização responsável por atividades de controle da Internet, anunciou que o estoque de endereços IPv4 havia acabado. De acordo com Brito (2013), a falta de endereços IP pode impedir o crescimento da maior rede mundial, a Internet. Visto que o esgotamento dos endereços IPv4 é um problema em evidência, especialmente com o advento da internet das coisas, torna-se cada vez mais necessária a implementação do IPv6, que soluciona o problema de esgotamento de endereços IP.

 © 2017 Sociedade Brasileira de Computação

A segurança da informação é um problema presente no protocolo IP e atualmente esse cenário traz preocupações. Para exemplificar a importância desse problema, temos o caso do site oficial das Olimpíadas do Rio de Janeiro em 2016, que foi alvo de mais de 200 milhões de ataques durante a competição. Essa informação é da Microsoft, que foi responsável pelo sistema de gerenciamento de conteúdo do site (GUGELMIN, 2016).

No IPv4, uma das falhas de segurança conhecidas está em seu mecanismo de descoberta de vizinhos. O protocolo responsável por esse mecanismo no IPv4 é o *Address Resolution Protocol* (ARP). Ele faz a tradução do endereço IP em um endereço físico, i. e. o endereço da interface de rede de um *host*. Esse mecanismo é necessário quando um *host* na rede conhece o endereço IP de seu destino mas não conhece seu endereço físico, necessário para a comunicação.

Portanto, o ARP realiza a tarefa de descobrir o endereço físico do *host* de destino. A falha nesse mecanismo permite que um atacante envie mensagens ARP forjadas na rede, fazendo com que o endereço físico da máquina do atacante seja associado ao endereço IP do *host* de destino original. O resultado disso é o direcionamento do tráfego de dados entre dois *hosts* legítimos para a máquina do atacante, sem que as vítimas percebam. Esse ataque é chamado *man in the middle* (MORIMOTO, 2008; ULBRICH; VALLE, 2007; CONTI; DRAGONI; LESYK, 2016).

De acordo com Denis, Zena e Hayajneh (2016), o ataque *man in the middle* é essencial para ganhar controle sobre uma rede que permite, posteriormente, a utilização de outras técnicas que acessam indevidamente as informações.

No IPv6, o protocolo responsável pelo mecanismo de descoberta de vizinhos é o *Neighbor Discovery Protocol* (NDP) (NARTEN et al., 2007). Logo, a ideia desta pesquisa é realizar experimentos que explorem o NDP da mesma forma que o ARP é explorado no IPv4, em busca da realização do ataque *man in the middle*. O objetivo disso é verificar se o novo mecanismo de descoberta de vizinhos que o IPv6 implementa também apresenta falhas, semelhante ao seu antecessor.

Para descrever a pesquisa e seus resultados, este artigo está organizado da seguinte forma: a Seção 2 aborda os conceitos utilizados para o desenvolvimento do experimento e também explica a diferença entre a técnica de ataque de trabalhos semelhantes. O experimento está descrito na Seção 3, com explicações sobre a topologia de rede usada, equipamentos, ferramentas, softwares e a técnica de ataque. Os resultados do experimento são detalhados na Seção 4, que também explica os efeitos do ataque na vítima. Finalmente, a Seção 5 apresenta as conclusões e algumas observações finais.

2. Conceitos fundamentais

O IPv6 trouxe um novo mecanismo de descoberta de vizinhos em relação ao IPv4 e o protocolo responsável por esse mecanismo é o NDP. Nesta pesquisa, o NDP foi o principal alvo dos estudos pelo fato de realizar um trabalho que no IPv4, com o protocolo ARP, apresentava falhas que podiam ser exploradas para a realização do ataque do tipo *man in the middle*. O IPv6 também apresenta outros conceitos importantes, que serão abordados nas seções a seguir, além de detalhes sobre o NDP e o tipo de ataque *man in the middle*.

 © 2017 Sociedade Brasileira de Computação

2.1. Tipos de endereços IPv6

Segundo o IPv6.br (2012), existem três tipos de endereços definidos no IPv6:

- *Unicast*: são endereços que identificam uma interface específica de maneira única, de modo que um pacote enviado a um endereço *unicast* é entregue a uma única interface. Os endereços *unicast* podem ser do tipo: *link-local*, *unique-local* ou *global unicast*.
- *Anycast*: identifica um conjunto de *hosts*. Um pacote destinado a um endereço *anycast* será entregue ao *host* mais próximo de um grupo de *hosts*.
- *Multicast*: também identifica um conjunto de *hosts*. Porém, um pacote enviado a um endereço *multicast* atinge todos os *hosts* do conjunto. É uma comunicação de um-para-muitos.

Endereços *Unicast* do tipo *link-local* são reservados para comunicação apenas no enlace especifico onde a interface está conectada e não devem ser roteados, ou seja, nenhum roteador tem autorização para encaminhar pacotes com esses endereços para qualquer outra rede. Esses endereços são atribuídos automaticamente utilizando o prefixo `fe80::/64` e são fundamentais para a operação de várias funcionalidades do IPv6. Entre tais funcionalidades estão a autoconfiguração de endereços e descoberta de vizinhança, as quais serão abordadas com maiores detalhes nas seções seguintes.

Já os endereços *Unicast* do tipo global são públicos e roteáveis, acessíveis na Internet. Os endereços *Unicast unique-local* e *anycast* não são relevantes para a pesquisa e por isso não serão abordados em detalhes.

Endereços *multicast* são cruciais para o funcionamento do IPv6. Como exemplo, todos os *hosts* de uma rede local fazem parte de um grupo denominado *multicast all-nodes* que é identificado pelo endereço `ff02::1`. Justamente por meio desse grupo *multicast* que o endereço de *broadcast*, do IPv4, pode ser eliminado no IPv6.

2.2. *Neighbor Discovery Protocol*

O NDP é um protocolo construído com base nas mensagens do *Internet Control Message Protocol version* 6 (ICMPv6) (CONTA; DEERING; GUPTA, 2006). Logo, cinco tipos de mensagens do ICMPv6 foram reservadas para a realização de tarefas do NDP. São elas:

- *Router Solicitation* (RS): enviada pelos *hosts* para encontrar roteadores no mesmo enlace.
- *Router Advertisement* (RA): enviada por um roteador em resposta a um RS. Também pode ser enviada periodicamente sem ter sido solicitada, anunciando a presença do roteador no enlace e fornecendo informações para a autoconfiguração dos *hosts* na rede.
- *Neighbor Solicitation* (NS): enviado por um *host* para descobrir o endereço físico de outro *host*.
- *Neighbor Advertisement* (NA): mensagem em resposta a um NS mas também podendo ser enviada sem solicitação, informando a atualização de endereço de um *host*.
- *Redirect*: enviada por um roteador para redirecionar rotas.

A função do NDP, alvo de estudo para ataques *man in the middle* no IPv6, foi a descoberta de vizinhança, que realiza o processo de resolução de endereço físico através

 © 2017 Sociedade Brasileira de Computação

de um endereço lógico IPv6. Esse processo acontece quando um *host* tenta se comunicar com outro *host* cujo endereço físico de destino é desconhecido. As mensagens ICMPv6 utilizadas para essa tarefa são a NS e a NA.

A partir do momento em que o *host* passa a conhecer o endereço físico de outro, o NDP mantém essa informação em uma tabela, associando os endereços lógicos (IPv6) aos endereços físicos. Logo, quando o endereço físico de destino já existe na tabela, o mecanismo de descoberta de vizinhança não é utilizado. Segundo Moreiras et al. (2015), essa tabela é chamada de *Neighbor Cache*.

Além da descoberta de endereços físicos, o NDP fornece suporte para a realização de outras funcionalidades. Entre elas, a autoconfiguração de endereços discutida na próxima seção.

2.3. Autoconfiguração de endereços

Segundo Brito (2013), a autoconfiguração de endereços é um processo em que um *host* cria seu próprio endereço IPv6, baseado em informações locais e informações recebidas da rede. Há dois mecanismos de autoconfiguração.

Na Autoconfguração *Stateful*, os *hosts* obtêm configurações ou endereços de um servidor, o qual mantêm informações dos endereços que foram distribuídos na rede. Já na Autoconfiguração *Stateless* (SLAAC) um *host* cria seu próprio endereço IPv6, baseado em informações locais e informações recebidas por mensagens do tipo *Router Advertisement*.

No caso do experimento realizado nesta pesquisa, os *hosts* nos quais os endereços não foram manualmente configurados usaram a autoconfiguração SLAAC. Ainda de acordo com Brito (2013), esse mecanismo consiste basicamente em duas etapas:

1. Configuração do prefixo de rede. Essa etapa é possível porque o *host* aprende o prefixo de rede por meio da mensagem RA anunciada pelos roteadores.
2. Configuração do sufixo, que é gerado automaticamente a partir do endereço físico da interface de rede, combinado a uma função de expansão denominada IEEE EUI-64 (HINDEN; DEERING, 1998).

Como há a presença de *hosts* com sistemas operacionais Microsoft Windows nesse experimento, vale citar que esses sistemas não utilizam a função de expansão EUI-64 na autoconfiguração de endereço IPv6. A Microsoft optou por gerá-los aleatoriamente por motivos de segurança, e essa é uma prática de privacidade proposta na RFC4941 (NARTEN; DRAVES; KRISHNAN, 2007).

2.4. Definição de ataque do tipo *man in the middle*

O ataque *Man in the middle* é um tipo de ataque que tem como objetivo interceptar dados de comunicação de um ou mais *hosts*, por meio de técnicas que direcionam o tráfego de dados desses *hosts* para o atacante. Os *hosts* não percebem que seus dados estão sendo interceptados. O objetivo do atacante ao interceptar dados de comunicação é buscar informações que lhe deem vantagens, como dados de *logins*, informações confidenciais, números de cartões de crédito, contas de e-mail, entre outros (CONTI; DRAGONI; LESYK, 2016; ULBRICH; VALLE, 2007).

 © 2017 Sociedade Brasileira de Computação

2.4.1. Ataque *Man in the middle* no IPv6

Segundo Brito (2013), podemos explorar a funcionalidade de anúncio de roteadores do NDP, onde uma máquina geraria mensagens RA falsas passando-se pelo roteador, para se tornar o dispositivo intermediário na comunicação.

De acordo com Florentino (2014), é possível que ocorra um ataque *man in the middle* enviando um anúncio RA forjado, no qual o atacante passa a ser o *gateway* da vítima podendo redirecionar e capturar o tráfego deste *host*.

O experimento desta pesquisa se difere por não explorar a funcionalidade do NDP de anúncio de roteadores (RA) para o *man in the middle*, e sim por explorar outra funcionalidade do NDP, a de descoberta de vizinhança através de mensagens do tipo *Neighbor Advertisement* (NA). Os experimentos estão descritos na seção a seguir.

3. Experimento

O experimento que será detalhado nas seções a seguir teve como objetivo explorar o mecanismo de descoberta de vizinhança do NDP, através de mensagens NA, para alterar e forjar entradas nas tabelas *neighbor cache* dos *hosts* da rede local (LAN). Assim, os pacotes de dados são direcionados dos *hosts* com destino externo para a máquina do atacante.

Neste cenário foi implementada uma rede com tráfego somente em IPv6, onde o objetivo foi explorar possíveis falhas do NDP em relação ao ataque *man in the middle*. A Figura 1 a seguir mostra a configuração do cenário, com a representação do pacote malicioso utilizado no experimento na parte inferior da figura.

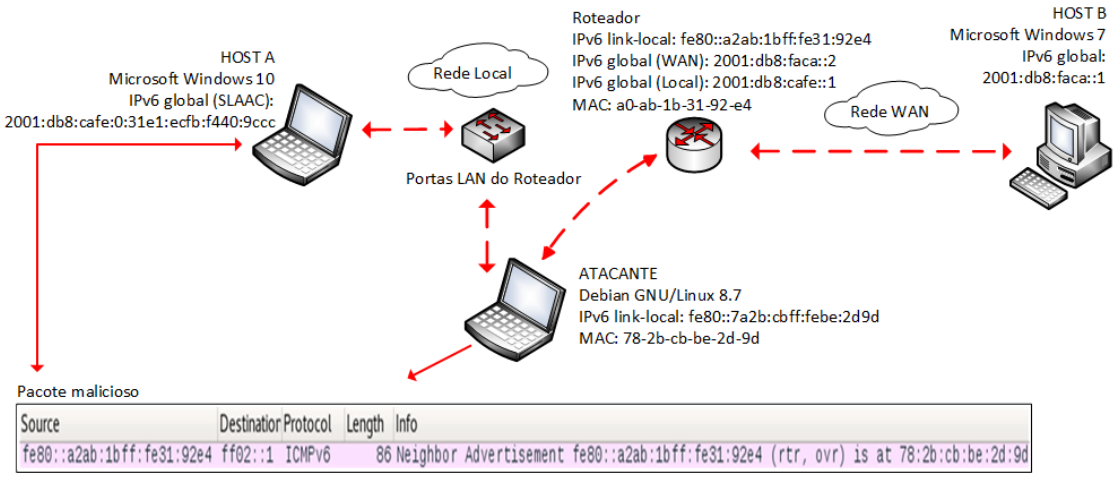


Figura 1. Cenário do Experimento.

Na Figura 1 estão os seguintes dispositivos e sistemas: um roteador D-Link DIR-615; um *Host* A com Microsoft Windows 10 Student; um *Host* B com Microsoft Windows 7 Professional; e um *Host* Atacante com Debian Jessie GNU/Linux.

3.1. Configuração de rede

O roteador usado no experimento oferece roteamento entre duas redes diferentes: uma rede local, geralmente uma rede restrita a um local físico, como uma casa ou escritório,

© 2017 Sociedade Brasileira de Computação

e a outra denominada rede de longa distância (WAN). Logo, a rede local é composta pelo *Host* A e o *Host* Atacante. O *switch* na Figura 1 serve apenas para ilustrar que os dispositivos da rede local estão ligados às portas da rede local do Roteador. A rede WAN é composta pelo *Host* B.

Através da mensagem RA, o roteador ilustrado na Figura 1 anuncia sua presença no enlace em intervalos regulares de tempo, informando o prefixo de rede `2001:db8:cafe::/64` para os *hosts* da rede interna. O roteador atua também como *gateway* (rota padrão). Baseado no prefixo de rede informado nas mensagens RA, os *hosts* usam o mecanismo *Stateless Address Autoconfiguration* (SLAAC) para se endereçarem, definindo os endereços IPv6 que os identificam de forma única dentro da rede (NARTEN; DRAVES; KRISHNAN, 2007).

Com relação à rede WAN, o prefixo de rede é `2001:db8:faca::/64`. Manualmente, foram configurados os endereços `2001:db8:faca::2` e `2001:db8:faca::1` para o roteador e para o *Host* B, respectivamente. Essa rede implementa de maneira mínima uma topologia em que duas redes diferentes se comunicam, simulando uma rede LAN acessando a Internet, por exemplo.

3.2. Serviços e Ferramentas

Foram configurados serviços e ferramentas nos *hosts* que permitissem a ilustração dos experimentos assim como seus resultados. São os seguintes:

- No *Host* A: FileZilla Client 3.24 – Cliente FTP; Firewall nativo do Windows ativo; e Windows Defender ativo (a ativação desses dois últimos é o padrão do sistema operacional).
- No *Host* B: FileZilla Server 0.9.60 - Servidor FTP; e Firewall nativo do Windows ativo, com regra de permissão para o serviço de servidor FTP FileZilla Server.
- No *Host* Atacante: Scapy - Manipulador de pacotes de rede (LALET, 2017); e Wireshark - Analisador de tráfego de rede (COMBS, 2017).

3.3. O ataque *man in the middle* no experimento

A técnica utilizada no experimento pressupõe que o atacante não conhece a rede, ou seja, o atacante desconhece informações tais como a quantidade de *hosts* ativos, que sistemas operacionais esses *hosts* utilizam, e se há roteadores na rede local. O atacante consegue reunir as informações necessárias para realizar o ataque consultando seu próprio sistema operacional, que no caso é o Debian GNU/Linux. As informações são as seguintes:

- Se há um *gateway* na rede e qual seu endereço IPv6 de *link-local*.
- Qual é o endereço físico da interface de rede do atacante.

Com as informações coletadas, o atacante monta um pacote de rede do tipo NA, usando a ferramenta Scapy. O objetivo do atacante com esse pacote é se passar pelo *gateway* da rede, realizando uma atualização de endereço físico nas tabelas *Neighbor Cache* de todos os *hosts* da mesma rede. Com isso, faz com que os *hosts* acreditem que o endereço físico do roteador é o endereço físico da sua interface de rede. O resultado disso é que todos os pacotes com destino à rede WAN (externa), passem pela máquina do atacante.

 © 2017 Sociedade Brasileira de Computação

A seguir está a descrição dos campos mais importantes do pacote malicioso:

- Na camada Ethernet
  - *Destination*: endereço físico `33:33:00:00:00:01`, usado quando o endereço IPv6 de destino é *Multicast all-nodes* `ff02::1`.
  - *Source*: endereço físico da interface de rede do próprio atacante.
- Na camada IPv6
  - *Source*: endereço de *link-local* do *gateway* da rede.
  - *Destination*: endereço multicast all-nodes `ff02::1`.
- Na camada ICMPv6
  - *Type: Neighbor Advertisement*.
  - *Flags*: $R = 1$, que indica que quem está enviando o pacote é um roteador. $S = 0$ que indica que o pacote não é uma resposta a um *Neighbor Solicitation*. $O = 1$ que indica que a informação do pacote é uma atualização de endereço de algum *host* da rede, no caso o *gateway*.
  - *Target Address*: Indica endereço IP associado às informações das *flags*. Nesse caso é o endereço *link-local* do *gateway*.
  - *Target link-layer address*: indica o endereço físico da interface de rede de origem do pacote, no caso a do próprio atacante.

Vale citar que a configuração desse pacote malicioso é bem semelhante a de um pacote do tipo RA. No experimento, esse pacote preparado pelo atacante é reenviado na rede em um intervalo regular de 3 segundos.

Outra configuração importante no sistema do atacante é a ativação do *IP Forwarding* no *Kernel* Linux. O sistema do atacante receberá pacotes os quais ele não é o destino. Portanto, essa configuração permite que o sistema do atacante realize roteamento, ou seja, encaminhe os pacotes para seus destinos originais. Do contrário, o sistema do atacante impediria conexões da rede local à rede WAN.

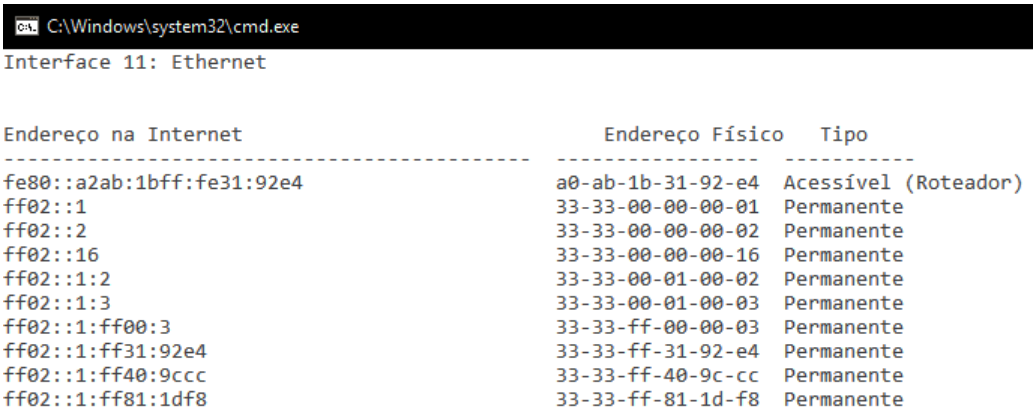
4. Resultados

Ao enviar o pacote malicioso na rede, o atacante atinge todos os *hosts* ativos na rede pelo fato de destinar o pacote ao endereço *multicast all-nodes* (`ff02::1`). Logo, no experimento desta pesquisa, o *Host* A era o sistema ativo na rede e seu sistema operacional, Windows 10, aceita a informação do pacote malicioso atualizando sua tabela *Neighbor Cache*. A Figura 2 a seguir ilustra a tabela *Neighbor Cache* do *Host* A antes do pacote malicioso ser enviado na rede.

Observe-se que o endereço `fe80::a2ab:1bff:fe31:92e4` está associado ao endereço físico `a0:ab:1b:31:92:e4` e que o mesmo é do tipo roteador. Após o envio do pacote malicioso na rede, a *Neighbor Cache* do *Host* A fica conforme ilustrada na Figura 3.

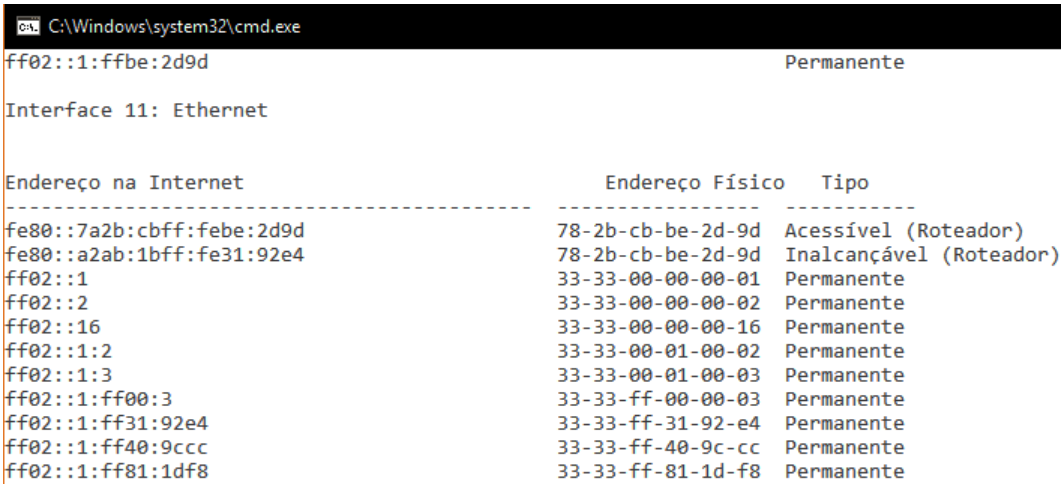
Observa-se que o endereço do roteador da rede passou a ser a segunda entrada na tabela e também está associado ao endereço físico da máquina do atacante, `78:2b:cb:be:2d:9d`. Além disso, o endereço de *link-local* do atacante – a primeira entrada na tabela – passa a ser reconhecido como roteador também. O resultado disso é que o *Host* A passa a enviar os pacotes com destino externo para a máquina do atacante.

 © 2017 Sociedade Brasileira de Computação



```
C:\Windows\system32\cmd.exe
Interface 11: Ethernet

Endereço na Internet                          Endereço Físico   Tipo
--------------------------------------------  ----------------  -----------
fe80::a2ab:1bff:fe31:92e4                     a0-ab-1b-31-92-e4  Acessível (Roteador)
ff02::1                                       33-33-00-00-00-01  Permanente
ff02::2                                       33-33-00-00-00-02  Permanente
ff02::16                                      33-33-00-00-00-16  Permanente
ff02::1:2                                     33-33-00-01-00-02  Permanente
ff02::1:3                                     33-33-00-01-00-03  Permanente
ff02::1:ff00:3                                33-33-ff-00-00-03  Permanente
ff02::1:ff31:92e4                             33-33-ff-31-92-e4  Permanente
ff02::1:ff40:9ccc                             33-33-ff-40-9c-cc  Permanente
ff02::1:ff81:1df8                             33-33-ff-81-1d-f8  Permanente
```

Figura 2. Tabela *Neighbor Cache* do *Host* A antes do ataque.

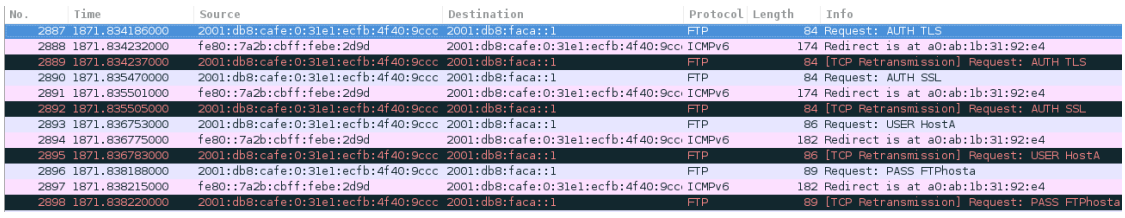


```
C:\Windows\system32\cmd.exe
ff02::1:ffbe:2d9d                                         Permanente

Interface 11: Ethernet

Endereço na Internet                          Endereço Físico   Tipo
--------------------------------------------  ----------------  -----------
fe80::7a2b:cbff:febe:2d9d                     78-2b-cb-be-2d-9d  Acessível (Roteador)
fe80::a2ab:1bff:fe31:92e4                     78-2b-cb-be-2d-9d  Inalcançável (Roteador)
ff02::1                                       33-33-00-00-00-01  Permanente
ff02::2                                       33-33-00-00-00-02  Permanente
ff02::16                                      33-33-00-00-00-16  Permanente
ff02::1:2                                     33-33-00-01-00-02  Permanente
ff02::1:3                                     33-33-00-01-00-03  Permanente
ff02::1:ff00:3                                33-33-ff-00-00-03  Permanente
ff02::1:ff31:92e4                             33-33-ff-31-92-e4  Permanente
ff02::1:ff40:9ccc                             33-33-ff-40-9c-cc  Permanente
ff02::1:ff81:1df8                             33-33-ff-81-1d-f8  Permanente
```

Figura 3. Tabela *Neighbor Cache* do *Host* A depois do ataque.

Para verificar o efeito desse resultado, foi realizada uma conexão FTP entre o *Host* A e o *Host* B, lembrando que o *Host* B pertence a outro segmento de rede. Usando as ferramentas citadas na Seção 3.2, o *Host* A se conecta ao servidor FTP informando o endereço IPv6 global do *Host* B (`2001:db8:faca::1`).

Usando a ferramenta de análise de pacotes de rede na máquina do atacante, foi possível observar que os pacotes de rede da conexão FTP do *Host* A para o *Host* B foram interceptados. Nesse caso, as informações de usuário e a senha foram expostas. A interceptação dos dados pode ser observada na Figura 4 a seguir.



No.	Time	Source	Destination	Protocol	Length	Info
2887	1871.834186000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	84	Request: AUTH TLS
2888	1871.834232000	fe80::7a2b:cbff:febe:2d9d	2001:db8:cafe:0:31e1:ecfb:4f40:9cc	ICMPv6	174	Redirect is at a0:ab:1b:31:92:e4
2889	1871.834237000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	84	[TCP Retransmission] Request: AUTH TLS
2890	1871.835470000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	84	Request: AUTH SSL
2891	1871.835501000	fe80::7a2b:cbff:febe:2d9d	2001:db8:cafe:0:31e1:ecfb:4f40:9cc	ICMPv6	174	Redirect is at a0:ab:1b:31:92:e4
2892	1871.835505000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	84	[TCP Retransmission] Request: AUTH SSL
2893	1871.836753000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	86	Request: USER HostA
2894	1871.836775000	fe80::7a2b:cbff:febe:2d9d	2001:db8:cafe:0:31e1:ecfb:4f40:9cc	ICMPv6	182	Redirect is at a0:ab:1b:31:92:e4
2895	1871.836783000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	86	[TCP Retransmission] Request: USER HostA
2896	1871.838188000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	89	Request: PASS FTPhosta
2897	1871.838215000	fe80::7a2b:cbff:febe:2d9d	2001:db8:cafe:0:31e1:ecfb:4f40:9cc	ICMPv6	182	Redirect is at a0:ab:1b:31:92:e4
2898	1871.838220000	2001:db8:cafe:0:31e1:ecfb:4f40:9ccc	2001:db8:faca::1	FTP	89	[TCP Retransmission] Request: PASS FTPhosta

Figura 4. Pacotes de rede interceptados.

© 2017 Sociedade Brasileira de Computação

Analisando a tabela dos pacotes capturados na Figura 4, observa-se que nos pacotes de número 2893 e 2896 (primeira coluna da tabela), são encontradas as informações de usuário e senha na coluna Info, respectivamente. Essas são as informações de *login* usadas no experimento.

5. Conclusões

Com os resultados dos experimentos foi possível concluir que o mecanismo de descoberta de vizinhos do IPv6 pode ser explorado para realizar um ataque *man in the middle*. O ataque permite que dados com destino a uma rede externa sejam interceptados e posteriormente analisados pelo atacante. Em outras palavras, pacotes de rede que precisam ser roteados são passiveis de serem interceptados por outro *host* da mesma rede interna através desse ataque que explora o protocolo NDP.

Um grande aliado desse ataque é o tipo de endereço *multicast*. Sem ele, a descoberta de *hosts* ativos na rede torna-se inviável, já que uma varredura de endereços em uma rede IPv6 é ineficiente devido a alta quantidade de endereços possíveis.

Portanto, sem o uso do endereço *multicast all-nodes* e sem a interceptação de pacotes que não sejam destinados a rede externa, o ataque deve tornar-se bem mais complicado. Vale observar que a disparada dos pacotes NDP elaborados no experimento pode servir para a descoberta de *hosts* ativos em uma rede IPv6.

Como trabalhos futuros, propomos a possibilidade de realizar o ataque *man in the middle* em rede IPv6 local, onde os dados interceptados têm como destino a mesma rede de origem. Também propomos como trabalhos futuros a aplicação da mesma técnica de ataque utilizada nesta pesquisa em redes de cenário misto IPv4/IPv6, onde há a existência de mecanismos de transição que permitem a coexistência das duas versões do protocolo na rede. Nesta proposta, o objetivo é verificar possíveis influências dos mecanismos de transição no método do ataque.

Referências

BRITO, S. H. B. *IPv6 - O Novo Protocolo da Internet*. São Paulo-SP: Novatec, 2013. 208 p. ISBN 978-85-7522-374-1.

COMBS, G. *Wireshark*. 2017. Disponível em: ⟨https://www.wireshark.org⟩.

CONTA, A.; DEERING, S.; GUPTA, M. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. [S.l.], 2006. Disponível em: ⟨http://www.rfc-editor.org/rfc/rfc4443.txt⟩.

CONTI, M.; DRAGONI, N.; LESYK, V. A Survey of Man In The Middle Attacks. *IEEE Communications Surveys & Tutorials*, v. 18, n. 3, p. 2027–2051, 2016. ISSN 1553-877X.

DENIS, M.; ZENA, C.; HAYAJNEH, T. Penetration testing: Concepts, attack methods, and defense strategies. In: *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. Farmingdale, NY, USA: IEEE, 2016. p. 1–6. ISBN 978-1-4673-8490-2.

FLORENTINO, A. A. *IPv6 na prática*. 1. ed. Rio de Janeiro-RJ: Alta Books, 2014. 168 p. ISBN 978-85-61024-30-3.

 © 2017 Sociedade Brasileira de Computação

GUGELMIN, F. *Site oficial das Olimpíadas suportou 223 milhões de ataques*. 2016. Disponível em: ⟨https://www.tecmundo.com.br/seguranca/110886-site-oficial-olimpiadas-suportou-223-milhoes-ataques.htm⟩.

HINDEN, R. M.; DEERING, S. E. *IP Version 6 Addressing Architecture*. [S.l.], 1998. Disponível em: ⟨http://www.rfc-editor.org/rfc/rfc2373.txt⟩.

IPv6.br. *Endereçamento IPv6*. 2012. Disponível em: ⟨http://ipv6.br/post/enderecamento⟩.

LALET, P. *Scapy: the python-based interactive packet manipulation program & library*. 2017. Disponível em: ⟨https://github.com/secdev/scapy⟩.

MOREIRAS, A. M. et al. *Laboratório de IPv6: aprenda na prática usando um emulador de redes*. São Paulo-SP: Novatec, 2015. 416 p. ISBN 978-85-7522-418-2.

MORIMOTO, C. E. *Redes: guia prático*. 2. ed. Porto Alegre: GHD Press e Sul Editores, 2008. 560 p. ISBN 978-85-99593-09-7.

MOTA FILHO, J. E. *Análise de tráfego em redes TCP/IP: utilize tcpdump na análise de tráfegos em qualquer sistema operacional*. São Paulo: Novatec, 2013. 416 p.

NARTEN, T.; DRAVES, R.; KRISHNAN, S. *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*. [S.l.], 2007. Disponível em: ⟨https://tools.ietf.org/html/rfc4941⟩.

NARTEN, T. et al. *Neighbor Discovery for IP version 6 (IPv6)*. [S.l.], 2007. Disponível em: ⟨http://www.rfc-editor.org/rfc/rfc4861.txt⟩.

ULBRICH, H. C.; VALLE, J. D. *Universidade Hacker*. São Paulo: Digerati Books, 2007. 352 p. ISBN 8578730968.

 © 2017 Sociedade Brasileira de Computação

Investigando as Percepções de Privacidade do Usuário em Redes Sociais Online

Ingrid L. dos Santos[1], Karla S. Pereira[1], Eduardo L. Feitosa[1],

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Caixa Postal 69077-000 – Manaus – AM – Brasil

{ils,karla.susiane,efeitosa}@icomp.ufam.edu.br

Abstract. *The exponential growth of users in Online Social Networking (OSN) increases the security and privacy implications of published private information. The objective of this paper is to investigate and measure user perceptions of privacy in the context of OSN. To that end, four hypotheses were elaborated and a questionnaire was set up to evaluate if OSN users understood the concept of privacy. The result showed that OSN users are not concerned about their privacy.*

Resumo. *O crescimento exponencial de usuários em Redes Sociais Online (RSO) aumentam as implicações de segurança e privacidade sobre as informações privadas publicadas. O objetivo deste artigo é investigar e mensurar as percepções de privacidade do usuário no contexto de RSO. Para tanto, quatro hipóteses foram elaboradas e um questionário foi montado para avaliar se os usuários de RSO compreendem o conceito de privacidade. O Resultado mostrou que os usuários de RSO não estão preocupados com sua privacidade.*

1. Introdução e Motivação

O poder e a abrangência das Redes Sociais Online (RSO), tais como Facebook, Twitter, Instagram e diversas outras, possibilita meios atraentes de interação social e comunicação entre os usuários. Dessa forma, é possível encontrar e compartilhar notícias sobre assuntos variados, localizar amigos distantes e publicar fotos e vídeos. Uma vez que as RSOs tornaram-se parte significante do dia a dia da população, elas se tornaram alvos de ataques e atividades maliciosas. Embora as organizações invistam em soluções para detectar problemas (intencionais e não intencionais) relacionados à segurança dos dados e a privacidade dos usuários, grande parte dessas soluções é baseada no consentimento, apoio ou configuração dos usuários.

É neste contexto que as seguintes questões precisam ser discutidas: Como os usuários de RSO entendem e percebem o conceito de privacidade? E como eles lidam com a sua própria privacidade dentro das RSOs? Estariam eles aptos a traçar limites que indiquem onde suas relações interpessoais em RSO começam e terminam? Ou será que eles são capazes de gerenciar diretamente a sua privacidade através das limitações de divulgação de informações?

Partindo destes questionamentos, este artigo propõe uma investigação para mensurar as percepções de privacidade do usuário no contexto de RSO. A partir de uma revisão bibliográfica, foram elaboradas hipóteses, baseadas em construtores, sobre o comportamento dos usuários em relação a sua privacidade em RSO. Para validar essas hipóteses,

© 2017 Sociedade Brasileira de Computação

um questionário foi elaborado e aplicado a um grupo de voluntários. O intuito deste trabalho de iniciação científica, renovado por mais um ano, é entender como os usuários percebem privacidade digital e como definem seus limites de relacionamento com outros usuários da(s) rede(s) social(is) que utilizam.

Este trabalho está dividido da seguinte forma: na Seção 2 é apresentado um breve conceito de privacidade no contexto das Redes Sociais Online; na Seção 3 são apresentados os trabalhos relacionados; na Seção 4 são detalhadas as hipóteses levantadas para este estudo, incluindo seus construtores e questões assertivas; na Seção 5 é apresentado o experimento envolvendo a aplicação do método proposto e a avaliação e, por fim, na Seção 6 encontram-se as considerações finais do trabalho.

2. Privacidade em Redes Sociais Online

As Redes Sociais Online (RSO) são serviços onde os usuários podem criar perfis públicos ou semi-públicos, conectar-se com outros usuários e visualizar as conexões de suas conexões [Ellison et al. 2007]. Com a Internet contando com mais de três bilhões de usuários[1], e consequentemente o aumento de usuários de RSOs, surgiram preocupações em relação à privacidade, visto que existe a possibilidade da divulgação de informações privadas (identidades, endereços virtuais ou físicos, estilo de vida, entre outros).

Mas o que é privacidade? Milberg *et al.* [Milberg et al. 1995] argumentam que a privacidade é um conceito complexo e de difícil obtenção de uma definição, possivelmente porque as atitudes em relação à privacidade são determinadas culturalmente. Dado o contexto das RSOs, o princípio básico para que o usuário sinta-se confortável em fazer parte dessa rede de informações é que ele tenha conhecimento sobre quais de suas informações privadas estão suscetíveis a terceiros e também o controle sob as mesmas. Dessa forma, a definição que melhor se adequa ao contexto é a dada por Westin [Westin 1968], onde a privacidade é "o direito de o indivíduo de decidir quais informações sobre si mesmo devem ser compartilhadas com outros e sob quais condições."

3. Trabalhos Relacionados

Como mencionado no começo deste artigo, as RSOs se tornaram alvo de ataques e atividades maliciosas, afetando a segurança dos dados e a privacidade. Mas como os usuários de RSO entendem e percebem o conceito de privacidade? E como eles lidam com a sua própria privacidade dentro das RSOs? Com base nestas perguntas, uma revisão bibliográfica, utilizando a técnica *snowballing*, foi realizada para identificar as percepções de privacidade do usuário no contexto de RSO.

A revisão bibliográfica resultou em 37 publicações (etapa 1), da biblioteca digital Scopus, que foram analisadas e ao final (etapa 2) apenas 14 publicações sobre privacidade foram selecionadas. Com base na revisão, os construtores foram selecionados e as hipóteses elaboradas. A seguir, alguns desses trabalhos são descritos.

Wisniewski *et al.* [Wisniewski et al. 2015] propuseram um modelo para observar a relação entre a divulgação de informações pessoais e as preocupações de privacidade do usuário no uso das RSOs, através de duas funcionalidades que proporcionam a divulgação

[1]http://www.internetlivestats.com

 © 2017 Sociedade Brasileira de Computação

de informações (*tagging* e a integração de aplicativos de terceiros (*apps*)). Como resultado, eles destacam o valor social que os indivíduos aplicam à possibilidade de interagir com outros amigos através das RSO.

Taneja *et al.* [Taneja et al. 2014] desenvolveram e testaram um modelo capaz de investigar como as crenças do usuário afetam as atitudes individuais e a intenção de usar os controles de privacidade disponíveis no Facebook. Os resultados mostraram que a intenção de usar os controles de privacidade depende da atitude e da norma social, e que a atitude é conduzida pelas crenças de um indivíduo em relação aos benefícios do uso ou não uso dos referidos controles.

Benson *et al.* [Benson et al. 2015] realizaram um estudo focado nos usuários e nas consequências do comportamento da divulgação de informações e perda de privacidade como resultado do envolvimento com as tecnologias emergentes, por exemplo RSO. Os autores identificaram construtores ligados à privacidade e a partir deles desenvolveram hipóteses de pesquisa. Para a verificação e validação, usaram um questionário online, onde os resultados mostraram que as percepções de uso do controle de informações pessoais têm um efeito negativo e estatístico sobre a divulgação de informações, e que quando os usuários se sentem seguros de que têm mais controle sobre suas informações pessoais, eles são mais cuidadosos em divulgar informações sobre si mesmo.

Li [Li 2015] estudou o impacto da conscientização dos usuários e a autoeficácia das opções de controle atuais disponíveis sobre suas percepções relacionadas à privacidade no ambiente das RSOs. O autor formulou dez (10) hipóteses que foram testadas via questionário online. Como resultado, ele verificou que a consciência dos usuários sobre as opções de controle disponíveis está positivamente associada ao controle de privacidade dos indivíduos e seu impacto é totalmente mediado pela autoeficácia percebida nas RSOs.

3.1. Discussão

Todos os trabalhos relacionados descritos tentam, de alguma forma, inferir, compreender e modelar alguns aspectos da privacidade dos usuários em RSO, mas de forma isolada ou simplesmente direcionada. Taneja *et al.* focou na necessidade de conscientização dos usuários (crenças) em relação à privacidade e no incentivo do uso de controles de privacidade. Wisniewski *et al.* focou em compreender a relação entre a divulgação de informações pessoais e as preocupações de privacidade do usuário no uso das RSOs. Benson *et al.* focou em identificar a relação entre divulgação de informações e três aspectos: controle de informações pessoais, conscientização dos usuários e notificações de segurança. Li ofereceu uma visão do impacto da conscientização sobre as opções de controle relacionadas à privacidade dos indivíduos em RSO.

A visão deste trabalho é mais ampla. Por isso, foram observados e escolhidos seis (06) construtores, relacionados aos trabalhos descritos, para auxiliar na composição (elaboração) das hipóteses sobre a privacidade dos usuários em RSO.

4. Hipóteses

A partir da revisão bibliográfica identificamos quatro (04) trabalhos que mais se aproximavam dos objetivos desta pesquisa. Os construtores, definidos como uma forma de generalizar o resultado do experimento para o conceito ou a teoria por trás do experimento, bem como as hipóteses foram embasadas nos trabalhos descritos na Seção 3. A Tabela

 © 2017 Sociedade Brasileira de Computação

1 apresenta as quatro hipóteses elaboradas para identificar a percepção de privacidade do usuário de RSO, incluindo os construtores e as questões assertivas.

Tabela 1. Hipóteses, Construtores e Questões Assertivas empregadas na Investigação. Fonte: Autor

Hipótese	Construtor	Questões Assertivas
H1. O usuário conhece e sabe utilizar os mecanismos de controle de segurança e privacidade da sua RSO	Autoeficácia Percebida (PSE)	**PSE1** Me sinto capaz de usar as opções de controle de privacidade da RSO para controlar minha timeline.
		PSE2 Me sinto confiante em usar as opções de controle de privacidade para gerenciar as *tags* na RSO.
		PSE3 Me sinto capaz de usar as opções de controle de privacidade da RSO para controlar minhas fotos.
	Controle Comportamental Percebido (PBC)	**PBC1** Se eu quiser, posso usar os controles de privacidade disponíveis na RSO.
		PBC2 Usar os controles de privacidade disponíveis na RSO é de inteira responsabilidade minha.
		PBC3 Tomar as medidas necessárias para proteger minhas informações pessoais na RSO está inteiramente sob meu controle.
H2. O usuário tem conhecimento das políticas de privacidade e uso da informação na RSO	Eficácia percebida da política de privacidade (PEPP)	**PEPP1** Acredito que as declarações de privacidade da RSO refletem o comprometimento deles em proteger minhas informações pessoais.
		PEPP2 Com as declarações de privacidade da RSO, acredito que minhas informações pessoais serão mantidas privadas e confidenciais.
		PEPP3 Acredito que as declarações de privacidade da RSO são uma maneira efetiva de demonstrar o comprometimento da companhia com a privacidade.
	Uso da Informação (UI)	**UI1** Acredito que a RSO não deveria utilizar as minhas informações pessoais para nenhum propósito, a menos que eu tenha autorizado.
		UI2 Acredito que se eu forneci informações pessoais para uma RSO, por qualquer que seja a razão, a companhia da RSO não deve sob hipótese alguma fazer uso dessas informações, não importando a razão de uso.
		UI3 Acredito que a RSO não deva em hipótese alguma vender minhas informações pessoais armazenadas em seus bancos de dados para outras companhias.
		UI4 Acredito que a RSO não deve em hipótese alguma compartilhar as minhas informações pessoais com outras companhias, a menos que tenha minha autorização.
H3. O usuário se preocupa com a sua privacidade na RSO	Preocupação de privacidade geral (GPC)	**GPC1** Estou preocupado (a) que muita informação sobre mim e minhas atividades online estão sendo coletadas propositalmente.
		GPC2 Estou preocupado (a) com minha privacidade quando estou navegando na RSO.
		GPC3 Estou preocupado (a) que minhas informações pessoais divulgadas online podem ser mal utilizadas.
		GPC4 Estou preocupado (a) que minhas informações pessoais divulgadas online podem ser acessadas por terceiros desconhecidos.
H4. O usuário acredita estar seguro quando usa a RSO.	Privacidade Percebida (PP)	**PP1** Sinto que tenho privacidade suficiente quando uso a RSO.
		PP2 Me sinto confortável com a quantidade de privacidade que tenho na RSO.
		PP3 Sinto que minha privacidade está preservada quando uso a RSO.
		PP4 Me sinto confortável com meu estado de privacidade quando uso a RSO.

É importante ressaltar que algumas questões assertivas foram adaptadas para refletir melhor as hipóteses no contexto de RSO, e para mensurar cada resposta foi empregada uma escala *likert* de sete (07) pontos, variando de (1) Discordo Totalmente, (2) Discordo Fortemente, (3) Discordo Parcialmente, (4) Nem Concordo nem Discordo, (5) Concordo Parcialmente, (6) Concordo Fortemente e (7) Concordo Totalmente.

A hipótese **H1** - **O usuário conhece e sabe utilizar os mecanismos de controle de segurança e privacidade da sua RSO**, busca compreender se os usuários estão cientes da existência dos mecanismos de segurança fornecidos pelas RSO, e se as utilizam como forma de proteger suas informações pessoais. Para sustentá-la, dois construtores provenientes da literatura foram empregadas. O primeiro é o PSE (*Perceived Self-efficacy*), autoeficácia percebida, que é definida como o nível de conscientização do usuário sobre a manipulação dos artefatos de controle da privacidade em RSO [Li 2015]. Para esse construtor foram definidas três (03) questões assertivas que visam questionar o usuário sobre a sua capacidade em manipular os controles de privacidade oferecidos pela RSO. O se-

 © 2017 Sociedade Brasileira de Computação

gundo é o PBC (*Perceived Behavioral Control*), controle comportamental percebido, que é definido como o grau de percepção do usuário em relação ao papel que ele desempenha sobre o uso e a configuração dos controle de privacidade em RSO [Taneja et al. 2014]. Para esse construtor foram definidas três (03) questões assertivas que visam questionar o usuário sobre seu entendimento da sua responsabilidade no uso ou não dos controles de privacidade oferecidos pela RSO.

A hipótese **H2** - **O usuário tem conhecimento das políticas de privacidade e uso da informação na RSO**, visa identificar se os usuários possuem alguma noção sobre o conteúdo das políticas de privacidade das RSOs e como as organizações lidam com seus dados. Em outras palavras, se os usuários de fato leêm os termos de consentimento e as políticas de privacidade antes de aceitar usar a RSO. Para sustentá-la, dois construtores provenientes da literatura foram empregadas. O primeiro é o PEPP (*Perceived Effectiveness of Privacy Policy*), eficácia percebida da política de privacidade, que é definida como a confiança que o usuário tem sobre as políticas de privacidade da RSO, acreditando que a mesma está agindo de boa fé para proteger suas informações [Wisniewski et al. 2015]. Para esse construtor foram definidas três (03) questões assertivas que visam questionar o usuário sobre a sua crença nas intenções de privacidade declaradas pelas RSO. O segundo é o UI (*Use of Information*), uso da informação, que é definida como a preocupação que o usuário tem sobre como a organização irá lidar com suas informações [Benson et al. 2015]. Para esse construtor foram definidas quatro (04) questões assertivas que visam questionar o usuário sobre seu conhecimento acerca da forma como as RSOs devem lidar com suas informações.

A hipótese **H3** - **O usuário se preocupa com a sua privacidade na RSO**, busca compreender os níveis de preocupação aparente que os usuários apresentam dentro da RSO. Para sustentá-la, emprega-se um construtor proveniente da literatura, o GPC (*General Privacy Concern*), preocupação de privacidade geral, definido como a medida de preocupação baseada nas características do usuário sobre privacidade percebida [Li 2015]. Para esse construtor foram definidas quatro (04) questões assertivas que visam questionar o usuário sobre a preocupação com sua privacidade dentro da RSO.

A última hipótese, **H4** - **O usuário acredita estar seguro quando usa a RSO**, procura identificar o nível de segurança que o usuário percebe ter ao usar as RSOs. Para sustentá-la, emprega-se o construtor PP (*Perceived Privacy*), privacidade percebida, que é definida como a quantidade de privacidade percebida pelos usuários de RSO [Li 2015]. Para esse construtor foram definidas quatro (04) questões assertivas que visam questionar o usuário sobre o quão preocupado com sua privacidade ele está ao utilizar a RSO.

5. Avaliação Experimental

Esta Seção descreve a aplicação do método proposto em uma amostra, bem como a análise dos resultados.

5.1. Protocolo Experimental

Para validar o método proposto nesta pesquisa, participaram do estudo piloto 10 estudantes de graduação, de cursos de computação, de uma universidade pública. O período de coleta de dados ocorreu no mês de Junho de 2017.

 © 2017 Sociedade Brasileira de Computação

5.2. Medidas de Avaliação

Como forma de avaliar as respostas do questionário, foi empregado o coeficiente de correlação de Pearson e o alfa de Cronbach [Cronbach 1951]. O primeiro mede o grau de correlação e sentido (positivo ou negativo) entre duas variáveis. Quando o coeficiente assume o valor de **1**, significa uma correlação perfeita positiva; quando assume **-1**, significa correlação negativa e inversamente proporcional; e quando assume **0**, significa que não existe dependência linear. Neste artigo, o o coeficiente de correlação de Pearson medirá se existe alguma correlação das respostas com a idade dos participantes.

O segundo analisa a confiabilidade das questões assertivas de cada construtor, estimando a confiabilidade do questionário através da correlação média das respostas do questionário e o perfil dessas respostas fornecidas pelos participantes. O coeficiente alfa (α) é calculado a partir da variância dos itens individuais e da variância da soma dos itens de cada questionário. Por meio do coeficiente alfa de Cronbach será possível mensurar a consistência interna dos construtores, ou seja, medir se as respostas ligadas ao mesmo codo questionário são semelhantes.

A interpretação do coeficiente de correlação de Pearson e do alfa de Cronbach é apresentada na Tabela 2.

Tabela 2. Interpretação do coeficiente de Pearson e do alfa de Cronbah.

Coeficiente de Pearson	Correlação indicada
0.9 para mais ou para menos	Correlação muito forte
0.7 a 0.9 positivo ou negativo	Correlação forte
0.5 a 0.7 positivo ou negativo	Correlação moderada
0.3 a 0.5 positivo ou negativo	Correlação fraca
0 a 0.3 positivo ou negativo	Correlação desprezível

Valor de Alfa	Consistência Interna
$\alpha > 0.9$	Excelente
$0.9 > \alpha \geqslant 0.8$	Boa
$0.8 > \alpha \geqslant 0.7$	Aceitável
$0.7 > \alpha \geqslant 0.6$	Duvidosa
$0.6 > \alpha \geqslant 0.5$	Pobre
$0.5 > \alpha$	Inaceitável

Para refutar ou aceitar as hipóteses, o Teste estatístico *t* foi usado para determinar se a hipótese nula deve ser rejeitada, com base nos dados da amostra. Adotou-se um nível de significância igual a 0.05. A hipótese é refutada se obtiver $p < 0.05$, caso contrário é aceita.

A aplicação do alfa de Cronbach, do coeficiente de Pearson e do Teste *t* entre os conjuntos de questões assertivas do questionário será feita por meio da ferramenta JASP (`https://jasp-stats.org`).

5.3. Resultados

Hipótese H1

Sobre a Autoeficácia Percebida (construtor PSE), a avaliação da confiabilidade do questionário, via alfa de Cronbach, obteve um resultado de 0.724, o que representa uma consistência interna aceitável. Já o coeficiente de correlação de Pearson mostrou uma correlação fraca entre a idade e a resposta das questões PSE1 (-0.450), PSE2 (-0.441) e PSE3 (-0.457). Ou seja, quanto menor a idade do participante maior é a sua autoeficácia percebida dos mecanismos de controle de segurança e privacidade na RSO.

Em relação as respostas, na questão PSE1, 0% discordaram totalmente, 10% discordaram fortemente, 10% discordaram parcialmente, 20% nem concordaram e nem dis-

 © 2017 Sociedade Brasileira de Computação

cordaram, 30% concordou parcialmente, 20% concordaram totalmente e 10% concordaram fortemente. Na questão PSE2, 10% discordaram totalmente, 0% discordaram fortemente, 10% discordaram parcialmente, 0 % nem concordaram e nem discordaram, 20% concordou parcialmente, 30% concordaram totalmente e 20% concordaram fortemente. Na questão PSE3, 10% discordaram totalmente, 20% discordaram fortemente, 10% discordaram parcialmente, 10 % nem concordaram e nem discordaram, 0% concordou parcialmente, 40% concordaram totalmente e 10% concordaram fortemente. O teste t da amostra para esse construto foi $p < 0.01$.

Sobre Controle Comportamental Percebido (construtor PBC), a avaliação da confiabilidade do questionário, via alfa de Cronbach, obteve um resultado de 0.655, o que representa consistência interna duvidosa. Já o coeficiente de correlação de Pearson mostrou que não ocorreu correlação do construto com a idade, mesmo que em PBC2 (-0.477) a correlação tenha sido fraca, pois em PBC1 (-0.111) e PBC3 (-0.067) a correlação foi desprezível.

Em relação as respostas, na questão PBC1, 0% discordaram totalmente, 0% discordaram fortemente, 0% discordaram parcialmente, 0 % nem concordaram e nem discordaram, 50% concordou parcialmente, 30% concordaram totalmente e 20% concordaram fortemente. Na questão PBC2, 10% discordaram totalmente, 20% discordaram fortemente, 10% discordaram parcialmente, 0 % nem concordaram e nem discordaram, 40% concordou parcialmente, 10% concordaram totalmente e 10% concordaram fortemente. Na questão PEPP2 10% discordaram totalmente, 20% discordaram fortemente, 0% discordaram parcialmente, 20% nem concordaram e nem discordaram, 20% concordou parcialmente, 20% concordaram totalmente e 10% concordaram fortemente. O teste t da amostra para esse construtor foi $p < .001$.

Com base nas respostas e, principalmente, no teste t, pode-se afirmar que a hipótese do usuário conhecer e saber utilizar os mecanismos de controle de segurança e privacidade da sua RSO foi **refutada**.

Hipótese H2

Sobre a Eficácia Percebida das Políticas de Privacidade (construtor PEPP), a avaliação da confiabilidade do questionário, via alfa de Cronbach, obteve um resultado de 0.497, o que representa uma consistência interna inaceitável. Já o coeficiente de correlação de Pearson mostrou que não ocorreu correlação do construtor com a idade, mesmo que em PEPP2 (-0.823) a correlação tenha sido forte, pois em PEPP1 (-0.068) e PEPP3 (-0.268) a correlação foi inaceitável.

Em relação as respostas, na questão PEPP1, 0% discordaram totalmente, 0% discordaram fortemente, 30% discordaram parcialmente, 0% nem concordaram e nem discordaram, 20% concordou parcialmente, 30% concordaram totalmente e 20% concordaram fortemente, Na questão PEPP2, 20% discordaram totalmente, 0% discordaram fortemente, 10% discordaram parcialmente, 0 % nem concordaram e nem discordaram, 20% concordou parcialmente, 50% concordaram totalmente e 0% concordaram fortemente, Na questão PEPP3, 10% discordaram totalmente, 30% discordaram fortemente, 0% discordaram parcialmente, 20 % nem concordaram e nem discordaram, 0% concordou parcialmente, 0% concordaram totalmente e 10% concordaram fortemente. O teste *t* da amostra para esse construtor foi $p < 0.01$.

 © 2017 Sociedade Brasileira de Computação

Sobre Uso da Informação (construtor UI), a avaliação da confiabilidade do questionário, via alfa de Cronbach, obteve resultado de 0.695, o que representa consistência interna duvidosa. Já o coeficiente de correlação de Pearson mostrou uma correlação fraca entre a idade e a resposta das questões UI2(0.372) e UI3 (0.342) e uma correlação moderada entre a idade e a resposta das questões UI1 (0.523) e UI2 (0.571). Ou seja, quanto menor a idade do participante maior é a sua preocupação em como a organização irá lidar com suas informações.

Em relação as respostas, na questão UI1, 0% discordaram totalmente, 0% discordaram fortemente, 30% discordaram parcialmente, 0% nem concordaram e nem discordaram, 0% concordou parcialmente, 10% concordaram totalmente e 60% concordaram fortemente. Na questão UI2, 20% discordaram totalmente, 0% discordaram fortemente, 20% discordaram parcialmente, 0% nem concordaram e nem discordaram, 30% concordou parcialmente, 20% concordaram totalmente e 10% concordaram fortemente, na questão UI3, 0% discordaram totalmente, 0% discordaram fortemente, 20% discordaram parcialmente, 0% nem concordaram e nem discordaram, 10% concordou parcialmente, 0% concordaram totalmente e 70% concordaram fortemente. na questão UI4, 0% discordaram totalmente, 10% discordaram fortemente, 20% discordaram parcialmente, 0% nem concordaram e nem discordaram, 0% concordou parcialmente, 10% concordaram totalmente e 60% concordaram fortemente. O teste *t* da amostra para esse construtor foi $p < 0.01$.

Com base nas respostas e, principalmente, no teste t, pode-se afirmar que a hipótese do usuário tem conhecimento das políticas de privacidade e uso da informação na RSO foi **refutada**.

Hipótese H3

sobre a Percepção de Privacidade Geral (construtor GPC), a avaliação da confiabilidade do questionário, via alfa de Cronbach, obteve resultado de 0.879, o que representa uma consistência interna boa. Já o coeficiente de correlação de Pearson mostrou que não ocorreu correlação do construtor com a idade, mesmo que em GPC3 (0.385) a correlação tenha sido fraca e em GPC1 (0.135), GPC2(0.144) e GPC4 (-0.082) a correlação tenha sido inaceitável.

Em relação as respostas, na questão GPC1, 30% discordaram totalmente, 0% discordaram fortemente, 0% discordaram parcialmente, 10% nem concordaram e nem discordaram, 30% concordou parcialmente, 20% concordaram totalmente e 10% concordaram fortemente. Na questão GPC2, 10% discordaram totalmente, 10% discordaram fortemente, 10% discordaram parcialmente, 10% nem concordaram e nem discordaram, 30% concordou parcialmente, 10% concordaram totalmente e 20% concordaram fortemente. Na questão GPC3, 0% discordaram totalmente, 10% discordaram fortemente, 10% discordaram parcialmente, 10% nem concordaram e nem discordaram, 20% concordou parcialmente, 0% concordaram totalmente e 50% concordaram fortemente. Na questão GPC4, 10% discordaram totalmente, 0% discordaram fortemente, 0% discordaram parcialmente, 20% nem concordaram e nem discordaram, 40% concordou parcialmente, 10% concordaram totalmente e 20% concordaram fortemente. O teste *t* da amostra para esse construtor foi $p < 0.01$.

Com base nas respostas e, principalmente, no teste t, pode-se afirmar que a

 © 2017 Sociedade Brasileira de Computação

hipótese de que o usuário se preocupa com a sua privacidade na RSO foi **refutada**.

Hipótese H4

Sobre a Privacidade Percebida (construtor PP), a avaliação da confiabilidade do questionário, via alfa de Cronbach, obteve resultado de 0.897, o que representa uma consistência interna boa. Já o coeficiente de correlação de Pearson mostrou que ocorreu uma correlação fraca entre a idade e a resposta das questões PP1(-0.444), PP2 (-0.382) e PP3 (-0.376) e uma correlação moderada entre a idade e a resposta da questão PP4 (-0.689). Ou seja, quanto menor a idade do participante maior é a sua privacidade percebida.

Em relação as respostas, na questão PP1, 10% discordaram totalmente, 0% discordaram fortemente, 10% discordaram parcialmente, 10% nem concordaram e nem discordaram, 50% concordou parcialmente, 10% concordaram totalmente e 10% concordaram fortemente. Na questão PP2, 30% discordaram totalmente, 10% discordaram fortemente, 20% discordaram parcialmente, 0% nem concordaram e nem discordaram, 20% concordou parcialmente, 10% concordaram totalmente e 10% concordaram fortemente. Na questão PP3, 30% discordaram totalmente, 10% discordaram fortemente, 20% discordaram parcialmente, 0% nem concordaram e nem discordaram, 10% concordou parcialmente, 20% concordaram totalmente, e 10% concordaram fortemente. Na questão PP4, 0% discordaram totalmente, 20% discordaram fortemente, 10% discordaram parcialmente, 10% nem concordaram e nem discordaram, 50% concordou parcialmente, 10% concordaram totalmente e 0% concordaram fortemente. O teste *t* da amostra para esse construtor foi $p < 0.01$.

Com base nas respostas e, principalmente, no teste t, pode-se afirmar que a hipótese de que o usuário acredita estar seguro quando usa a RSO foi **refutada**.

5.3.1. Discussão dos Resultados

Com a análise das respostas obtidas no teste piloto pode-se verificar que o questionário relativo a Eficácia Percebida das Políticas de Privacidade (construtor PEPP), empregado na hipótese **H2**, resultou em consistência interna inaceitável. Os questionários relativos ao Controle Comportamental Percebido (construtor PBC) e Uso da Informação, empregados, respectivamente, nas hipóteses **H1** e **H2**, resultaram em consistência interna duvidosa. Desta forma, caberá à trabalhos futuros o aperfeiçoamento das questão assertivas relativas a estes construtores.

Além disso, apesar dos participantes da amostra demonstrarem não acreditar que estão seguros na RSO, eles: (i) não conhecem ou sabem utilizar os mecanismos de controle de segurança e privacidade; (ii) não possuem conhecimento das políticas de privacidade e uso da informação; e (iii) não se preocupam com a sua privacidade na RSO. Para aceitar ou refutar as hipóteses, utilizamos o teste estatístico *t*, onde formula-se uma hipótese nula e consequentemente uma hipótese alternativa, e então calcula-se o valor de *t* conforme a fórmula apropriada ($1-$ p-valor) e aplica-se a função densidade de probabilidade da distribuição *t* de Student, medindo o tamanho da área abaixo dessa função para valores maiores ou iguais a *t*. Essa área representa a probabilidade da média dessas amostras em questão terem apresentado os valores observados ou algo mais externo. Se a probabilidade desse resultado ocorrer for muito pequena, conclui-se que o resultado ob-

 © 2017 Sociedade Brasileira de Computação

servado é estatisticamente relevante. Essa probabilidade também é chamada de p-valor ou valor de p. Consequentemente, o nível de confiança α é igual a $1-$ p-valor.

Sendo assim, como todas as hipóteses foram refutadas, é fácil perceber que os usuários não possuem preocupações com a sua privacidade nas RSOs, mesmo demonstrando que não acreditam estar seguros no ambiente das RSOs. Em outras palavras, os usuários mostram-se desacreditados quanto a segurança das RSOs, mas mesmo assim continuam a utilizá-las e pouco fazem para proteger a privacidade de seus dados, deixando-os consciente, ou inconscientemente, a disposição para possíveis incidentes de segurança.

6. Conclusão

Neste trabalho buscou-se compreender como os usuários de RSO percebem a sua privacidade. A partir de uma Revisão Bibliográfica identificou-se os principais construtores ligados a privacidade e, baseado neles, elaboraram-se quatro hipóteses de estudo. A avaliação das hipóteses foi feita através de um questionário, com resposta na escala *likert* de sete (07) pontos, e aplicada a um grupo de dez voluntários.

A análise das respostas refutou todas as hipóteses, o que permite concluir que os usuários de RSO da amostra pouco compreendem sobre privacidade. Por não conhecerem ou saberem utilizar os mecanismos de controle de privacidade, não estão aptos a gerenciar diretamente sua privacidade. Em linhas gerais, os usuários de RSO da amostra demonstraram não possuir quaisquer preocupações com a sua privacidade.

Referências

Benson, V., Saridakis, G., and Tennakoon, H. (2015). Information disclosure of social media users: Does control over personal information, user awareness and security notices matter? *Information Technology & People*, 28(3):426–441.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334.

Ellison, N. B. et al. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230.

Li, T. (2015). *The impact of users' awareness and self-efficacy of control options on disclosure intention in online social networks*. PhD thesis, University of British Columbia.

Milberg, S. J., Burke, S. J., Smith, H. J., and Kallman, E. A. (1995). Values, personal information privacy, and regulatory approaches. *Communications of the ACM*, 38(12):65–74.

Taneja, A., Vitrano, J., and Gengo, N. J. (2014). Rationality-based beliefs affecting individual's attitude and intention to use privacy controls on facebook: An empirical investigation. *Computers in Human Behavior*, 38:159–173.

Westin, A. F. (1968). Privacy and freedom. *Washington and Lee Law Review*, 25(1):166.

Wisniewski, P., Xu, H., Lipford, H., and Bello-Ogunu, E. (2015). Facebook apps and tagging: The trade-off between personal privacy and engaging with friends. *Journal of the Association for Information Science and Technology*, 66(9):1883–1896.

 © 2017 Sociedade Brasileira de Computação

Uma Análise Comparativa de Ferramentas de Esteganálise em Estego-Imagens com Texto Embutido

Lucas C. Possatti[1], **Leandro C. Resendo**[1], **Gilberto Sudré Filho**[1], **Jefferson O. Andrade**[1], **Karin S. Komati**[1]

[1] Coordenação de Informática – Instituto Federal do Espírito Santo (Ifes) Campus Serra
Rodovia ES-010 – Km 6,5 – Manguinhos – CEP: 91.501-970 – Serra – ES – Brazil

lucas.c.possatti@gmail.com, {leandro, gilberto.sudre}ifes.edu.br

{jefferson.andrade, kkomati}ifes.edu.br

Abstract. *This work presents a comparative study of the efficiency of the StegExpose tool. StegExpose implements the LSB steganalysis methods: Chi Square, RS Analysis, Primary Sets, and Sample Pair, as well as a proprietary fusion method. For each of the 30 images in the database, text was embedded using five steganography tools: Stepic, LSBSteg, Steghide, Outguess and F5. Also, the sizes of the secret message was iterated in increments of 10%. The experiments show that the performance of a steganalysis method depends on the steganography technique used. Therefore, the steganalysis tools detected data hidden in the stego-images generated by Stepic and LSBSteg, but did not detect stego-images generated by Steghide, Outguess and F5, which use different methods of LSB.*

Resumo. *Este trabalho faz um estudo comparativo de eficiência da ferramenta StegExpose, que reúne os métodos de esteganálise LSB: Chi Square, RS Analysis, Primary Sets, e Sample Pair, além de um método de fusão próprio. Para cada uma das 30 imagens da base de dados, embutiu-se texto usando cinco ferramentas de esteganografia, Stepic, LSBSteg, Steghide, Outguess e F5, e variando o tamanhos da mensagem secreta, em incrementos de 10%. Ao fim dos experimentos constatou-se que o desempenho de um método de esteganálise depende da técnica de esteganografia, assim as ferramentas de esteganálise detectaram dados escondidos nas estego-imagens geradas pelo Stepic e LSBSteg, porém não detectaram nas estego-imagens geradas pelo Steghide, Outguess e F5, que usam métodos diferentes do LSB.*

1. Introdução

A palavra "esteganografia" vem do grego: "*estegano*" significa esconder, e "grafia" significa escrita, isto é, a arte da escrita escondida [Julio et al. 2007]. O objetivo da esteganografia é prover um meio secreto de transmissão de dados.

Esteganografia, em um contexto atual, é a prática de esconder um arquivo digital dentro de outro. Um desses arquivos é o arquivo de cobertura (ou *cover-message* ou *carrier-medium*), enquanto o outro arquivo é uma mensagem secreta que se deseja esconder dentro do arquivo de cobertura, chamado de dado embutido ou *embedded data*. Após a inserção do dado embutido na mensagem de cobertura se obtém o estego-objeto

 © 2017 Sociedade Brasileira de Computação

ou esteganograma ou *stego-object* [Sujatha 2013]. Conforme mostrado na Figura 1, em que se gera uma estego-imagem, que ao passar por uma função de extração, retira-se a mensagem secreta.

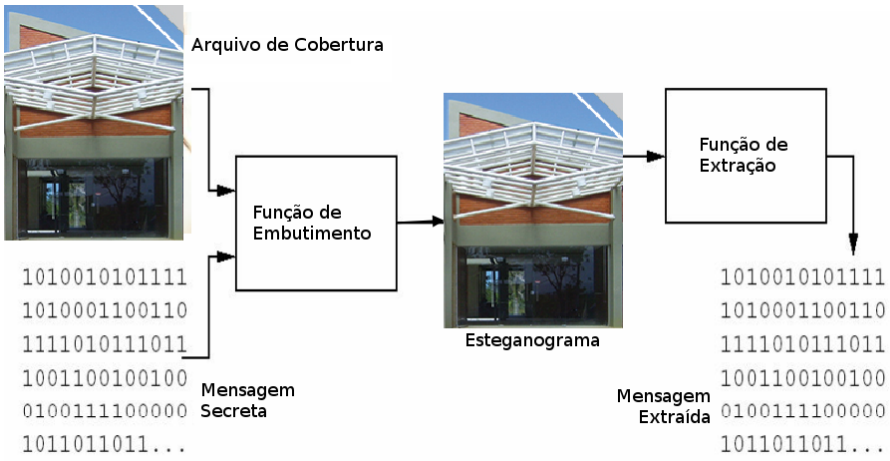


Figure 1. Arquitetura genérica de um sistema de esteganografia.

Técnicas esteganográficas podem ser usadas ilegitimamente para transmitir imagens de pornografia ou pedofilia. Um caso bem conhecido ocorreu em 2002, na ação policial "Operations Twins" que culminou na captura de criminosos associados ao grupo "Shadowz Brotherhood", uma organização de pedofilia que distribuía pornografia infantil usando esteganografia em imagens que pareciam inocentes [Mazurczyk et al. 2016], [Wendzel et al. 2014].

Para este casos ilícitos, existe a esteganálise, que é o estudo que busca detectar mensagens que foram escondidas através da esteganografia. A maioria dos algoritmos de esteganálise se limita a dizer se há ou não dados escondidos, pois em geral, recuperar os dados escondidos está além da capacidade da maioria das técnicas [Julio et al. 2007], dado que os bits estão espalhados pelo objeto de cobertura de forma desconhecida.

O objetivo deste trabalho é analisar a eficiência de algumas das ferramentas e métodos de esteganálise em estego-imagens com texto embutido. Os questionamentos que se pretende responder com o estudo deste trabalho são: Qual é o índice de acerto das técnicas de esteganálise em relação à imagens com texto embutido? Este índice de acerto varia com o tamanho do texto embutido? – Para responder à tais questionamentos, serão usadas ferramentas existentes de esteganografia e esteganálise.

2. Referencial Teórico

A esteganografia é uma área da Segurança da Informação, que lida com a ocultação de informações. O objetivo da esteganografia é modificar um arquivo portador de forma imperceptível [Westfeld 2001].

Técnicas de esteganografia que usam imagens como arquivo de cobertura podem atuar de duas formas possíveis: no domínio espacial ou no domínio de frequência. No domínio espacial, a técnica de esteganografia mais comum é o LSB (*Least Significant Bit*, literalmente, bit menos significativo) [Choudhary 2012]. E a manipulação no domínio

 © 2017 Sociedade Brasileira de Computação

da frequência é comum em arquivos JPEG, para o qual alguns algoritmos manipulam os coeficientes DCT (*Discrete Cosine Transform* ou em português Transformada de Cosseno Discreta) para esconder a mensagem secreta [Cheddad et al. 2010].

O LSB faz a troca dos bits menos significativos dos bytes que compõe a imagem, por bits do dado. O bit menos significativo é o que torna uma sequência de bits par ou impar, por exemplo, no byte 01101101, o bit menos significativo é o bit **1** que está na extrema direita. Ao alterar o LSB, o valor do byte muda para mais um ou menos um, o que não altera muito o valor do byte se comparado a mudança de qualquer um dos outros bits. Alterando apenas os LSBs, os valores de cada byte não têm uma alteração significativa, e a imagem não terá uma alteração perceptível ao olho nu. A forma mais simples utiliza apenas o último plano de bits e alteram os bits sequencialmente, as variações do algoritmo usam mais de um plano ou distribuem as alterações aleatoriamente pelos LSBs. Embora seja uma técnica simples é considerada de difícil detecção [Sujatha 2013], [Julio et al. 2007].

As técnicas de esteganografia baseadas em transformadas de frequência aproveitam-se da compressão do próprio formato do arquivo [Julio et al. 2007]. Assim, é comum o uso da DCT, que é utilizada nas compressões do formato de imagem JPEG. No formato de imagem JPEG, para cada componente de cor o formato usa uma DCT para transformar blocos de 8x8 píxeis em 64 coeficientes DCTs [Provos and Honeyman 2003].

Segundo [Fridrich et al. 2002], o formato JPEG atraiu bastante atenção para o ramo da esteganografia, pois ele é o formato mais comum para armazenar imagens e usado amplamente na internet. Métodos de esteganografia modernos usam o formato por oferecer uma boa capacidade esteganográfica sem necessariamente sacrificar a segurança. O algoritmo F5 (proposto por [Westfeld 2001] como um desafio à comunidade científica) é um bom exemplo disso.

2.1. Esteganálise

A esteganálise é uma área de pesquisa que busca criar métodos para detectar quando a esteganografia foi aplicada em um arquivo ou não. Existem dois tipos principais de esteganálise: visuais e estatísticos.

Nos ataques visuais, processa-se a imagem de forma que um humano consegue ver a imagem e detectar características que denunciam o uso de esteganografia. Quando disponíveis, ataques estatísticos são superiores aos visuais, pois são menos dependentes do arquivo original e podem ser completamente automatizados, o que permite processar vários itens em larga escala [Westfeld and Pfitzmann 1999].

De acordo com [Boehm 2014], os principais métodos de esteganálise estatísticos são: Chi Square [Westfeld and Pfitzmann 1999], RS Analysis [Fridrich et al. 2001a], Primary Sets [Dumitrescu et al. 2002] e Sample Pair [Dumitrescu et al. 2003]. Todos são métodos de análise contra o método LSB. Neste trabalho, usou-se uma ferramenta que reúne implementações destes quatro métodos, chamada StegExpose.

O Chi Square foi um dos primeiros métodos de esteganálise em embutimento LSB e detecta bem embutimento LSB sequencial, mas não é muito efetivo para a mensagem que está espalhada aleatoriamente no plano LSB. Já o método RS Analysis foi criado para detectar embutimento LSB espalhado pelo esteganograma [Fridrich et al. 2001b]. O

 © 2017 Sociedade Brasileira de Computação

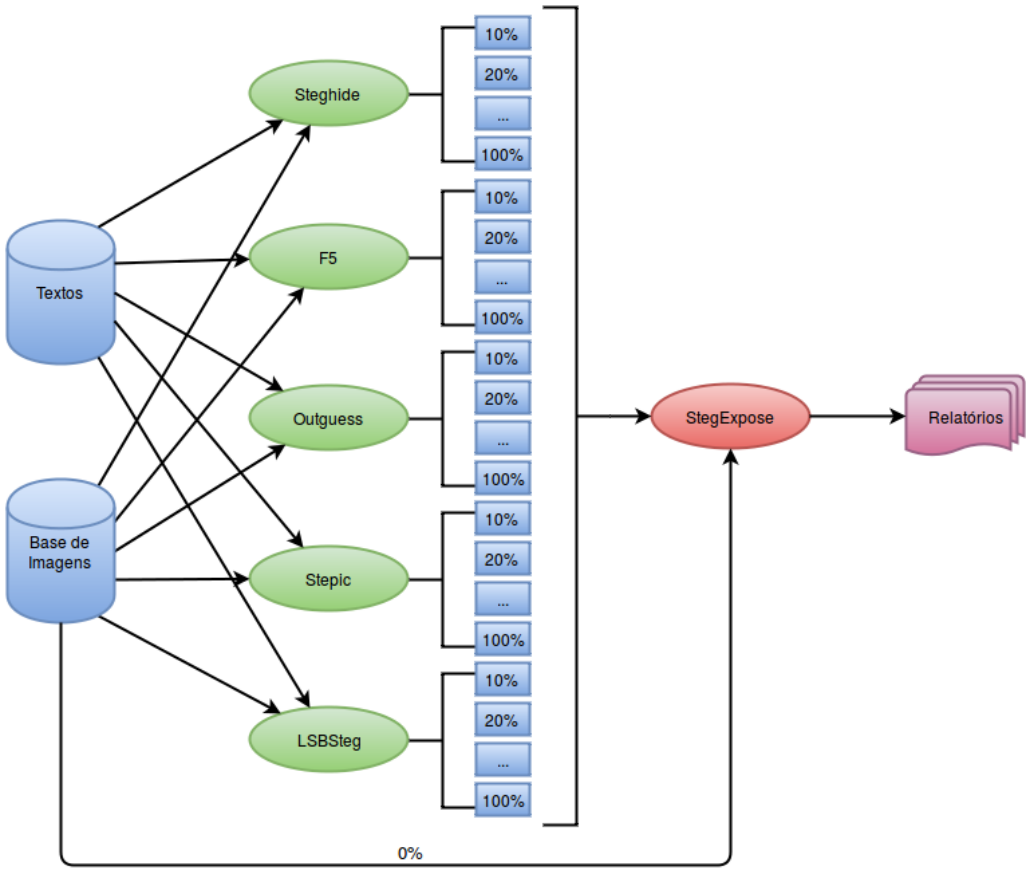


Figure 2. Processo dos experimentos

Primary Sets é uma melhoria do RS Steganalysis, bem como Sample Pair é uma melhoria do Primary Sets, que pode detectar esteganografia LSB não apenas em imagens, mas também em áudio, e outros sinais digitais.

3. Materiais e Métodos

A partir de um banco de imagens, embute-se mensagens secretas do tipo texto de tamanhos variados (entre 10% à 100%, variando de 10% em 10%) usando ferramentas de esteganografia: Stepic, LSBSteg, Steghide, Outguess e F5. Depois, é usada a ferramenta StegExpose para analisar cada uma das estego-imagens, lembrando que o StegExpose reune as técnicas: Chi Square, RS Analysis, Primary Sets, e Sample Pair. O StegExpose gera relatório detalhado para análise. Esse experimento tem o objetivo de avaliar a eficácia das ferramentas de esteganálise, o processo da aplicação dos sistemas de esteganografia e esteganálise é ilustrado na Figura 2.

Há centenas de ferramentas de esteganografia disponíveis, apenas na listagem da página do Dr. Neil F. Johnson [Johnson 2012], há 110 (cento e dez) ferramentas. Foram escolhidas três ferramentas, Outguess [Provos 2001], o Steghide [Hetzl 2002] e o F5 [Westfeld 2001], pois segundo o [Miche 2010] são algumas das ferramentas mais conceituadas para a esteganografia, nenhuma delas é baseada em LSB. Além destes, foram escolhidas mais duas ferramenta de esteganografia baseadas em LSB: o Stepic [Domnitser 2007] e o LSBSteg [David 2015].

 © 2017 Sociedade Brasileira de Computação

3.1. Base de Dados de Imagens e Texto

Construiu-se um script para baixar imagens da API do Pixabay [1]. Todas as imagens são de domínio público, licenciadas sob a *Creative Commons CC0*, e em sua grande maioria são fotografias. Para encontrar imagens no Pixabay usou-se nomes de países como termos de pesquisa, assim como foi feito também por [Boehm 2014]. Os nomes dos países usados foram: Rússia, Canada, China, United, Brazil, Australia, India, Argentina, Kazakhstan. Ao final, reduziu-se a seleção de imagens para trinta e todas as imagens foram normalizadas ou recortadas para a resolução de 400x400 píxeis.

O texto usado para este trabalho foi composto por 15 (quinze) livros do Project Gutenberg [2] de domínio público. Todos os arquivos são em formato de texto puro (.txt), codificados em UTF-8. Todos os livros foram compilados para um único arquivo (aproximadamente 8 MB) e, a partir deste arquivo único, extraía-se a quantidade exata de bytes para ser usada pelos programas de esteganografia.

3.2. Relatório do StegExpose

Todas as quatro técnicas de esteganálise utilizadas no StegExpose (Chi Square, RS Analysis, Primary Sets e Sample Pair) retornam um número real entre 0,0 e 1,0 (*score*) que reflete a probabilidade do arquivo conter conteúdo esteganográfico ou não. E somente após uma imagem receber seu *score*, ele compara este *score* com o limiar (ou *threshold*) fornecido pelo usuário para dar uma classificação binária da imagem (se ela contém uma mensagem secreta ou não).

De acordo com [Boehm 2014], o valor de *threshold* que apresentou os melhor resultado foi de 0,2, sendo o valor padrão (*default*) do StegExpose. Assim, adotando-se o valor padrão, se o resultado da técnica de esteganálise for maior ou igual a 0,2, a resposta final é "*true*" que a imagem analisada possui dados ocultos, ou "*true*" caso contrário.

Assim, o resultado final é um classificador binário, que será avaliado pela sua acurácia, pela curva ROC (*Receiver Operating Characteristics*) e pela AUC (*Area Under the Curve*) [Fawcett 2006]. O AUC é um valor que expressa o desempenho do classificador, e é medido pela área abaixo da respectiva curva. Quanto maior esse valor, melhor é o desempenho do classificador.

A fusão usada no StegExpose usa a média aritmética dos retornos dos quatro detectores individuais, e de acordo com [Boehm 2014] esta fusão apresentou maior precisão nos experimentos de seu trabalho. A fusão padrão se mostrou mais precisa do que qualquer um dos detectores individuais.

4. Resultados e Discussão

Esta seção analisa os dados sob dois aspectos: de acordo com as ferramentas de esteganografia e no desempenho dos métodos de esteganálise contra cada uma das ferramentas de esteganografia.

4.1. Testes por Ferramenta de Esteganografia

O StegExpose gera um relatório com cinco *scores* diferentes, mas nesta seção usam-se apenas o *score* da fusão. A Tabela 1 apresenta os resultados dos experimentos, onde cada

[1] `pixabay.com`

[2] `http://www.gutenberg.org/wiki/PT_Principal`

 © 2017 Sociedade Brasileira de Computação

coluna apresenta a ferramenta de esteganografia utilizada para gerar a estego-imagem e cada linha apresenta o percentual de informação escondida. Cada célula da tabela corresponde à quantidade de imagens detectadas de um total de 30 imagens, exceto para a célula do resultado do Outguess com 100% (indicado com o *)[3]. Por exemplo, a leitura deve ser da seguinte forma: a ferramenta StegExpose indicou que lê-se que 25 das 30 imagens com 40% de texto oculto geradas pela ferramenta Stepic foram detectadas.

	F5	**Steghide**	**Outguess**	**Stepic**	**LSBSteg**
10%	5	7	6	7	8
20%	4	7	8	13	16
30%	6	7	6	20	24
40%	6	9	6	25	29
50%	4	8	7	28	29
60%	3	9	8	29	30
70%	3	8	8	29	30
80%	3	8	6	30	30
90%	2	9	7	30	30
100%	2	8	3*	30	30

Table 1. Número de imagens detectadas para cada ferramenta de esteganografia e porcentagem de embutimento para limiar de 0,1.

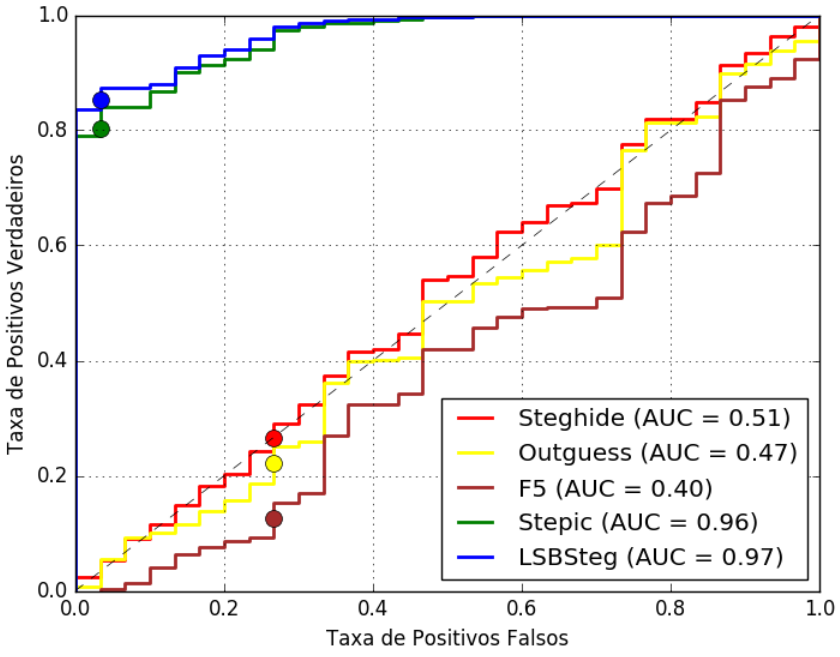


Figure 3. Curvas ROC para o desempenho do StegExpose contra cada um dos programas usados. Com pontos destacando o limiar de 0,1.

Podemos também notar que as estego-imagens produzidas pelo LSBSteg foram mais facilmente detectadas, mas seguido de perto pelo Stepic. Observa-se também que

[3]A execução do Outguess para 100% falhou para algumas imagens, das 30 (trinta) imagens, 6 (seis) apresentaram mensagem de erro, assim, foram 24 estego-imagens testadas pelo Outguess

© 2017 Sociedade Brasileira de Computação

a partir das estego-imagens com 60% de embutimento é que o StegExpose foi capaz de detectar corretamente as imagens.

Nota-se que apenas os programas de esteganografia LSB (Stepic e LSBSteg) foram realmente detectados. Isso já era esperado, pois o StegExpose foi feito para detectar especificamente esteganografia LSB. Podemos ver que o StegExpose não pode detectar consistentemente nenhuma estego-imagem gerada pelo Outguess, F5 e Steghide.

Pode-se perceber no gráfico ROC (Figura 3) que o desempenho do StegExpose contra o F5, Steghide e Outguess é completamente aleatório, dado que suas curvas estão com ângulo de 45º e que apresenta um bom desempenho contra o LSBSteg e o Stepic. Quanto maior o valor de AUC, melhor é o desempenho da ferramenta.

A curva ROC é uma função escada, pois a avaliação foi feita para valores discretos de limiares. O melhor resultado é obtido para o *threshold* de 0,1, que são os pontos marcados na Figura 3. Os resultados são piores para o valor de limiar padrão sugerido (0,2).

4.2. Testes por Método de Esteganálise

Nesta seção, analisa-se o desempenho de cada um dos métodos de esteganálise individualmente contra o Stepic e LSBSteg. Foram considerados apenas os resultados das estego-imagens destas duas ferramentas dada a análise da seção anterior. Nas Figuras 4 e 5, apresentam-se o desempenho dos métodos (gráficos ROC) contra o Stepic e LSBSteg respectivamente.

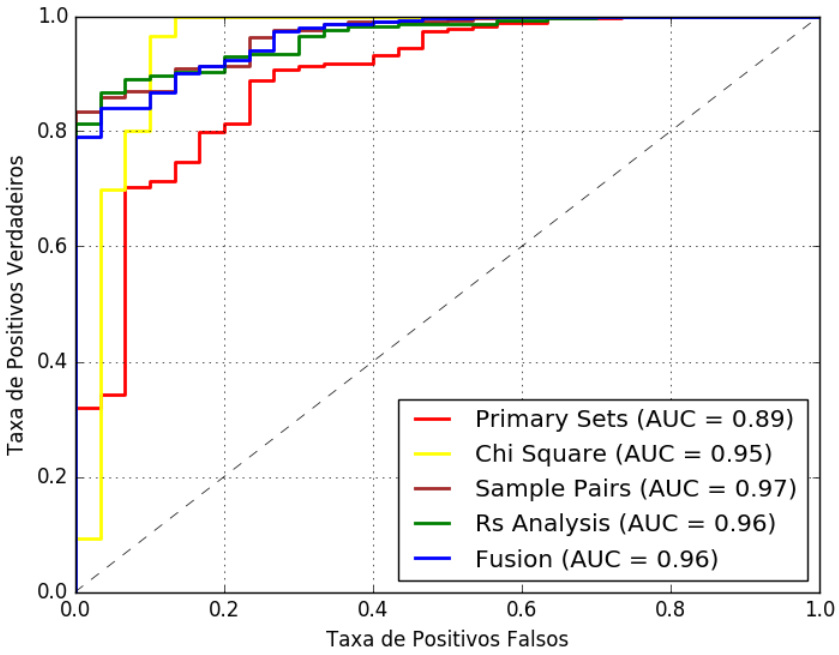


Figure 4. Curvas ROC para o desempenho dos métodos de esteganálise contra o Stepic.

Com o método de esteganografia do Stepic, o melhor resultado, AUC de 0.97, foi o Sample Pairs, seguido do método RS e Fusion, ambos com AUC de 0.96. O pior desempenho foi o Primary Sets, com AUC de 0.89.

© 2017 Sociedade Brasileira de Computação

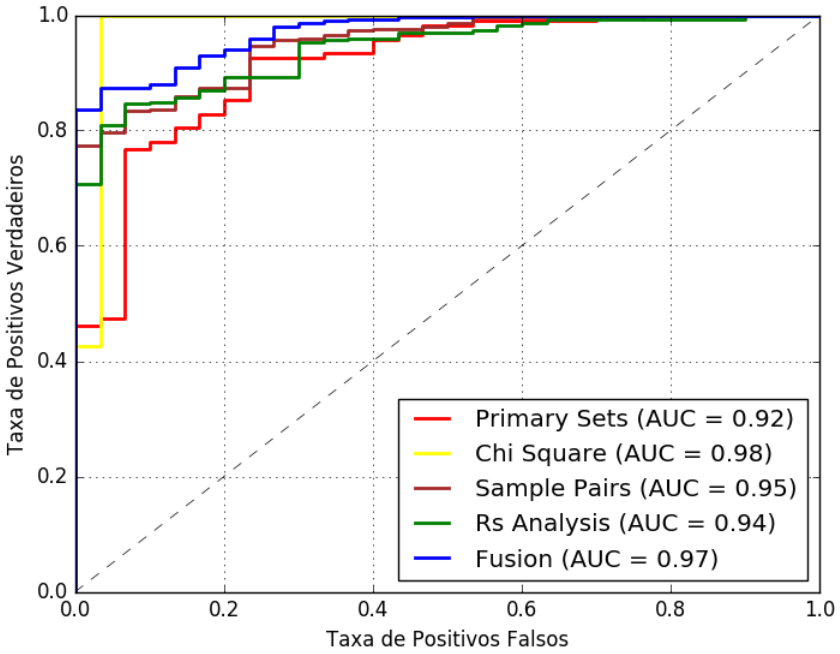


Figure 5. Curvas ROC para o desempenho dos métodos de esteganálise contra o LSBSteg.

Para o método de esteganografia do LSBSteg, o melhor resultado, AUC de 0.98, foi o Chi Square, seguido do método do Fusion com AUC de 0,97. O pior desempenho foi o Primary Sets, novamente.

O Chi Square, de acordo com a literatura, realmente apresenta bom desempenho contra programas de esteganografia sequencial. Mas dificilmente detecta esteganografia LSB espalhada aleatoriamente pelo esteganograma, exceto quanto embutimos mensagens secretas muito grandes e o esteganograma fica saturado. E veja que tanto o Stepic quanto o LSBSteg são programas de esteganografia LSB sequencial. Se tivéssemos usado algum programa de esteganografia LSB aleatório, os resultados para o Chi Square nesta seção provavelmente seriam diferentes. Essa questão é bem discutida em [Kharrazi et al. 2006], no qual algoritmos específicos de esteganálise são desenvolvidos para métodos específicos de esteganografia.

Os resultados deste trabalho são diferentes dos resultados do trabalho de [Boehm 2014], que mostra que o método de fusão do StegExpose é sempre melhor. Um dos motivos é que as bases de dados de imagens e de texto são diferentes e as ferramentas de esteganografia são diferentes.

5. Considerações Finais

Para atingir o objetivo deste trabalho foi necessário construir um banco de imagens e texto; pesquisar ferramentas de esteganografia e esteganálise; desenvolver um *script* automático de testes. Ao todo foram produzidas 1.500 (um mil e quinhentas) estego-imagens, usou-se a ferramenta StegExpose para analisar cada uma das imagens estego-imagens.

Observou-se que o StegExpose detectou um número razoável de estego-imagens

 © 2017 Sociedade Brasileira de Computação

produzidas pelo Stepic e LSBSteg, mas não detectou praticamente estego-imagem alguma produzida pelo Steghide, Outguess e F5. Algo que era esperado, pois o StegExpose é uma ferramenta de esteganálise específica para detectar esteganografia LSB.

O desempenho do StegExpose contra o Stepic e o LSBSteg foi melhor usando um *threshold* de 0,1, diferente do valor padrão recomendado pelos autores da ferramenta (0,2). Não foi possível indicar uma única ferramenta que tenha tido sempre o melhor desempenho, e sim, verificar que o desempenho de uma ferramenta de esteganálise depende do método de esteganografia usado. Este resultado está de acordo com a bibliografia, dado que foram utilizadas técnicas de esteganálise específicas e não genéricas (que tentam detectar informação oculta sem considerar técnica de esteganografia usada).

Como trabalhos futuros, seria interessante repetir os testes com um banco de imagens maior e mais variado, usando imagens com diferentes espaços de cores, resoluções e formatos. Além disso, aplicar esteganografia em outros tipos de formatos como, áudio, vídeo e texto, explorando diferentes tipos de mídia. Poderia-se usar outras programas e métodos de esteganálise e fazer um estudo comparativo dos mesmos, bem como Utilizar outras ferramentas de esteganografia. Também sera interessante avaliar os resultado das ferramentas de esteganálise em imagens que sofreram manipulação de sites de redes sociais, como o Facebook que altera o formato e resolução das imagens postadas.

E pesquisar correlações entre características estatísticas globais de uma imagem com a sua capacidade de ocultar mensagens secretas. Por exemplo, uma imagem muito simples que contém apenas uma única cor possui capacidade praticamente zero para ocultar mensagens, por outro lado, uma imagem com muitos objetos num ambiente com texturas variadas como grama ou areia podem esconder mensagens. Assim, dado uma imagem, seria possível estimar o tamanho de mensagem a ser escondida.

References

[Boehm 2014] Boehm, B. (2014). Stegexpose-a tool for detecting lsb steganography. *arXiv preprint arXiv:1410.6656*.

[Cheddad et al. 2010] Cheddad, A., Condell, J., Curran, K., and Mc Kevitt, P. (2010). Digital image steganography: Survey and analysis of current methods. *Signal processing*, 90(3):727–752.

[Choudhary 2012] Choudhary, K. (2012). Image steganography and global terrorism. *International Journal of Scientific & Engineering Research*, 3(4):12.

[David 2015] David, R. (2015). Lsb-steganography. Disponível em: `https://github.com/RobinDavid/LSB-Steganography`.

[Domnitser 2007] Domnitser, L. (2007). Spy vs. spy. Disponível em: `http://domnit.org/stepic/doc/`.

[Dumitrescu et al. 2002] Dumitrescu, S., Wu, X., and Memon, N. (2002). On steganalysis of random lsb embedding in continuous-tone images. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages 641–644. IEEE.

[Dumitrescu et al. 2003] Dumitrescu, S., Wu, X., and Wang, Z. (2003). Detection of lsb steganography via sample pair analysis. *IEEE transactions on Signal Processing*, 51(7):1995–2007.

 © 2017 Sociedade Brasileira de Computação

[Fawcett 2006] Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

[Fridrich et al. 2001a] Fridrich, J., Goljan, M., and Du, R. (2001a). Detecting lsb steganography in color, and gray-scale images. *IEEE multimedia*, 8(4):22–28.

[Fridrich et al. 2001b] Fridrich, J., Goljan, M., and Du, R. (2001b). Reliable detection of lsb steganography in color and grayscale images. In *Proceedings of the 2001 workshop on Multimedia and security: new challenges*, pages 27–30. ACM.

[Fridrich et al. 2002] Fridrich, J., Goljan, M., and Hogea, D. (2002). Steganalysis of jpeg images: Breaking the f5 algorithm. In *International Workshop on Information Hiding*, pages 310–323. Springer.

[Hetzl 2002] Hetzl, S. (2002). Steghide. Disponível em: `http://steghide.sourceforge.net/index.php`.

[Johnson 2012] Johnson, N. F. (2012). Steganography software. Disponível em: `http://www.jjtc.com/Steganography/tools.html`.

[Julio et al. 2007] Julio, E. P., Brazil, W. G., and Neves, C. V. (2007). Esteganografia e suas aplicações. *Livro de minicursos do SBSEG*, pages 54–102.

[Kharrazi et al. 2006] Kharrazi, M., Sencar, H. T., and Memon, N. (2006). Improving steganalysis by fusion techniques: A case study with image steganography. In *Transactions on Data Hiding and Multimedia Security I*, pages 123–137. Springer.

[Mazurczyk et al. 2016] Mazurczyk, W., Wendzel, S., Zander, S., Houmansadr, A., and Szczypiorski, K. (2016). *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures*. IEEE Press Series on Information and Communication Networks Security. Wiley.

[Miche 2010] Miche, Y. (2010). *Developing fast machine learning techniques with applications to steganalysis problems*. PhD thesis, Institut National Polytechnique de Grenoble-INPG.

[Provos 2001] Provos, N. (2001). Defending against statistical steganalysis. In *Usenix security symposium*, volume 10, pages 323–336.

[Provos and Honeyman 2003] Provos, N. and Honeyman, P. (2003). Hide and seek: An introduction to steganography. *IEEE security & privacy*, 99(3):32–44.

[Sujatha 2013] Sujatha, P. (2013). *Image steganalysis using artificial neural networks*. PhD thesis, Department of Computer Science, School of Computing Sciences Pallavaram, Chennai, Vels University.

[Wendzel et al. 2014] Wendzel, S., Mazurczyk, W., Caviglione, L., and Meier, M. (2014). Hidden and uncontrolled–on the emergence of network steganographic threats. In *ISSE 2014 Securing Electronic Business Processes*, pages 123–133. Springer.

[Westfeld 2001] Westfeld, A. (2001). F5—a steganographic algorithm. In *International workshop on information hiding*, volume 2137, pages 289–302. Springer.

[Westfeld and Pfitzmann 1999] Westfeld, A. and Pfitzmann, A. (1999). Attacks on steganographic systems. In *International workshop on information hiding*, pages 61–76. Springer.

 © 2017 Sociedade Brasileira de Computação

Uma Proposta de Arquitetura para Autorização Federada com Internet das Coisas

Carlos Eduardo da Silva[1], **Gabriela Cavalcante da Silva**[1]

[1]Instituto Metrópole Digital - Universidade Federal do Rio Grande do Norte (UFRN)
Avenida Senador Salgado Filho, 3000, Lagoa Nova – CEP: 59.078-970 – Natal/RN

gabicavalcantesilva@gmail.com, kaduardo@imd.ufrn.br

Abstract. *Internet of Things (IoT) has been applied to several application domains, including as mechanisms for physical access control. However, existing solutions do not take into account appropriate protocols for the process of authentication and authorization. In this context, this paper presents a proposal of architecture for physical access control based on IoT and integrated with mechanism for federated authentication and authorization based on the RBAC model.*

Resumo. *A Internet das coisas (Internet of Things - IoT) vem sendo utilizada em diversos domínios de aplicação, incluindo como mecanismos para controle de acesso físico. Entretanto, as soluções existentes não consideram o uso de protocolos apropriados no processo de autenticação e autorização. Neste contexto, este artigo apresenta uma proposta de arquitetura para controle de acesso físico baseado em IoT e integrada a mecanismos de autenticação federada e autorização baseada no modelo RBAC.*

1. Introdução

Existem diversas definições para o que seria Internet das Coisas (*Internet of Things* - IoT) [Minerva et al. 2015], que apresentam como características em comum a existência de dispositivos e sistemas que interligam sensores e atuadores do mundo real à Internet. Neste contexto, IoT pode ser enxergada sob a perspectiva de sistema ciber-físicos (*Cyber-Physical Systems* - CPS). Os CPSs são compostos por um conjunto de agentes em rede, incluindo sensores, atuadores, unidades de processamento e controle e dispositivos de comunicação [Cardenas et al. 2008], que executam embarcados em dispositivos ou em ambiente centralizado, como uma nuvem computacional.

Dentre as diversas áreas de aplicação para IoT (por exemplo, carros conectados [Evans-Pughe 2005], dispositivos portáteis, dispositivos de monitoramento de saúde, bem estar e *fitness* [Islam et al. 2015], sensores de rede sem fio [Lewis 2004], entre outros) é possível encontrar seu uso para controle de acesso físico. Neste contexto, é possível identificar soluções[1] que visam o uso de IoT para controle de acesso físico, como a solução formada por um Arduino e um leitor RFID para controlar portas[2]. Entretanto, baseado em informações públicas, essas soluções não consideram o uso de protocolos seguros nos processo de autenticação e autorização, se baseando no identificador do cartão RFID e na

[1]https://kintronics.com/solutions/ip-door-access-control/
[2]http://www.instructables.com/id/Arduino-RFID-Door-Lock/

 © 2017 Sociedade Brasileira de Computação

utilização de ACLs (*Access Control Lists* - listas de controle de acesso) para a definição de políticas de acesso.

Neste contexto, nosso objetivo é abordar a utilização da IoT para modelarmos um *Cyber-Physical System* que forneça controle de acesso físico através de uma autenticação federada. Mais especificamente, este artigo apresenta uma proposta de arquitetura para permitir integração entre federação de identidades, controle de acesso e Internet das Coisas (IoT - Internet of Things). Com isso, um usuário poderá transpor suas credenciais de uma federação de identidades (como a federação CAFe da RNP) para o mundo físico. Como estudo de caso, iremos desenvolver um protótipo de sistema de controle de acesso físico baseado em IoT.

Este artigo está organizado da seguinte forma: a seção 2 apresenta uma breve introdução sobre autorização, seguida de algumas soluções existentes que exploram IoT para controle de acesso. Na seção 3 temos a nossa proposta de arquitetura, e por fim, concluímos na seção 5.

2. Contextualização e Trabalhos Relacionados

O modelo de identidades federadas [Chadwick 2009] permite resolver o problema de múltiplas credenciais de acesso. Neste modelo, cada usuário só precisa gerenciar uma única credencial de acesso e essa o permite acessar diferentes provedores de serviço. Porém, esse acesso só é liberado após a autorização ser dada pelas políticas de segurança de cada provedor de serviço. Estes procedimentos constituem mecanismos que permitem a realização de controle de acesso.

Existem diversos modelos para a especificação de políticas de controle de acesso. Um dos modelos mais difundidos é o RBAC/ABAC, em que um é baseado em papéis (RBAC - *Role based access control* [Sandhu et al. 1996]) e outro em atributos (ABAC - *Attribute Based Access Control* [Hu et al. 2014b]). Nestes modelos, permissões são associadas a papés/atributos que são atribuídos aos usuários, de forma que um acesso será concedido caso o usuário requisitante tenha o papel/atributo especificado na política. O modelo ABAC se apresenta como uma evolução do RBAC e visa herdar as melhores práticas de modelos anteriores e tratar atributos variáveis e específicos de um ambiente, ao invés de considerar o papel como único atributo.

Um mecanismo de controle de acesso, como os baseados nos modelos RBAC/ABAC, seguem uma arquitetura funcional similar, na qual pode-se identificar um conjunto de componentes. A Figura 1 apresenta esses componentes e a interação entre eles. Quando um indivíduo solicita uma operação em um determinado objeto, essa solicitação é interceptada pelo *Policy Enforcement Point* - **PEP**. O PEP tem o papel de proteger o objeto e de fazer cumprir as decisões de controle de acesso, permitindo ou não o acesso do sujeito (*subject*) aos recursos (*object*). Para isso, o PEP interage com o **PDP** (*Policy Decision Point*). O PDP é o componente responsável por tomar uma decisão permitindo ou não o acesso requisitado. Essas decisão são baseadas nas políticas de controle de acesso, armazenadas em um repositório (*Policy Repository*). O PDP utiliza também o *Policy Information Point* - **PIP** para obter informações que o auxiliarão na tomada de decisão, tais como atributos do sujeito e/ou objeto (a partir do *Attribute Repository*) e condições do ambiente (*Environment Conditions*). Uma vez de posse das informações necessárias, o PDP toma uma decisão e informa a mesma para o PEP, que por sua vez, permite ou não

 © 2017 Sociedade Brasileira de Computação

o acesso ao objeto. Políticas de controle de acesso são administradas através do *Policy Administration Point* - **PAP**.

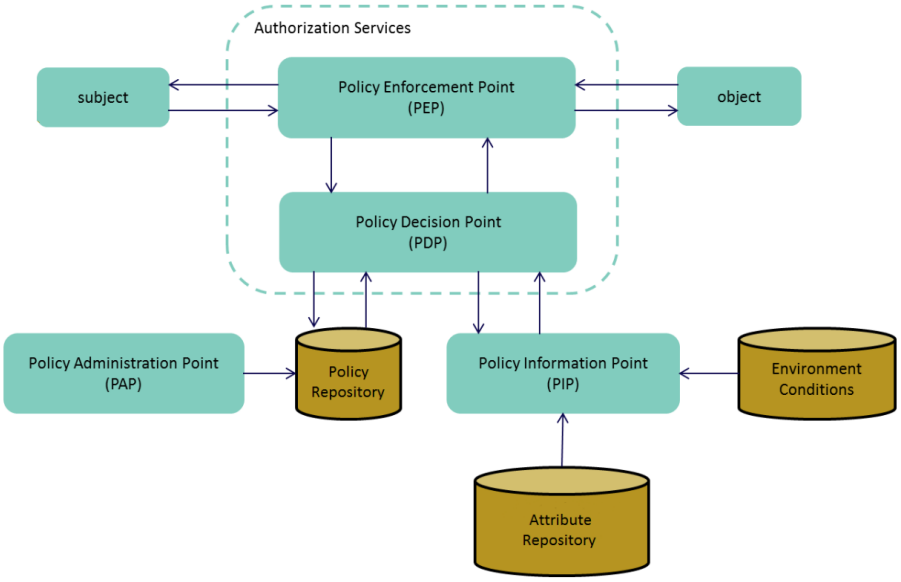


Figura 1. Exemplo de Pontos Funcionais de Autorização [Hu et al. 2014a]

Nos voltando para o contexto de IoT, encontramos algumas soluções que propõem desde arquitetura até topologias com IoT para autorização e controle de acesso. Uma dessas soluções pode ser encontrada em [Fremantle et al. 2014], que traz como principal contribuição verificar se seria viável e eficaz usar OAuth2.0 [Hardt 2012] como parte do fluxo do protocolo *Message Queue Telemetry Transport* (MQTT)[3] e também dentro de um MQTT *broker* para tomadas decisões de controle de acesso federadas e direcionadas pelo usuário. Outra solução considerada foi [Liu et al. 2012]. Neste trabalho, os autores propõem uma arquitetura abstrata para autenticação de usuários de Internet das Coisas, identificando uma série de componentes como um gateway para coisas, que também atua como um provedor de serviço usando OpenID Connect para autenticar usuários. Eles também apresentam um método de autenticação e controle de acesso baseado nos componentes definidos.

Em [Ndibanje et al. 2014] é feita uma análise sobre autenticação e controle de acesso do método usando IoT apresentado em [Liu et al. 2012]. Segundo os autores, o protocolo apresentado é custoso na troca de mensagens e a avaliação de segurança não é suficientemente forte. A primeira crítica levantada por [Ndibanje et al. 2014], foi sobre a falta de detalhes claros sobre todo o processo de autenticação em relação às mensagens trocadas. Em seguida, ele destaca a não separação entre as principais etapas conhecidas de um processo de autenticação normal, como a fase de registro e a fase de login. E questiona a contribuição para o aspecto de controle de acesso, [Liu et al. 2012] não possui uma proposta de esquema e não descreve efetivamente como o RBAC seria usado no projeto.

Este levantamento preliminar mostrou que a utilização de IoT em cenário de controle de acesso físico está começando a receber atenção da comunidade, com diversas

[3]http://mqtt.org/documentation

 © 2017 Sociedade Brasileira de Computação

soluções em desenvolvimento. Entretanto, essas soluções não consideram questões relacionadas a múltiplos fatores de autenticação, ou a utilização de protocolos de autenticação e autorização adequados. Algumas abordagens têm atacado este problema com foco em OpenID Connect, ou com SAML [Domenech et al. 2016, Domenech 2015], onde foi explorado a utilização de um cliente "ativo" que permita com que coisas possam realizar a autenticação em uma federação de identidades. Entretanto, notamos uma ausência de trabalhos que explorem o perfil ECP definido pela especificação SAML para permitir a autenticação federada em ambientes não-Web.

3. Proposta de Arquitetura

Apesar da falta de clareza ou fragilidade encontrada em parte das soluções analisadas [Ndibanje et al. 2014], o estudo delas foi importante para podermos modelar nossa proposta de solução IoT levando em consideração o que poderíamos ou não incorporar, assim como o que seria possível trazer de novo. O resultado pode ser visto abaixo, na Figura 2.

Nossa arquitetura é composta por um *Controlador Físico*, atuando como um *gateway* para os dispositivos físicos (em nosso exemplo um leitor e um mecanismo de trava de uma porta). Esse controlador físico por sua vez é conectado a um *Controlador Lógico*, que representa um componente de software capaz de armazenar e processar os dados produzidos pelas "coisas" (representado pelo módulo *CORE*).

No tocante a uma federação de identidades, todo o conjunto dispositivos físicos, controladores físico e lógico constituem um provedor de serviço (*Service Provider*). Como o *Controlador Lógico* apresenta maior poder computacional, ele é responsável por implementar os protocolos (representado pelo móduglo *SAML SP*) exigidos pela federação. Desse modo, o *Controlador Lógico* é responsável por interagir com um provedor de identidades (*Identity Provider* - IdP).

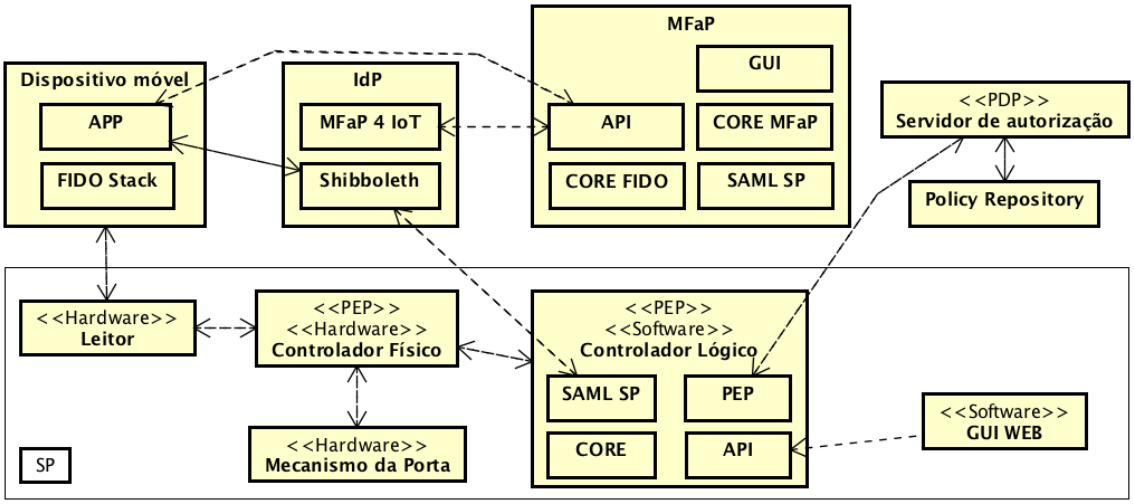


Figura 2. ***Diagrama de Componente do Protótipo IoT com AuthN/AuthZ em federação***

O IdP é o componente responsável pela autenticação do usuário, ele vai interagir com outro componente do nosso protótipo, o *Multi-Factor Provider* (MFaP), para fazer a

 © 2017 Sociedade Brasileira de Computação

autenticação do usuário com o segundo fator. O MFaP é responsável por implementar um serviço FIDO (*FIDO UAF Server* representado pelo módulo *CORE FIDO*) que é acessado através de uma API REST. Desse modo, o IdP delega ao MFaP o processo de autenticação via FIDO através do módulo *MFaP 4 IoT* (um cliente para a API oferecida pelo MFaP). Adicionalmente, o MFaP também oferece um serviço de registro dos dispositivos móveis (módulo *CORE MFaP*) atuando como um SP dentro da federação que é acessado através de uma interface Web (módulgo *GUI*). O acesso à interface Web do MFaP se dá através do fluxo de autenticação normal via SAML (módulo *SAML SP*).

O dispositivo móvel do usuário deve possuir suporte ao FIDO (representado pelo módulo *FIDO Stack*) e contem um aplicativo desenvolvido por nós. O *FIDO Stack* é um componente de software embarcado no telefone móvel durante seu processo de fabricação, isto é, de acordo com a especificação [Machani et al. 2014], todo dispositivo FIDO UAF deverá passar por um processo de certificação promovido pela *FIDO Alliance*. Para o desenvolvimento desse protótipo, como será feito uso de telefone não certificado pelo FIDO Alliance, optou-se por fazer uso do aplicativo *Dummy FIDO UAF Client*[4] não certificado pelo FIDO, mas que implementa o protocolo FIDO UAF.

O aplicativo Android permite ao usuário cadastrar o dispositivo como autenticador FIDO junto ao MFaP, além de atuar como cliente para a solução de controle de acesso físico, interagindo com o *Leitor* e permitindo ao usuário se autenticar através do FIDO.

No que diz respeito a controle de acesso, as políticas de autorização, assim como os mecanismos de suporte são responsabilidade do PDP (*Policy Decision Point*). O *Controlador Físico* e *Controlador Lógico* fazem o papel de PEP (*Policy Enforcement Point*), sendo responsáveis por executar as decisões do PDP. Em nosso cenário, consideramos o uso de políticas XACML [OASIS 2010] baseadas no modelo RBAC/A-BAC [Hu et al. 2014b]. Além disso, está sendo considerado o padrão FIDO UAF [Machani et al. 2014] como único fator de autenticação para o cenário com IoT.

Por uma questão de espaço processo, não detalharemos o processo de registro com diagramas, nos limitaremos a dar uma breve descrição. O processo de registro de um dispositivo FIDO UAF (no nosso exemplo é o telefone Android) se inicia com o usuário acessando o MFaP para habilitar o FIDO UAF como fator de autenticação. Como o MFaP é um provedor de serviço, temos então um fluxo de autenticação tradicional SAML. Nessa etapa, será possível realizar autenticação do usuário através do envio das credenciais via IdP. O MFaP então gera um QRCode que deverá ser lido pelo App no dispositivo móvel. Como o MFaP implementa FIDO UAF Server, ele possui um protocolo próprio de desafio e resposta para registrar uma chave pública que será gerada no dispositivo do usuário. De acordo com o protocolo FIDO UAF, o usuário precisa se autenticar localmente em seu dispositivo como parte do fluxo para geração do par de chaves. Se a autenticação do usuário em seu dispositivo ocorreu com sucesso, então o aplicativo mobile envia a resposta do desafio para o MFaP, que valida a resposta, salva em banco a chave pública enviada pelo aplicativo e envia para ele uma mensagem de sucesso.

Uma vez registrado o dispositivo, consideramos um cenário onde não seria desejado ou possível o usuário se autenticar com seu nome de usuário e senha como primeiro fator, como por exemplo, em sistemas de controle de acesso físico a ambientes. Sendo

[4] `https://github.com/emersonmello/dummyuafclient`

 © 2017 Sociedade Brasileira de Computação

assim, o usuário, em posse de um dispositivo com FIDO UAF registrado previamente em seu IdP, faria uso desse como o único fator de autenticação. O fato do usuário ter que passar pela autenticação biométrica local em um dispositivo FIDO UAF certificado, garantia aqui a confiança de que o usuário é realmente quem diz ser. A Figura 3 apresenta um exemplo de autenticação de acordo com este cenário.

Em nosso cenário de exemplo, representado no diagrama de atividade na Figura 3, um usuário requisita abrir uma porta (atividade 1) aproximando seu dispositivo móvel a um leitor (por exemplo, via NFC ou *bluetooth*). O leitor então notifica o Controlador Físico da proximidade do dispositivo, e ele por sua vez requisita autenticação ao Controlador Lógico, que requisita o APP Cliente (atividades 2-4). O APP solicita o FIDO AuthN Request ao Servidor FIDO (atividade 5). Como resultado desta solicitação, o Servidor FIDO gera o desafio e o envia para o App Cliente (atividade 6). O usuário responde o desafio e envia de volta ao Servidor FIDO (atividade 7). Caso a resposta seja válida (atividade 8), o APP Cliente solicita um SAML Request ao Controlador Lógico (atividade 10), que gera e envia de volta o SAML Request para o App Cliente. O aplicativo manda este SAML Request juntamente com o ID do usuário e o FIDO AuthN Responde para o IdP ECP (atividade 12).

Quando o IdP recebe a mensagem vinda do dispositivo móvel, ele valida o AuthN Response junto ao Servidor FIDO (atividades 13-14), e então gera e envia um SAML Response para o App Cliente (atividade 15), que encaminha para o Controlador Lógico (atividade 16). O Controlador Lógico valida a mensagem (atividade 17) e disponibiliza e realiza o processo de autorização com base nos atributos do usuário (atividade 18). Por fim, a porta pode ser destravada (atividade 19).

4. Proposta de Implementação

Na arquitetura do SP para o cenário de controle de acesso físico de ambientes, projetamos dois protótipos pensadas para a solução com Bluetooth e com NFC. Na Figura 4 vemos a solução NFC, envolvendo Raspberry Pi, módulo Leitor RFID-RC522, e a trava magnética para porta como componentes de Hardware. Para a solução Bluetooth, o protótipo é similar, temos somente a substituição do módulo Leitor RFID-RC522 pelo USB Bluetooth Adapter.

A interface GUI provida pelo *Controlador Lógico* é implementada usando *framework* Flask[5]. No *Controlador Lógico* teremos o Apache httpd com módulo mod_shib, para que possa assim atuar como um provedor de serviço (SP) Shibboleth e permitir a interação com IdPs da federação.

Para o SP solicitar a autenticação do usuário com mais de um fator ao IDP, é necessário utilizar o MFA Profile [Refeds 2017]. O SP utilizará o elemento <*RequestedAuthnContext*> na mensagem SAML para informa ao IdP como esse deverá autenticar seus usuários.

Por parte do IdP é necessário criar um novo fluxo de autenticação dentro de um contexto de configuração MFA. A comunicação entre IdP e FIDO exige que a API REST do MFaP seja acessada pelo IdP, mais precisamente pelo authnFlow com os parâmetros esperados. Os principais métodos são de requisição e resposta da operação de registro,

[5]http://flask.pocoo.org

 © 2017 Sociedade Brasileira de Computação

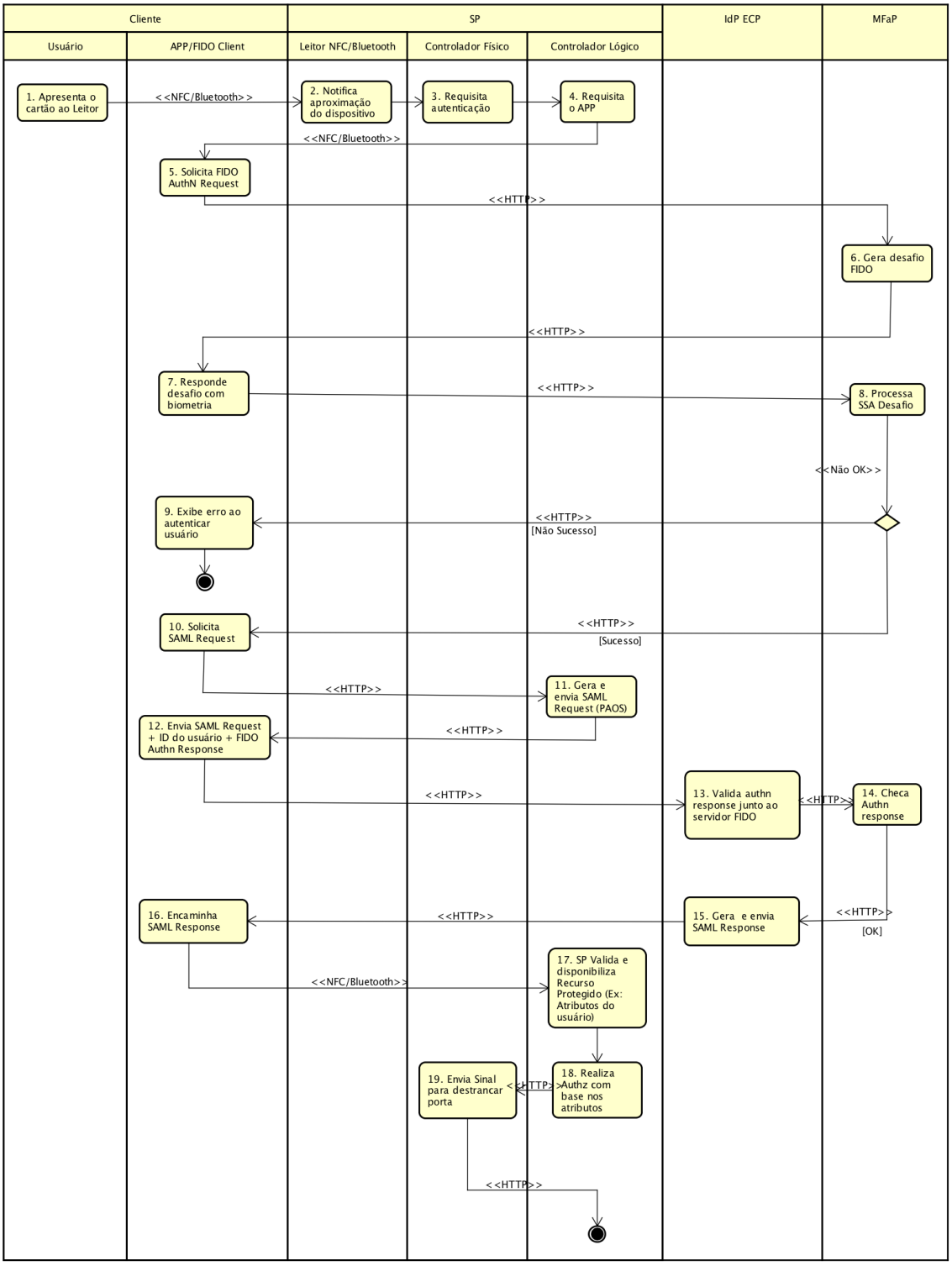


Figura 3. ***Diagrama de Atividade do Protótipo IoT com AuthN/AuthZ em federação***

© 2017 Sociedade Brasileira de Computação

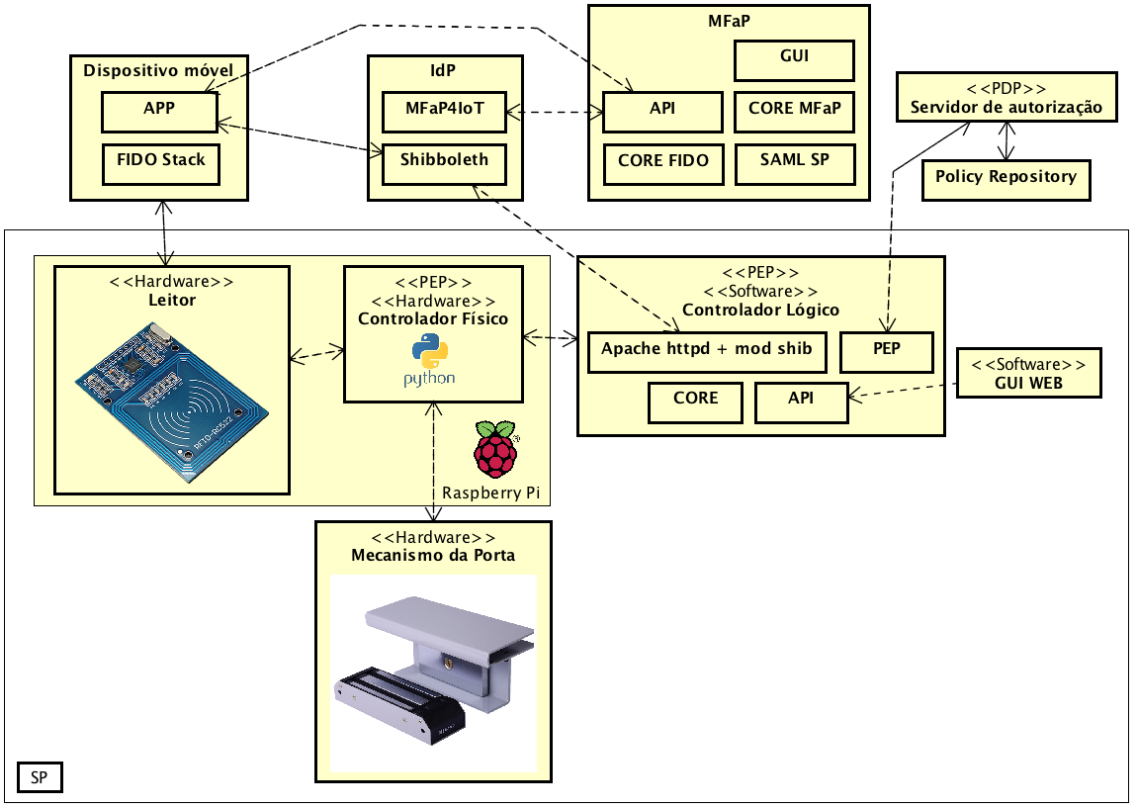


Figura 4. Diagrama de Componentes Concretos da solução SP.

requisição e resposta da operação de autenticação, e verificação de responsta de autenticação FIDO. Desse modo, o fluxo de autenticação consistirá em um módulo cliente para a API REST do MFaP, que será invocado durante o fluxo de autenticação de usuário.

Para a autenticação no IdP será desenvolvido um novo fluxo de autenticação em *SWF* chamado *mfa-fido-flow* que será tratado dentro do contexto de configuração MFA. Este fluxo será acionado através da lógica configurada no arquivo *mfa-auth-config.xml*. Esse arquivo está localizado no diretório *authn* do *IdP*, e deve ser editado para que o para que se delegue o controle do fluxo de login para o *MFA*. As transições entre fluxos de autenticação existentes e o momento em que cada fluxo chega ao fim são controlados por mapas de regras definidos em um *bean* chamado `shibboleth.authn.MFA.TransitionMap`.

5. Conclusão

Neste trabalho apresentamos nossa proposta de arquitetura para integrar IoT, controle de acesso e autenticação federada. Este trabalho é parte do GT-AMPTo (Autenticação Multi-fator para Todos) da RNP, onde a nossa responsabilidade é em implementar os mecanismos de autorização. A arquitetura apresentada está em fase de detalhamento, com a definição de um primeiro protótipo em bancada envolvendo Raspberry Pi 2, módulo Leitor RFID-RC522, trava Lockitron Mechanical Assembly, Arduino Nano, e Driver para Motor de passo DRV8834.

Como próximo passo, projetamos acrescentar suporte a autenticação federada ao protótipo, integrando-o com a federação CAFe. Em seguida, acrescentaremos suporte a

 © 2017 Sociedade Brasileira de Computação

autorização com políticas RBAC, e autenticação em dois fatores com FIDO.

Agradecimentos

Este trabalho foi desenvolvido no contexto do Grupo de Trabalho Autenticação Multi-fator para Todos (GT-AMPTo) financiado pela RNP.

Referências

Cardenas, A. A., Amin, S., and Sastry, S. (2008). Secure control: Towards survivable cyber-physical systems. In *2008 The 28th International Conference on Distributed Computing Systems Workshops*, pages 495–500.

Chadwick, D. W. (2009). Federated identity management. In *Foundations of Security Analysis and Design V*, pages 96–120.

Domenech, M. C. (2015). Uma infraestrutura de autenticação e de autorização para a web das coisas. Master's thesis, Universidade do Vale do Itajaí.

Domenech, M. C., Boukerche, A., and Wangham, M. S. (2016). An authentication and authorization infrastructure for the web of things. In *Proceedings of the 12th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Q2SWinet '16, pages 39–46, New York, NY, USA. ACM.

Evans-Pughe, C. (2005). The connected car. *IEE Review*, 51(1):42–46.

Fremantle, P., Aziz, B., Kopecký, J., and Scott, P. (2014). Federated identity and access management for the internet of things. In *Proceedings of the 2014 International Workshop on Secure Internet of Things*, SIOT '14, pages 10–17, Washington, DC, USA. IEEE Computer Society.

Hardt, D. (2012). The oauth 2.0 authorization framework. RFC 6749, RFC Editor. `http://www.rfc-editor.org/rfc/rfc6749.txt`.

Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., and Cybersecurity, S. (2014a). Guide to attribute based access control (abac) definition and considerations (draft). *NIST Special Publication*.

Hu, V. C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., and Scarfone, K. (2014b). SP 800-162. Guide to Attribute Based Access Control (ABAC) Definitions and Considerations. Technical report, National Institute of Standards and Technology, McLean and Clifton, VA, United States.

Islam, S. M. R., Kwak, D., Kabir, M. H., Hossain, M., and Kwak, K. S. (2015). The internet of things for health care: A comprehensive survey. *IEEE Access*, 3:678–708.

Lewis, F. L. (2004). Wireless sensor networks. *IEE Review*, pages 11–46.

Liu, J., Xiao, Y., and Chen, C. L. P. (2012). Authentication and access control in the internet of things. In *ICDCS Workshops*, pages 588–592. IEEE Computer Society.

Machani, S., Philpott, R., Srinivas, S., Kemp, J., and Hodges, J. (2014). Fido uaf architectural overview. *FIDO Alliance, December*.

 © 2017 Sociedade Brasileira de Computação

Minerva, R., Biru, A., and Rotondi, D. (2015). Towards a definition of the internet of things (iot). Technical report, IEEE.

Ndibanje, B., Lee, H.-J., and Lee, S.-G. (2014). Security analysis and improvements of authentication and access control in the internet of things. *Sensors*, 14(8):14786–14805.

OASIS (2010). extensible access control markup language (xacml) version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf`.

Refeds (2017). Refeds mfa profile. `https://refeds.org/profile/mfa`.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2):38–47.

 © 2017 Sociedade Brasileira de Computação

Estudo Sobre o Impacto das Vulnerabilidades de um Cartão Inteligente Sem Contato

Luiz Eduardo M. Matheus[1], **Eduardo P. Julio**[1], **Alex B. Vieira**[1], **Edelberto Franco**[1]
Departamento de Ciência da Computação

{**luizmendes, epagani, alex, edelberto**}**@ice.ufjf.br**

Abstract. *This paper shows a practical and theoretical study about vulnerabilities in a contactless smart card. In order to perform this research, we consider scenarios in which this type of smart card is used, confirming its large adoption. Besides, a state of the art review was conducted to identify the main attacks in a specific type of smart card, known as Mifare Classic. Finally, we perform a case study, where we explore the weaknesses previously discussed and successfully brake the security of the card, allowing a series of features such as recovering the secret keys, analyzing, identifying, and modifying sensible data.*

Resumo. *Este artigo realiza um estudo e avaliação teórica e prática sobre os impactos de ataques a um cartão inteligente sem contato. Para a condução do estudo são levados em consideração alguns dos principais ambientes que utilizam o modelo de cartão escolhido, demonstrando sua larga adoção. Além da apresentação do estudo do estado da arte sobre ataques em cartões inteligentes sem contato da Mifare. Um estudo de caso é realizado ao final do trabalho, possibilitando a quebra da segurança do cartão, a recuperação de suas chaves secretas, analise do conteúdo, identificação dos dados sensíveis e, ainda, modificação dos dados.*

1. Introdução

A identificação por rádio-frequência (*Radio Frequency IDentification* – RFID) é uma forma de comunicação sem fio que utiliza ondas de rádio para identificar e rastrear objetos ou pessoas através de um número de identificação [Finkenzeller 2010]. Um dos motivos da grande aceitação dessa tecnologia é sua facilidade de produção em massa e a portabilidade dessas etiquetas (*tags*) de identificação. Como apresentado pela [Das 2015] em 2015, o mercado de RFID já recebeu investimentos na ordem de US$10.1 Bilhões, e é previsto que em 2020 esse mercado será responsável pela movimentação de mais de US$13.2 bilhões. Todo esse investimento demonstra a importância da realização de estudos e avaliações sobre a tecnologia envolvida.

Sistemas RFID estão presentes nos mais diversos setores, como no varejo e transporte público, e, tradicionalmente, possuem apenas dois componentes principais: o leitor e a etiqueta RFID. O leitor é um dispositivo ligado à rede, responsável por enviar energia e dados para as etiquetas através de uma antena, assim como enviar os dados recebidos para um computador. As etiquetas RFID são compostas por pequenos *chips*, que armazenam dados em sua memória, ligados à uma antena e acoplados à um dispositivo como um cartão, por exemplo. A integridade dos dados deve ser mantida, assim como a autenticidade do portador da etiqueta RFID. Uma falha pode evidenciar vulnerabilidades, levando a possíveis ataques como: clonagem de cartões, corrupção ou adulteração de dados.

 © 2017 Sociedade Brasileira de Computação

Neste trabalho é abordado um modelo específico de cartão inteligente sem contato, utilizado como base para um estudo de caso com a avaliação de possíveis impactos aos sistemas que o utilizam.

O artigo está organizado da seguinte maneira: além desta introdução, a Seção 2 apresenta a justificativa e motivação do trabalho na condução de seus estudos, através do levantamento sobre a adoção do modelo de cartão inteligente sem contato *Mifare Classic* no mercado. Na Seção 3, alguns trabalhos relacionados são abordados. A Seção 4 apresenta a arquitetura de memória e segurança do cartão. Já a Seção 5, apresenta um estudo de caso sobre um ataque em um ambiente real. Por fim, o estudo é concluído na Seção 6.

2. Justificativa

O Mifare Classic MF1S50YYX é um cartão desenvolvido pela empresa NXP Semicondutores [*]. Ele se encaixa na categoria de cartões inteligentes sem contato, sendo amplamente utilizado desde o pagamento no transporte público até o controle de acesso a prédios, áreas restritas, e em substituição aos tickets de estacionamento rotativo.

A fim de justificar a escolha do modelo de cartão estudado, foi realizado um levantamento sobre as empresas e ambientes que o utilizam atualmente. Sendo assim, pode-se citar as empresas: RioCard[†], responsável pelo acesso ao transporte público por ônibus, metrô e barcas no grande Rio, BHTrans, com mais de 34 milhões de viagens realizadas no mês de maio de 2017[‡], sistema de ticket e pagamento de Taiwan (EasyCard), além de vários sistemas de transporte público em cidades como Miami (Miami-Dade), Buenos Aires (SUBE), Istanbul (Istanbul Kart), Stockholm (SL Access)[§]e por fim a Estapar.[¶] A Estapar, por exemplo, oferece serviços como a área azul (estacionamento rotativo) em 17 cidades brasileiras de médio e grande porte. Levando em consideração essa empresa e a cidade de Juiz de Fora/MG[‖], pode-se ter um exemplo do valor e impacto, caso um ataque de repetição tenha sucesso e seja utilizado por parte dos usuários do sistema. A empresa é responsável por mais de 2392 vagas para carros e 358 para motos, ofertados a R$ 2,30 o período de 2h. Exemplificando a ocupação de forma pessimista, com apenas 50% das vagas para carros ocupadas no período de 8h/dia (comerciais) e 40h/semana (somente dias úteis), tem-se um total de R$ 11.003,20/dia ou R$ 220.064,00/mês (20 dias úteis), só na cidade de Juiz de Fora/MG. Considerando que 10% dos usuários realizem o ataque, o prejuízo mensal seria na ordem de R$ 20 mil reais (R$ 240 mil reais/ano).

3. Trabalhos Relacionados

Na última década, diversos trabalhos foram realizados com o objetivo de demonstrar as fraquezas do cartão *Mifare Classic* em diversos cenários diferentes. A ampla utilização desse cartão, em especial para aplicações de pagamentos eletrônicos e controle de acesso, é a principal justificativa para tais trabalhos.

Um dos primeiros trabalhos de segurança envolvendo o *Mifare Classic* em uma aplicação real foi conduzido na Holanda [de Koning Gans 2008]. Em sua tese, Gerhard de

[*] https://www.mifare.net/en/products/chip-card-ics/mifare-classic/

[†] https://www.cartaoriocard.com.br/

[‡] https://goo.gl/eRTmjU

[§] http://sl.se/en/eng-info/fares/sl-access/

[¶] https://www.estapar.com.br/

[‖] Cidade com mais de 550 mil habitantes.

 © 2017 Sociedade Brasileira de Computação

Koning Gans estuda a arquitetura do cartão e o protocolo de comunicação, para o sistema de bilhetagem eletrônico do transporte público daquele país. Entre suas contribuições estão: (I) uma visão geral dos comandos e respostas do protocolo; (II) o desenvolvimento de um método para ler informações do cartão sem conhecimento da chave secreta; (III) a apresentação da possibilidade de modificar os dados presentes no cartão. No mesmo ano, Ryan *et al.* realizou uma demonstração do ataque de repetição nos cartões Mifare utilizados nos metrôs de Chicago [Ryan 2008].

Tan et al. [Tan 2009] descrevem em seu trabalho dois ataques práticos à cartões *Mifare Classic*. Primeiro, as vulnerabilidades do cartão de controle de acesso utilizado na *Imperial College of London* são expostas, demonstrando a possibilidade de se passar por qualquer funcionário válido. Os autores também demonstram a viabilidade de se obter acesso à toda a estrutura interna do *Oyster Card*, cartão utilizado pelos metrôs de Londres na época, localizando precisamente os dados que representam o crédito atual e o histórico de transações. É interessante observar que os impactos das descobertas deste trabalho, caso tornados públicos, poderiam gerar um prejuízo de milhões de libras para as operadores do transporte público de Londres. Menos de um ano depois, os cartões utilizados pelos sistemas avaliados pelos autores foram substituídos pelo *Mifare DESfire* [Balaban 2010].

Ao longo dos últimos anos, vários trabalhos evidenciaram tais vulnerabilidades e demonstraram estudos de caso práticos. Em 2013, Golic *et al.* apresentaram uma revisão bibliográfica das principais técnicas utilizadas em ataques ao cartão [Golic 2013]. No mesmo ano, Chiu *et al.* demonstraram um ataque prático a alternativas adotadas após a descoberta das vulnerabilidades, em que o mecanismo de segurança dos cartões era superficialmente modificada com base nas falhas descobertas, de forma a evitar novos ataques [Chiu et al. 2013]. Em 2015, Meijer *et al.* observaram o fato do cartão ainda ser o mais utilizado ao redor do mundo, apesar das falhas, e apresentaram um novo ataque que explora o processo de autenticação do cartão [Meijer and Verdult 2015].

Em especial, no Brasil, há um trabalho na área, realizado em 2011 [de Souza 2011]. Nesta dissertação, foram expostas as fraquezas do cartão, apresentados os principais ataques possíveis e propostas alternativas para evitar tais ataques. Entretanto, não há nenhuma experiência envolvendo aplicações do mundo real.

	de Koning *et al.*	Ryan *et al.*	Tan *et al.*	**Este Trabalho**
País	Holanda	EUA	Inglaterra	Brasil
Ano	2008	2008	2009	2016
Aplicação	OV-Chipkaart	Charlie Card	Oyster Card	Estacionamento Rot.
Ferramentas	Proxmark III	USPR	Proxmark III	RPi + ACR122U
Valor gasto	U$ 450,00	U$ 700,00	U$ 450,00	U$ 65,00

Tabela 1. Trabalhos relacionados.

Apesar da literatura acerca de problemas de segurança em cartões *Mifare Classic*, ainda são comuns os casos de sistemas que o utilizam. A Tabela 1 apresenta um comparativo entre trabalhos realizados previamente ao estudo aqui apresentado. Já a última coluna da Tabela 1 se refere a este trabalho de pesquisa. Em todos os trabalhos foi avaliado o modelo de cartão *Mifare Classic 1k*. É interessante notar, principalmente, que os equipamentos necessários para executar os mesmos ataques têm preços cada vez mais

 © 2017 Sociedade Brasileira de Computação

acessíveis.

4. Arquitetura do *Mifare Classic*

Com base em toda a justificativa e motivação apresentada até aqui, escolhemos o *Mifare Classic 1k* para conduzir o estudo e demonstração da fragilidade que deve ser levada em consideração na adoção dessa tecnologia. Sendo assim, apresentamos a arquitetura do *Mifare Classic 1k*.

O *chip* do cartão inteligente *Mifare Classic 1k* consiste de uma memória EEPROM de 1kb, uma interface RF, responsável pela comunicação com o leitor, e uma unidade de controle digital, responsável pelo registro de dados. Energia e dados são transmitidos por uma antena feita por uma bobina ligada diretamente ao *chip*. A memória do cartão é divida em 16 setores.**

Qualquer operação em um bloco de dados ou bloco *sector trailer* só é autorizada pelo cartão caso as condições de acesso sejam seguidas. Estas condições são representadas por 3 *bits*, localizados de forma invertida e não-invertida em cada bloco *sector trailer*, como pode ser visto na Figura 1. Em outras palavras, cada setor tem uma série de condições de acesso específicas para cada um de seus blocos.

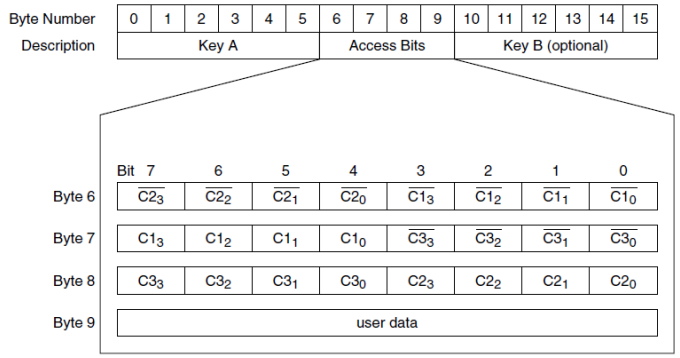


Figura 1. Condições de acesso de um cartão *Mifare Classic 1k* [NXP 2011].

O cartão inteligente *Mifare Classic 1k* possui alguns mecanismos de segurança internos. A cifra utilizada para criptografar a comunicação é proprietária, conhecida como *CRYPTO-1*. Porém, em [Nohl et al. 2008] foi realizada a engenharia reversa no *hardware* do cartão, o que expôs totalmente a cifra em questão. Há também chaves de segurança para acesso aos dados, responsáveis pelas operações de acesso aos blocos, são elas as duas chaves A e B, ambas com 48 *bits*, que ficam registradas no *sector trailer* de cada setor. Durante o processo de autenticação, o leitor deve ter conhecimento dessas chaves, para realizar qualquer tipo de operação de memória no cartão seguindo as regras definidas pelas condições de acesso.

A autenticação em um *Mifare Classic 1k* é feita seguindo um protocolo de autenticação mútua em três passos. Neste processo, o leitor especifica o bloco ao qual quer ter acesso. A partir daí são realizados três passos, como ilustrado na Figura 2:

Seus passos podem ser descritos como: (1) O cartão envia um desafio para o leitor; (2) A partir deste ponto, as mensagens trocadas passam a ser criptografadas. O

**O cartão *Mifare 4k* apresenta estrutura muito semelhante à do *Mifare 1k*.

 © 2017 Sociedade Brasileira de Computação

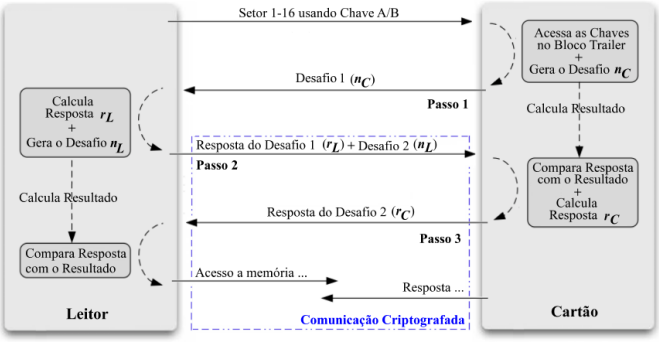


Figura 2. Processo de autenticação de um Mifare Classic 1k [de Souza 2011].

leitor responde ao desafio e envia um novo desafio; (3) Por fim, se o leitor respondeu corretamente, o cartão responde ao seu desafio. Caso contrário, o cartão não responde.

A autenticação segura no *Mifare Classic 1k* só era possível por causa dos componentes envolvidos no mecanismo de segurança proprietário do cartão. A cifra *CRYPTO-1* implementa a criptografia em *hardware*, e possui dois principais componentes: um gerador de números pseudo-aleatórios e um registrador linear de *feedback*. Porém, estes dados se tornaram públicos depois da engenharia reversa feita em [Nohl et al. 2008].

5. Estudo de Caso

Para analisar as vulnerabilidades em um ambiente real, foi feito um estudo de caso utilizando um cartão inteligente *Mifare Classic 1k*, adotado como forma de pagamento para estacionamentos rotativos em diversas cidades do Brasil.[††]Para este estudo, foram utilizados dois equipamentos acessíveis e portáteis: *Raspberry Pi B+* e leitor NFC ACR122U. O *Raspberry Pi B+* é um microcomputador, contendo 512MB de RAM e o sistema operacional Kali Linux instalado, cuja escolha se deve ao fato de já conter um algoritmo importante para o procedimento: *Mifare Classic Offline Cracker (mfoc)*. O leitor NFC ACR122U é utilizado para fazer todas as operações com os cartões estudados.

Dois trabalhos exploraram aspectos da autenticação nos cartões Mifare Classic: o trabalho [de Koning Gans 2008] recuperou, com sucesso, informações secretas da memória do cartão, explorando fraquezas encontradas no gerador de números pseudo-aleatórios. Além desta pesquisa, [Courtois 2009] propôs um novo ataque, em que uma única chave era recuperada. É importante observar que estes dois trabalhos geraram os dois algoritmos que serão utilizados para recuperar as chaves do cartão: *MiFare Offline Cracker (mfoc)* e *MiFare Classic Universal Toolkit (mfcuk)*.

Daqui em diante tem-se em quatro partes do estudo. São elas: (I) a tentativa de recuperar as chaves secretas do cartão utilizando dois algoritmos conhecidos: *Nested Authentication Attack* [de Koning Gans et al. 2008] e *Dark Side Attack* [Courtois 2009] (Seção 5.1). Neste ponto, o interesse deste trabalho está em saber qual o nível de dificuldade que se tem, nos dias de hoje, para executar os ataques com ferramentas acessíveis. (II) Tendo acesso ao conteúdo do cartão, a segurança envolvida no registro de dados é analisada (Seção 5.2). E (III), onde são identificadas fraquezas encontradas no sistema como

[††]O nome da empresa foi suprimido em respeito à mesma.

 © 2017 Sociedade Brasileira de Computação

um todo (Seção 5.3). Por fim, são discutidos alguns ataques que podem ser executados no cartão e suas possíveis consequências (Seção 5.4).

5.1. Recuperação de chaves secretas

Inicialmente, foi utilizado o algoritmo *Mifare Classic Offline Cracker (mfoc)* na tentativa de recuperar as chaves dos setores do cartão. Este algoritmo só é bem sucedido caso uma chave seja conhecida, ou se o cartão utiliza alguma chave padrão (de fábrica). A Figura 3 mostra um trecho da saída gerada pelo algoritmo *mfoc*, contendo algumas informações sobre o cartão lido, todos os setores e as tentativas de autenticação utilizando chaves padrões. Na mesma figura, podemos observar quatro símbolos diferentes: "x", indicando que ambas as chaves foram descobertas, "/", apenas a chave A foi descoberta, "n", apenas a chave B foi descoberta e ".", onde nenhuma chave foi descoberta.

```
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
* UID size: single
* bit frame anticollision supported
      UID (NFCID1): e4  d6  66  30
     SAK (SEL_RES): 08
* Not compliant with ISO/IEC 14443-4
* Not compliant with ISO/IEC 18092

Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE Classic 1K
* MIFARE Plus (4 Byte UID or 4 Byte RID) 2K, Security level 1
* SmartMX with MIFARE 1K emulation
Other possible matches based on ATQA & SAK values:

Try to authenticate to all sectors with default keys...
Symbols: '.' no key found, '/' A key found, '\' B key found, 'x' both keys found
[Key: ffffffffffff] -> [xxxxxxxxxxxx..xx]
[Key: a0a1a2a3a4a5] -> [xxxxxxxxxxxx..xx]
[Key: d3f7d3f7d3f7] -> [xxxxxxxxxxxx..xx]
[Key: 000000000000] -> [xxxxxxxxxxxx..xx]
[Key: b0b1b2b3b4b5] -> [xxxxxxxxxxxx..xx]
[Key: 4d3a99c351dd] -> [xxxxxxxxxxxx..xx]
[Key: 1a982c7e459a] -> [xxxxxxxxxxxx..xx]
[Key: aabbccddeeff] -> [xxxxxxxxxxxx..xx]
[Key: 714c5c886e97] -> [xxxxxxxxxxxx..xx]
[Key: 587ee5f9350f] -> [xxxxxxxxxxxx..xx]
[Key: a0478cc39091] -> [xxxxxxxxxxxx..xx]
[Key: 533cb6c723f6] -> [xxxxxxxxxxxx..xx]
[Key: 8fd0a4f256e9] -> [xxxxxxxxxxxx..xx]
```

Figura 3. Execução do algoritmo *Mifare Offline Cracker*.

Como pode ser visto na Figura 3, o cartão analisado possuía diversas chaves padrão, e os setores que continham essas chaves foram explorados para autenticar os demais. Além disso, dos 16 setores presentes no cartão, apenas dois são utilizados para registrar algum tipo de conteúdo (setores 12 e 13), e estes são os únicos que possuem uma chave personalizada. Dessa forma, em menos de 5 minutos, as chaves A e B dos setores 12 e 13 foram recuperadas, possibilitando a leitura completa de todos os blocos do cartão.

Neste caso, não houve a necessidade de utilizarmos o algoritmo *MiFare Universal ToolKit (mfcuk)*, pois o cartão possuía pelo menos uma chave padrão, suficiente para o algoritmo *mfoc* ser executado com sucesso. É importante notar que mesmo que a empresa responsável tivesse se preocupado com a segurança neste ponto, atribuindo chaves personalizadas para cada setor, ainda assim elas seriam recuperadas com sucesso. Isso porque, neste caso, executaríamos o *mfcuk*, recuperando uma única chave de um setor, que seria utilizada posteriormente como parâmetro para o *mfoc*, chegando ao mesmo resultado.

Em outras palavras, como esperado, não é possível confiar na segurança proprietária do cartão *Mifare Classic 1k*. Isso leva à segunda parte do estudo de caso, onde

 © 2017 Sociedade Brasileira de Computação

são analisadas com mais detalhes as práticas de segurança adotadas pela empresa ao registrar os seus dados no cartão, assim como a possibilidade de praticar ataques como clonagem de cartões e adulteração de dados.

5.2. Análise do conteúdo do cartão

Depois do processo descrito anteriormente, obteve-se acesso à todo o conteúdo registrado no cartão. O próximo passo consiste em identificar cada dado registrado, e a que este dado se refere (crédito, identificação, etc).

Inicialmente, foi feito um *dump* do conteúdo do cartão antes de realizar qualquer tipo de operação (direto da empresa). Dessa forma, obtemos a imagem do cartão com crédito inicial de R$9,00. Na próxima etapa, foi realizado um pagamento regular, seguido de um *dump*. Através da observação dos diferentes *dumps*, foi possível identificar que os setores que possuíam chaves personalizadas de fato guardavam os dados da aplicação.

Foi observado que alguns dados são escritos de forma indireta no cartão. No entanto, como há liberdade de escolha de horário e local de transação, foi realizado um ataque similar ao conhecido "ataque de texto escolhido", em que pode-se escolher o texto plano que será escrito, e obter os dados correspondentes. Neste caso, o texto plano é o horário, dia, mês e crédito. Para que esta técnica seja bem sucedida, e seja possível identificar os outros dados escritos nos blocos, outras operações foram feitas, em dias diferentes, sendo que um *dump* do conteúdo era feito sistematicamente após cada operação realizada. Assim, através da observação dos blocos modificados, foi possível identificar onde e quais informações eram registradas no cartão.

Todas as informações relacionadas à transação realizada ficam registradas em dois blocos diferentes (50 e 53), provavelmente por questões de redundância. Os dois primeiros *bytes* correspondem ao dia e ao mês, respectivamente. O bloco não registra os dados de crédito de maneira direta, ou seja, R$ 5,00 não é representado como $0x00\ 0x05$, e sim como $0xf4\ 0x01$ (dois bytes). Outra forma de analisar este valor é transformando-o em centavos. Neste caso, R$ 5,00 passa a ser representado como 500 centavos. Fazendo a conversão deste valor para binário, obtemos $00000001\ 11110100$, que pode ser transformado em $0x01\ 0xf4$, valor encontrado nos *bytes* 5 e 6, dos blocos 50 e 53 do cartão, de forma invertida. Foi observado que o horário em que a transação foi efetuada é registrado no cartão, em forma de hexadecimal. Dessa forma, se a transação foi feita às 16:36h, os dados $0x10$ (16) $0x24$ (36) são encontrados no cartão, nos *bytes* 3 e 4 dos blocos 50 e 53.

Com relação aos blocos de dados, há o último *byte*. Depois de alguns pagamentos realizados, foi observado que este *byte* sempre era modificado, e aparentemente não seguia um padrão. Logo, uma possibilidade era de que algum tipo de soma de verificação (*checksum*) era utilizada, com o objetivo de validar os dados registrados naquele bloco. Uma operação conhecida para criação de *checksum* é o XOR. Depois de alguns testes realizados, com valores diferentes, chegou-se à conclusão de que o *byte* de *checksum* é calculado através de uma série de operações XOR entre os dados daquele bloco. Na Figura 4, é possível observar um exemplo da soma de verificação executada no cartão para os dados referentes à Figura 5.

Em outro bloco, há 6 *bytes* registrados que não são modificados, independente das transações realizadas. Foi assumido que estes dados constituem algum tipo de identificação para o cartão. Além disso, há um terceiro bloco que possui apenas dois

 © 2017 Sociedade Brasileira de Computação

$$49 \oplus 9f \oplus 0d \oplus 0a \oplus 84 \oplus 03 = 56$$

Figura 4. Operação de soma de verificação presente no cartão.

.

bytes registrados: ”02”e 02”, no início e no final. Este bloco também é estático. Esses dois blocos não são levados em consideração para o estudo realizado, visto que não contém informações sobre créditos ou transações financeiras.

Dessa forma, após análise do conteúdo registrado no cartão, foi possível identificar o local exato de todos os dados referentes às transações financeiras. Na Figura 5, cada *byte* do bloco pode ser visto, com seu respectivo valor. Neste caso, estes dados são: mês, dia, hora, minuto, crédito e soma verificadora. Assim, o próximo passo consiste em analisar as fraquezas encontradas no sistema estudado, assim como suas possíveis consequências.

49	9f	0d	0a	84 03	00 00 00 00 00 00 00 00 00	56
Mês	Dia	Hora	Minuto	Crédito		Check Sum

Figura 5. Dados referentes às transações financeiras.

.

5.3. Fraquezas Encontradas

Após obter acesso ao conteúdo do cartão e analisar a natureza dos dados registrados nele, uma série de vulnerabilidades foram identificadas:

- Setores com chaves padrões: foi observado que diversos setores do cartão possuem chaves padrões (14 dos 16). Este fato faz com que o algoritmo *mfoc* execute rapidamente, recuperando as chaves dos dois outros setores. Após uma primeira tentativa, pode-se registrar as chaves recuperadas e utilizá-las como parâmetros para rodar o algoritmo, obtendo acesso aos demais setores do cartão em segundos;
- Dados registrados diretamente: grande parte dos dados relacionados às transações realizadas são registrados no cartão de maneira direta, sem nenhum tipo de criptografia adicional. Isso é um problema grave, pois não se pode contar apenas com a criptografia do cartão. Assim, uma vez com acesso ao conteúdo dos blocos, é possível identificar o significado de cada *byte*;
- Setores com chaves iguais: os dois setores citados no item anterior possuem a mesma chave, o que faz o algoritmo executar ainda mais rápido, dando acesso ao seu conteúdo. A execução do algoritmo pode ser vista na Figura 3;
- Mecanismo de validação de dados fraco: o bloco do cartão onde são registrados os dados referentes à transação financeira possui um único mecanismo de segurança para validar os dados: um *checksum* simples. Descobrindo o funcionamento desse mecanismo, é possível editar os dados do cartão, envolvendo crédito;
- Sistema opera *offline*: o sistema estudado funciona completamente *offline*, e os dados não são registrados em nenhum outro local que não seja o próprio cartão. Em outras palavras, toda a segurança, neste caso, é confiada em algoritmos de criptografia proprietários do cartão. Portanto, caso um usuário compre um cartão com R$20,00, faça um *dump* do conteúdo, utilize o cartão até zerar os créditos

 © 2017 Sociedade Brasileira de Computação

e depois recupere o valor inicial através da cópia do *dump* para o cartão, não haverá nenhuma maneira de conferir a integridade e autenticidade daqueles dados, permitindo que este usuário use o cartão ilimitadamente.

5.4. Avaliações de Impacto

Considerando o estudo realizado e as fraquezas identificadas, há uma série de ataques que podem ser realizados. Primeiro, tendo em mãos o *dump* do conteúdo do cartão com um certo valor, é possível realizar um ataque conhecido como ataque de repetição (*Replay Attack*), a fim de recarregar o cartão quantas vezes o atacante desejar.

Além disso, a partir do momento em que os *bytes* são devidamente localizados e identificados, e o mecanismo de *checksum* descoberto, é possível adulterar os dados contidos no cartão, de maneira a simular qualquer valor de crédito. Este processo é simples, bastando editar os valores de crédito de acordo com a nova quantia desejada e calcular o *checksum* com os novos valores.

Por último, um possível ataque é a clonagem de cartões. Os cartões *Mifare Classic* vêm de fábrica com seu bloco 0 protegido contra escrita, essa estratégia garantiria a unicidade do identificador. Entretanto, atualmente, há os chamados "cartões mágicos *mifare classic 1k*", cujos blocos 0 são editáveis, o que permite a clonagem de um cartão.

As consequências destes ataques são claras: qualquer empresa que utilize o cartão *Mifare Classic 1k* para aplicações envolvendo transações financeiras e confie unicamente no mecanismo de segurança proprietário, está sujeita a grandes prejuízos financeiros. A situação se torna mais sensível quando se trata de aplicações de controle de acesso, em que a entrada a certos locais é restrita à um grupo de funcionários.

É importante observar que grande parte dos ataques podem ser evitados caso o sistema opere de forma *online*, de modo que um terceiro dispositivo mantenha as informações registradas em cada cartão atualizadas. Assim, é possível fazer a verificação da integridade dos dados, para cada transação realizada. A utilização de criptografia para registrar os dados também é uma medida consistente, visto que com esse mecanismo torna-se mais difícil localizar os dados sensíveis. Entretanto, a medida mais prudente a ser tomada é a substituição do cartão inteligente *Mifare Classic 1k* nas aplicações, recomendada pela própria fabricante [NXP 2009].

6. Conclusões

Este trabalho apresentou um estudo e avaliação teórica e prática sobre os impactos aos ambientes que utilizam um cartão inteligente sem contato frágil, do ponto de vista da segurança. Além de um levantamento sobre alguns dos principais ambientes que utilizam esse modelo de cartão, foi levado em consideração o impacto ao qual tais ataques acarretariam a essas empresas. Para validar a motivação deste trabalho foi ainda realizado um levantamento e estudo do estado da arte sobre ataques em cartões inteligentes sem contato *Mifare*, culminando no estudo dos cartões *Mifare Classic 1k*. Ao final do trabalho, foi possível realizar um estudo de caso considerando a utilização de algoritmos disponíveis na literatura para quebrar a segurança do cartão e recuperar as chaves secretas; analisar o conteúdo presente no cartão, e identificar os dados sensíveis (créditos e *checksum*) e, ainda, modificar estes dados com o auxílio de dispositivos acessíveis, alcançando o resultado esperado.

 © 2017 Sociedade Brasileira de Computação

A metodologia aplicada neste trabalho pode ser utilizada para a verificação de outros sistemas que utilizam cartões inteligentes sem contato, assim como a avaliação da arquitetura de outros modelos de cartões inteligentes sem contato. É intenção dos autores continuar sua pesquisa investigativa sobre fragilidades em cartões inteligente sem contato e dos sistemas que o utilizam. Desta forma, serão avaliados outros cartões da família *Mifare* em um trabalho futuro.

Referências

Balaban, D. (2010). Transport for London to Discard Mifare Classic. `https://goo.gl/z1g9mu`. [Online; acessado em 09/06/2017].

Chiu, Y.-H., Hong, W.-C., Chou, L.-P., Ding, J., Yang, B.-Y., and Cheng, C.-M. (2013). A practical attack on patched mifare classic. In *International Conference on Information Security and Cryptology*, pages 150–164. Springer.

Courtois, N. T. (2009). The dark side of security by obscurity and cloning mifare classic rail and building passes, anywhere, anytime.

Das, R. (2015). Rfid forecasts, players and opportunities 2016-2026. `https://goo.gl/9FpRcX`. Online; acessado em 29/06/2017.

de Koning Gans, G. (2008). Analysis of the mifare classic used in the ov-chipkaart project. *Radboud University Nijmegen, Netherlands*.

de Koning Gans, G., Hoepman, J.-H., and Garcia, F. D. (2008). *A practical attack on the MIFARE Classic*. Springer.

de Souza, W. B. (2011). *Cartão MIFARE classic ataques e medidas de contorno*. PhD thesis, Universidade de São Paulo.

Finkenzeller, K. (2010). *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. John Wiley & Sons.

Golic, J. D. (2013). Cryptanalytic attacks on mifare classic protocol. In *CT-RSA*, pages 239–258. Springer.

Meijer, C. and Verdult, R. (2015). Ciphertext-only cryptanalysis on hardened mifare classic cards. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 18–30. ACM.

Nohl, K., Evans, D., Starbug, S., and Plötz, H. (2008). Reverse-engineering a cryptographic rfid tag. In *USENIX security symposium*, volume 28.

NXP (2009). End to end system security risk considerations for implementing MIFARE Classic. `https://goo.gl/dnLcuh`. [Online; acessado em 09/06/2017].

NXP (2011). Mifare classic 1k - mainstream contactless smart card ic for fast and easy solution development. `https://goo.gl/V3zb9F`. Online; acessado em 01/07/2017.

Ryan, R. (2008). Anatomy of a subway hack. `http://tech.mit.edu/V128/N30/subway/Defcon_Presentation.pdf`. Online; acessado em 11 de Setembro de 2017.

Tan, W. H. (2009). Practical attacks on the mifare classic. *Imperial College London*.

 © 2017 Sociedade Brasileira de Computação

Desenvolvendo Aplicações de Cidades Inteligentes com Autenticação e Autorização baseada na Plataforma FIWARE

Gabriela Cavalcante da Silva[1], **Carlos Eduardo da Silva**[1]

[1]Instituto Metrópole Digital - Universidade Federal do Rio Grande do Norte (UFRN)
Avenida Senador Salgado Filho, 3000, Lagoa Nova – CEP: 59.078-970 – Natal/RN

gabicavalcantesilva@gmail.com, kaduardo@imd.ufrn.br

Abstract. *The FIWARE platform offers a set of reusable components that simplify the development of Smart City applications. Among these components, we can find mechanisms for authentication and authorization. However, their use is not a trivial process. This paper presents a study on the authentication and authorization mechanisms of the Fiware platform. As a case study, we have developed a Web application that demonstrates the use of these mechanisms. Among the results achieved, we can mention reference material in Portuguese and contribution to the official FIWARE platform source code.*

Resumo. *A plataforma FIWARE oferece um conjunto de componentes reutilizáveis que facilita o desenvolvimento de aplicações para Cidades Inteligentes. Dentre estes componentes, podemos encontrar mecanismos para autenticação e autorização. Entretanto, sua utilização não é um processo trivial. Este artigo apresenta um estudo sobre os mecanismos de autenticação e autorização da plataforma Fiware. Como estudo de caso, foi desenvolvida uma aplicação Web que demonstra a utilização dos referidos mecanismos. Dentre os resultados alcançados, podemos mencionar material de referência em português e contribuição ao código oficial da plataforma FIWARE.*

1. Introdução

O desenvolvimento de aplicações para cidades inteligentes [Caragliu et al. 2009] envolve um número de tecnologias, tais como Internet das Coisas (IoT) [Minerva et al. 2015], Computação em Nuvem [Vogels 2008] e Big data [Townsend 2013], necessitando de uma infraestrutura de suporte que sirva de base para o desenvolvimento de tais aplicações. Nesse contexto, a plataforma FIWARE[1] surge como uma candidata a infraestrutura de apoio ao desenvolvimento de aplicações e serviços de Cidades Inteligentes, oferecendo uma gama de serviços comuns para o desenvolvimento de soluções para Cidades Inteligentes, tratando diversos aspectos, como por exemplo, mecanismos de Computação em nuvem, IoT, segurança, entre outros.

Sabemos que é altamente desejado que informações, aplicativos, ou softwares sejam acessados somente por aqueles autorizados para tal. Para isso, existem vários mecanismos que lidam com diferentes aspectos de segurança da informação, tais como autenticação, políticas de segurança, entre outros. A plataforma FIWARE foi projetada para considerar estes aspectos em uma abordagem *secure by design*, onde atributos de segurança da plataforma em si, e das aplicações que executem sobre ela, são considerados

[1]https://www.fiware.org

 © 2017 Sociedade Brasileira de Computação

através de uma arquitetura de segurança que engloba questões relacionadas a gestão de identidades, controle de acesso e monitoramento de eventos de segurança. Isso é conseguido com um conjunto de componentes (denominados *Generic Enablers* - GEs) que podem ser integrados com outros GEs da plataforma [Fiware 2014].

No contexto do projeto Smart Metrópolis[2] realizamos um estudo dos mecanismos de segurança da plataforma FIWARE com o objetivo de fornecer apoio aos outros membros do projeto no desenvolvimento de aplicações com autenticação e autorização. Entretanto, enquanto estudávamos a plataforma FIWARE, encontramos alguns problemas que dificultaram o seu uso. Um deles foi o fato da sua utilização não ser trivial, tanto por falta de uma infraestrutura que forneça um ambiente de desenvolvimento, como por falta de aplicações de estudo de caso que possam ser utilizadas como exemplo por desenvolvedores.

Neste contexto, este trabalho apresenta um estudo sobre a plataforma FIWARE com foco nos GEs responsáveis pela autenticação e autorização: *Identity Management* GE, *Authorization PDP* GE e *PEP Proxy* GE. Nosso objetivo é demonstrar como utilizar os mecanismos de controle de acesso da plataforma FIWARE no desenvolvimento de aplicações seguras, tendo em mente as fragilidades encontradas na plataforma e como poderíamos contribuir para corrigi-las. Para tal, desenvolvemos uma aplicação Web Python que será utilizada como estudo de caso explorando a ferramenta de gerenciamento de contêineres Docker para implantação da infraestrutura de suporte para a aplicação exemplo. Este estudo resultou na produção de referência em Português[3], incluindo código-fonte e descrições Docker, demonstrando como utilizar as GEs de segurança da plataforma FIWARE. Outro resultado alcançado foi a correção de um bug no componente PEP Proxy que foi aceita pela comunidade de desenvolvedores FIWARE, e agora se encontra no repositório oficial.

2. Contextualização

Esta seção apresenta uma visão geral dos componentes de segurança da plataforma FIWARE, identificando os diferentes GEs que implementam os recursos de segurança da plataforma, tais como gerenciamento de identidade, e controle de acesso. Em seguida, descrevemos a infraestrutura Docker criada para suportar nosso estudo de caso.

2.1. GEs de Segurança Fiware

A arquitetura de segurança do FIWARE [Fiware 2014] é composta por três GEs que juntas fornecem um conjunto abrangente de serviços para aplicações com o intuito de cumprir com os requisitos de segurança de autenticação e autorização. São estes *Identity Management* GE, *Authorization PDP* GE e *PEP Proxy* GE.

O *Identity Management* GE (IdM) é responsável pelo gerenciamento de identidades de usuários, organizações e aplicações, assim como de suas respectivas credenciais, e da autenticação dessas entidades. Este GE considera requisitos para a gestão de identidades de "coisas" (IoT), e atende às especificações para o gerenciamento de identidades sobre diferentes domínios. Ele oferece uma API para o gerenciamento de identidades em

[2] `http://smartmetropolis.imd.ufrn.br`

[3] Disponível em `https://projetos.imd.ufrn.br/SmartMetropolis-InfraestruturaGroup/Example-Application-Security`

 © 2017 Sociedade Brasileira de Computação

ambientes multi-tenant, permitindo ainda a gestão de autorização com suporte a definição de papéis e permissões específicas de uma organização ou de uma aplicação. Desenvolvedores interagem com este componente para registrarem suas aplicações no IdM, e para gerenciar a segurança de suas aplicações por meios de políticas baseada em papel (*Role-Based Access Control* - RBAC) [ANSI 2004]. Usuário finais usam o IdM GE para se registrarem, e gerenciar seus respectivos perfis e suas organizações, enquanto que todas as entidades clientes de aplicações utilizam o IdM para fins de autenticação, suportando as especificações dos padrões SAML 2.0[4] e OAuth2[5]. A implementação de referência deste GE é o **Keyrock**[6].

O *Authorization PDP* GE (PDP) é o componente responsável por gerenciar políticas de autorização na plataforma FIWARE. Este componente utiliza a linguagem XACML para representação de políticas, implementando a especificação XACML Core v3.0. As políticas definidas pelos desenvolvedores por meio do IdM são gerenciadas pelo PDP, que é responsável também por avaliar requisições de acesso, encaminhadas pelo PEP, para permitir ou negar o acesso a serviços. Para tomar uma decisão de acesso, o PDP leva em consideração a política de segurança definida, informações acerca da requisição do cliente, e informações acerca do token enviado (tais como atributos do cliente obtidos do IdM). A implementação de referência deste GE é o **AuthZForce**[7].

O *PEP Proxy* GE (PEP) atua como ponto de aplicação de políticas de segurança na forma de um proxy-reverso HTTP, com o objetivo de proteger serviços REST. Este componente intercepta as requisições para o serviço, e interage com o IdM para autenticar o token da entidade cliente anexado à requisição, e recuperar informações sobre o token e seu usuário. Em seguida, o PEP interage com o PDP que autoriza ou não o acesso ao serviço. O PEP aplica a decisão do PDP, e em caso positivo, encaminha a requisição ao serviço, e a resposta do serviço ao cliente. A implementação de referência para este GE é o **Wilma**[8]. A plataforma FIWARE suporta ainda a utilização de outros PEPs, como por exemplo PEPs que protejam serviços não-REST ou serviços que utilizem outros protocolos além de HTTP.

Atualmente o Fiware disponibiliza tutoriais e vídeos para as GEs, como o Keyrock[9]. Porém esse material em geral é superficial e aborda a utilização da GE de maneira isolada, sendo insuficiente quando precisamos integrar com outras GEs, e muitas vezes não sendo útil em casos de entraves e bugs.

2.2. Aplicações em contêineres Docker

O Docker[10] é uma plataforma que automatiza a implantação de aplicações dentro de ambientes isolados denominados *containers*, de maneira semelhantes a máquinas virtuais. O principal objetivo dessa tecnologia é proporcionar o empacotamento de aplicações com todas as suas dependências (bibliotecas do sistema operacional, ferramentas, etc.) em uma unidade padronizada para desenvolvimento de software. Dependendo da arquitetura

[4]http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html
[5]http://oauth.net/2/
[6]https://catalogue.fiware.org/enablers/identity-management-keyrock
[7]https://catalogue.fiware.org/enablers/authorization-pdp-authzforce
[8]https://catalogue.fiware.org/enablers/pep-proxy-wilma
[9]https://edu.fiware.org/course/view.php?id=79
[10]https://docs.docker.com

 © 2017 Sociedade Brasileira de Computação

da aplicação, a mesma pode ser desenvolvida em um ou múltiplos contêineres que se comunicam entre si.

Gerenciar o ciclo de vida de um elevado número de containers manualmente se torna inviável, pois vários fatores terão que ser considerados, tais como ordem em que os mesmos serão iniciados (ex.: a aplicação não pode ser iniciada antes do banco), integração entre containers (ex.: o container do IdM deve ter acesso ao endereço IP do PEP, etc.), recriar os containers quando estes forem alterados, etc. O projeto Docker disponibiliza uma ferramenta open-source denominada Docker-Compose, que permite gerenciar múltiplos containers. Os containers e seus respectivos serviços, portas, links e demais particularidades são definidos no arquivo de configurações declarativas *docker-compose.yml*, e usando o comando *docker-compose* é possível gerenciar de forma integrada o processo de build e execução dos containers.

Para atender aos requisitos de segurança da nossa aplicação do estudo de caso, foi necessário projetar um ambiente que oferecesse os GEs Fiware Keyrock, PEP Proxy e AuthZForce. Cada GE disponibiliza imagens Docker que podem ser utilizadas para testes e desenvolvimento. Entretanto, tais GEs são configurados para funcionarem de maneira independente uns dos outros. Dessa forma, é necessário realizar algumas modificações para permitir seu uso de maneira integrada.

3. Autenticação e Autorização com Fiware

Esta seção apresenta os artefatos desenvolvidos para demonstrar a utilização dos componentes de segurança da plataforma FIWARE. Iniciamos apresentando uma visão geral do ambiente de desenvolvimento implantado. Em seguida, demostramos como utilizar o FIWARE para fins de autenticação e autorização de aplicações Web.

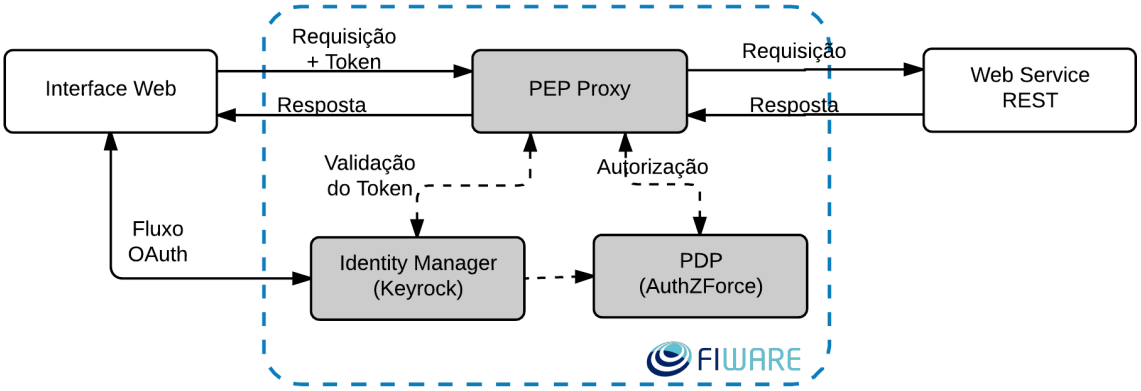


Figura 1. Arquitetura da aplicação de exemplo com suporte a autenticação e autorização.

Antes disso, apresentamos uma aplicação de estudo de caso que será utilizada ao longo deste trabalho para demonstrar a utilização dos mecanismos de segurança da plataforma FIWARE. Na Figura 1 vemos a arquitetura da aplicação de exemplo, que consiste em um componente de backend (*Web Service REST*) que é protegido através do PEP e acessado através de uma *Interface Web*. O funcionamento dos componentes do FIWARE será apresentado nas próximas seções. A aplicação de exemplo desenvolvida em Python, junto com outro exemplo desenvolvido em Java, está disponível no repositório do projeto Smart Metropolis.

 © 2017 Sociedade Brasileira de Computação

3.1. Implantando Ambiente de Desenvolvimento

Para integrar os GE's de segurança utilizamos o Docker-Compose. Foi criado um arquivo **docker-compose.yml** especificando detalhes sobre cada serviço, tais como nome do serviço, local da imagem no Docker Hub, portas TCP utilizadas, Links entre outros GE's, entre outras opções de configuração. Uma visão arquitetural deste ambiente é apresentado na Figura 2.

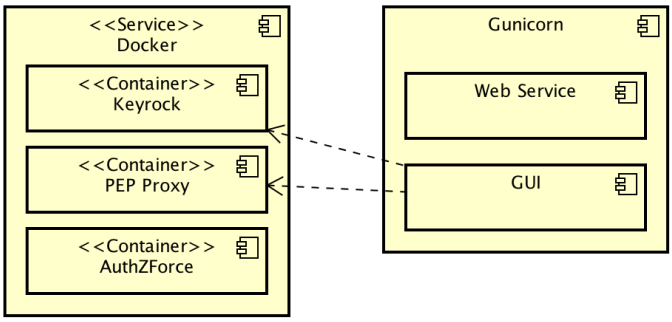


Figura 2. Arquitetura Docker da aplicação de exemplo.

Para cada GE foi criado um container docker. Esses containers são acessados pela GUI da aplicação de exemplo, quando o usuário deseja por exemplo, se autenticar no Keyrock ou requisita um determinado recurso do Web Service passando pelo PEP. Eles também vão se comunicar entre si, por exemplo, quando o PEP deseja verificar a validade de um token no Keyrock, ou consultar a autorização no PDP de um determinado usuário. Também vemos na figura que tanto a GUI como o Web Service estão rodando no servidor Python WSGI, Gunicorn[11].

Entretanto, encontramos uma dificuldade na integração entre os componentes PEP Proxy e Keyrock. Esta dificuldade está relacionada com o *config.js*, um arquivo de configurações que o PEP Proxy requer contendo o endereço IP do Keyrock e da aplicação back-end, além das credencias do PEP, etc [Fiware 2015]. Esse arquivo pode necessitar ser atualizado a cada inicialização do container onde o GE do IdM está sendo executado, pois quando os contaniers são iniciados, é possível que os IP's associados a eles mudem, obrigadando essa constante edição do arquivo de configuração. Para resolver esse problema, optamos por integrar os containers através do parâmetro *link* no **docker-compose.yml**. Dessa forma, o Docker passa a resolver o IP do container onde o Keyrock está sendo executado apartir do nome do container, como mostramos abaixo no trecho do **docker-compose.yml**.

```
keyrock:
    image: fiware/idm:v5.3.0
    hostname: keyrock
    container_name: keyrock
...
pepproxy:
    build: Docker/fiware-pep-proxy
    hostname: pepproxy
    container_name: pepproxy
    expose:
```

[11]http://gunicorn.org

 © 2017 Sociedade Brasileira de Computação

```
11          - "80"
12      ports:
13          - "80:80"
14      links:
15          - authzforce
16          - keyrock
```

Listagem 1. Trecho do arquivo docker-compose.yml envolvendo PEP e IDM

Para que essa atualização automática de IP também seja refletida no arquivo *config.js*, decidimos criar uma imagem personalizada do PEP Proxy, que utiliza a imagem ging/fiware-pep-proxy como referência, e a estende novas funcionalidades, tais como a possibilidade de utilizar um *config.js* com parâmetros especificos do nosso projeto. Essa solução também facilitará futuras modificações na imagem do PEP Proxy, por exemplo, para que ele se adeque a novos requisitos do projeto. No *docker-compose.yml* essa alteração é refletida no uso da instrução *build* em vez de *image*. A instrução *image* informa ao Docker que o mesmo deve procurar essa imagem no Docker Hub, um repositório público de imagens, já no caso da instrução *build* o Docker vai procurar em um diretório dentro do próprio projeto, que nesse caso é o diretório *Docker/fiware-pep-proxy*. Neste diretório há um arquivo *Dockerfile*, que faz referência a imagem do PEP do DockerHub e ao arquivo *config.js*, previamente configurado no diretório do PEP-Proxy. Durante a evolução do projeto outras imagens podem necessitar de customizações, e para facilitar esse gerenciamento optamos por criar um sub-diretório denominado *Docker*, e dentro dele outros sub-diretórios para cada imagem a ser personalizada. Esses sub-diretórios das imagens contém um arquivo *Dockerfile* com a especificação da imagem.

3.2. Autenticando e Autorizando Usuários com o Fiware

De acordo com a documentação da plataforma FIWARE, a integração de uma aplicação com os componentes de segurança podem ser resumidos em seis passos. Inicialmente registramos a aplicação no IdM GE, para permitir o uso de autenticação via o protocolo OAuth2.0. Então implementamos o PEP Proxy entre a Interface Web e o serviço back-end, de modo que Interface Web aponta para o PEP Proxy, e o PEP Proxy aponta para o serviço. Em seguida, implementamos um mecanismo OAuth2 na aplicação (essa etapa permitirá que os usuários façam login na aplicação usando as contas registradas no IdM FIWARE). Após a autenticação do usuário na aplicação o IdM, um token OAuth2 é gerada e representará o usuário toda vez que um recurso tentar ser acessado. Este token deverá ser enviado no cabeçalho HTTP das requisições para o serviço de back-end. Essas requisições são enviadas para o PEP Proxy, que então analisa se o token é válido e, caso o mecanismo de autorização esteja habilitado, se as políticas de autorização permitem o acesso requisitado.

Nossos exemplos englobam três maneiras de se incorporar autenticação e autorização a aplicações Web: (1) *Autenticação com IdM* através do protocolo OAuth2.0; (2) *Autenticação com PEP*, onde credenciais são verificadas pelo PEP; e (3) *Autorização básica como PEP*, onde políticas RBAC são utilizadas para fins de controle de acesso. Esses três exemplos serão apresentados em seguida.

 © 2017 Sociedade Brasileira de Computação

3.2.1. Autenticando Usuários com IdM

Esta seção exemplifica a realização da autenticação da aplicação com o OAuth2[12]. O primeiro passo para se utilizar o serviço de autenticação do Keyrock é efetuar o registro da aplicação no IdM. Neste caso, não se faz necessário o uso do PEP Proxy, e a Interface Web poderia fazer requisições diretamente ao serviço de backend. Ao acessar o endereço do IdM e realizar o login é possível ver na tela inicial uma listagem de aplicações cadastradas e a opção para cadastro de uma nova aplicação. O cadastro é divido em três partes: o registro das informações da aplicação (nome, descrição, etc), a seleção de uma imagem de exibição, e o cadastro dos papéis e permissões associados a aplicação.

Parte das informações passada inclui uma URL de *Callback*, que deve existir na aplicação. Após autenticação, o IdM redireciona a sessão de navegação para onde a Callback URL estará indicando. Ao finalizar o registro de uma aplicação, suas informações serão exibidas (Figura 3). Nesta tela, temos também um *Client Id* e *Client Secret* gerados para a aplicação. Essas informações serão utilizados no momento em que fizer requisições de autenticação para o Keyrock.

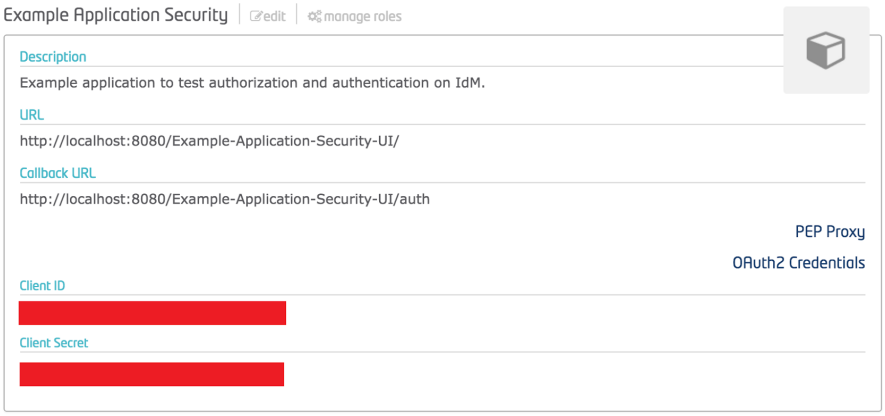


Figura 3. Tela com informações da aplicação registrada.

Uma vez registrada, é necessário alterar a aplicação para que a mesma delegue o processo de autenticação ao Keyrock. Para tal, foram inseridos elementos de segurança que só permitem o envio de requisição ao *back-end* após a autenticação do usuário [13].

Na aplicação de exemplo, adicionamos um botão responsável por requisitar a autenticação caso seja pressionado, redirecionando o usuário para a tela de login no Keyrock. Esse redirecionamento precisa incluir o *Client Id* e *Client Secret* da aplicação. Uma vez autenticado, o Keyrock solicita uma autorização do usuário para a aplicação na qual ele deseja acessar. O Keyrock então retorna um *access-code* para a aplicação, utilizando o endereço cadastrado na URL de Callback. Em seguida, a aplicação realiza uma requisição ao IdM para obter um *access-token*. De posse do *access-token*, a aplicação pode agora seguir seu funcionamento normal, redirecionando o usuário para sua página principal.

[12]https://projetos.imd.ufrn.br/SmartMetropolis-InfraestruturaGroup/Example-Application-Security/tree/master/Python/1-SimplesApplication

[13]https://projetos.imd.ufrn.br/SmartMetropolis-InfraestruturaGroup/Example-Application-Security/blob/master/Python/2-BasicAuthentication/securityapp-ui/web/oauth_fiware.py

© 2017 Sociedade Brasileira de Computação

Uma vez obtido o *access-token*, requisições podem ser realizadas ao IdM para confirmar a autenticação o token recebido, ou para solicitar informações do usuário autenticado no Keyrock. Também é possível utilizar o token em chamadas para acessar outras GE Fiware, que fariam o papel de back-end e usariam o token para confirmar a autenticação do usuário.

3.2.2. Autenticação com PEP

Outra opção de autenticação é a utilização do PEP Proxy para validar o token recebido[14]. De acordo com a Figura 1, o PEP receberá da Interface Web uma requisição de acesso ao serviço back-end com um token OAuth2 do usuário.

Para que seja possível utilizar o PEP na autenticação, é necessário registra-lo no Keyrock. Como é possível ver na Figura 3, existe um link *PEP Proxy*, que é responsável por realizar esse registro, e gerar um *username* e *password* para o PEP. Essas credenciais deverão ser adicionadas no arquivo de configuração do PEP, o *config.js*.

Outra alteração necessária na aplicação foi a inclusão do token no cabeçalho da requisição, conforme exibido na Listagem 2. Neste exemplo[15], temos uma requisiçao GET com o cabeçalho *X-Auth-Token* associado ao *access_token* (linha 6), sendo feita para acessar o *service2/add*, para adicionar um novo *username* a lista.

```
1 @app.route("/add", methods=['GET'])
2 def add():
3   if request.args.get('name') == '':
4     error = 'Fill the name field first!'
5     return render_template('index.html', error=error)
6   headers = {'X-Auth-Token': session['access_token']}
7   url = auth_app.proxy_address + "service2/add/" + request.args.get('
    name')
8   response = requests.get(url, headers=headers)
9   return render_template('index.html', content=response.text)
```

Listagem 2. Trecho de código mostrando a adição do token no cabeçalho da requisição HTTP.

Quando o PEP Proxy recebe a requisição, ele extrai o token do cabeçalho HTTP e envia uma requisição ao IdM para validá-lo. Se o token é válido, o PEP redireciona a solicitação para o serviço back-end.

3.2.3. Autorização Básica com PEP e PDP

Nosso terceiro exemplo[16] envolve o uso de autenticação e autorização por meio do PEP e PDP. Para isso, definimos uma política de autorização RBAC através do IdM que então

[14]https://projetos.imd.ufrn.br/SmartMetropolis-InfraestruturaGroup/Example-Application-Security/tree/master/Python/4-BasicPEPProxy

[15]https://projetos.imd.ufrn.br/SmartMetropolis-InfraestruturaGroup/Example-Application-Security/blob/master/Python/4-BasicPEPProxy/securityapp-ui/web/app.py

[16]https://projetos.imd.ufrn.br/SmartMetropolis-InfraestruturaGroup/Example-Application-Security/tree/master/Python/5-PDP

 © 2017 Sociedade Brasileira de Computação

a registra no PDP (O Keyrock suporta RBAC básica com Verbo + recurso, ou políticas XACML). Considerando a existência de dois usuários, definimos um papel para cada (*Desenvolvedor* e *Visitante*). O papel *Desenvolvedor* tem permissão de realizar um *GET* no recurso */service/list*, enquanto que o papel *Visitante* não tem permissão nenhuma.

Neste exemplo, o PEP Proxy irá verificar se o token é válido, recebendo uma resposta com as informações do usuário, incluindo os papéis associados ao usuário. Em seguida, o PEP Proxy interage com o PDP, enviando os papéis do usuário, a ação da requisição (o verbo HTTP da requisição REST) e o recurso que o usuário está tentando acessar. O PDP verifica os dados recebidos junto à política de autorização cadastrada, e retorna uma decisão para o PEP Proxy.

3.3. Discussão

O tema Cidades Inteligentes tem recebido atenção crescente no Brasil e no mundo, de modo que o desenvolvimento de aplicações seguras é de fundamental importância. Este exercício nos mostrou que o desenvolvimento de solução com mecanismos de segurança não é algo trivial, o que demanda material instrucional e exemplos práticos que possam ser utilizados por desenvolvedores brasileiros.

A experiência adquirida com os componentes de segurança da plataforma FIWARE nos permitiu servir como ponto de apoio à comunidade de pesquisadores vinculados ao projeto Smart Metrópolis da Universidade Federal do Rio Grande do Norte, sendo procurados também por pesquisadores de outras universidades brasileiras. Além disso, um dos resultados alcançados foi a correção de um bug no componente PEP Proxy, que foi aceita pela comunidade de desenvolvedores FIWARE e agora se encontra em seu repositório oficial. O erro foi detectado no log (Listagem 3) dos componentes PEP e PDP. Ao enviar um *token* válido ao AuthZForce, verificamos que tudo estava funcionando corretamente, ele retornava para o PEP a resposta *Permit*, o que é esperado caso o usuário tenha acesso ao recurso solicitado.

```
1 pep-proxy_1 | 2016-12-16 21:16:31.699 - DEBUG: AZF-Client - Decision:
      [ 'Permit' ]
2 pep-proxy_1 | 2016-12-16 21:16:31.699 - ERROR: Root - User access-
    token not authorized: User not authorized in AZF for the given
    action and resource
```

Listagem 3. Trecho de log dos componentes PDP AuthZForce e PEP Wilma

Entretanto, o PEP não liberava o acesso ao recurso solicitado. Com isso, passamos para os arquivos do PEP, e encontramos no arquivo *lib/azf.js* o trecho apresentado na Listagem 4.

```
1 log.debug('Decision: ', decision);
2 if (decision === 'Permit') {
3   success();
4 } else {
5   error(401, 'User not authorized in AZF for the given action and
      resource');
6 }
```

Listagem 4. Trecho de código fonte do componente PEP (arquivo azf.js).

Nesse trecho o código compara a variável *decision*, obtida como resposta do AuthZForce, com a string *Permit*. De acordo com o log do AuthZForce, o resultado da comparação

 © 2017 Sociedade Brasileira de Computação

deveria ser *True*, foi então que fomos verificar o tipo da variável *decision* e vimos que era *object*. Porém, para o javascript quando === é usado, o tipo *object* é diferente do tipo *string*, por isso o resultado da comparação seria sempre *False*, mesmo que o conteúdo fosse igual. A solução que escolhemos foi fazer um parser antes da comparação, para converter a variável *object* para *string*. Depois de encontrarmos a solução para o erro recebido, decidimos criar uma *issue*[17] no repositório oficial do PEP, para que o problema fosse corrigido no código fonte. Com essa contribuição, teremos nossa solução sendo utilizada por todos que desenvolvem aplicações seguras baseada na plataforma FIWARE.

4. Conclusão

Neste trabalho apresentamos um estudo sobre a utilização dos componentes de segurança da plataforma Fiware para a autenticação e autorização de usuários. Desenvolvemos uma aplicação Web de exemplo que demonstra o processo de autenticação através do protocolo OAuth2. Esse processo nos permitiu adquirir um conhecimento mais aprofundado destes componentes. Nossos próximos passos incluem o desenvolvimento de exemplos envolvendo cenários mais complexos de autenticação e autorização utilizando políticas de controle de acesso XACML, além de apoiar o desenvolvimento de aplicações seguras no contexto do projeto Smart Metropolis.

Agradecimentos

Este trabalho foi desenvolvido no contexto do projeto Smart Metropolis conduzido pelo Instituto Metrópole Digital da Universidade Federal do Rio Grande do Norte.

Referências

ANSI (2004). Role based access control. Technical Report ANSI INCITS 359-2004, NIST.

Caragliu, A., Del Bo, C., and Nijkamp, P. (2009). Smart cities in europe. Serie Research Memoranda 0048, VU University Amsterdam, Faculty of Economics, Business Administration and Econometrics.

Fiware (2014). Fiware security chapter architecture. Technical report.

Fiware (2015). Pep proxy - wilma - installation and administration guide. Technical report.

Minerva, R., Biru, A., and Rotondi, D. (2015). Towards a definition of the internet of things (iot). Technical report, IEEE.

Townsend, A. M. (2013). *Smart Cities: Big Data, Civic Hackers, and the Quest for a New Utopia*. W. W. Norton & Company, Inc.

Vogels, W. (2008). Head in the clouds—the power of infrastructure as a service. In *in First workshop on Cloud Computing and in Applications (CCA'08*.

[17]https://github.com/ging/fiware-pep-proxy/issues/36

 © 2017 Sociedade Brasileira de Computação

SF - Salão de Ferramentas

© 2017 Sociedade Brasileira de Computação

CATRACA: uma Ferramenta para Classificação e Análise de Tráfego Escalável Baseada em Processamento por Fluxo*

Martin Andreoni Lopez[1,2], Igor J. Sanz[1], Diogo M. F. Mattos[1], Otto Carlos M. B. Duarte[1] e Guy Pujolle[2]

[1]GTA / PEE-COPPE / UFRJ – Brasil
[2]LIP6/CNRS (UPMC Sorbonne Universités) – França

Abstract. *The late detection of security threats causes a significant increase in the risk of irreparable damages, restricting any defense attempt. In this paper, we propose the CATRACA tool, an efficient online Intrusion Detection and Prevention System implemented as a Network Virtualized Function. The tool is based on a Big Data Streaming processing system, the Apache Spark, and it is deployed on the Open Platform for Network Functions Virtualization (OPNFV), providing an accurate real-time threat-detection service. The tool has two operating modes: on-line and off-line. The detection process includes feature selection and machine learning algorithms to differentiate normal traffic from denial of service (DoS) and probe attacks. Once the data arrive on the platform, the tool enriches these data with associated metadata to improve the detection process. Upon a threat detection, the system is able to promptly react and create blocking firewall rules. Our prototype demonstration shows the reaction and blocking of malicious flows in real time through the injection of traffic from a telecommunications operator. The system has a friendly graphical interface that provides a real-time visualization of the parameters and the attacks that occur in the network.*

Resumo. *A detecção tardia de ameaças à segurança causa um aumento significativo no risco de danos irreparáveis e restringe qualquer tentativa de defesa. Neste artigo, é proposta a ferramenta CATRACA, um sistema eficiente de detecção e prevenção de intrusão em tempo real como uma função virtualizada de rede. A ferramenta baseia-se em um sistema de processamento por fluxos para grandes massas de dados, o Apache Spark, e é implementada na Open Platform for Network Functions Virtualization (OPNFV), fornecendo um serviço eficiente de detecção de ameaças em tempo real e sobre a base de dados histórica. O processo de detecção inclui algoritmos de seleção de características e aprendizado de máquina para diferenciar o tráfego normal dos ataques de negação de serviço e varredura de porta. Uma vez que os dados chegam na plataforma, a ferramenta enriquece os dados com metadados associados para melhorar o processo de detecção. Assim que uma ameaça é detectada, o sistema é capaz de reagir rapidamente e enviar regras de bloqueio para fazer um encadeamento com firewalls. A demonstração da ferramenta mostra a reação e o bloqueio de fluxos maliciosos em tempo real através da injeção de tráfego de uma operadora de telecomunicações. O sistema possui uma interface gráfica amigável que fornece uma visualização em tempo real dos parâmetros e dos ataques que ocorrem na rede.*

1. Introdução

O monitoramento de tráfego é fundamental para manter a estabilidade, a confiabilidade e a segurança das redes de computadores [Hu et al. 2015]. O monitoramento de rede se estende

*Este trabalho foi realizado com recursos do CNPQ, da CAPES, da FAPERJ, e da FAPESP (2015/24514-9, 2015/24485-9, E 2014/50937-1).

© 2017 Sociedade Brasileira de Computação

desde uma simples coleta de estatísticas até uma complexa análise de tráfego de camadas superiores para ajuste de desempenho de rede e depuração de protocolos. As ferramentas atuais de monitoramento de rede, como NetFlow, SNMP, Bro, ou Snort, não atendem às necessidades atuais de velocidade e gerenciamento de grandes domínios de rede, precisando de *hardware* externo para satisfazer essas necessidades. Além disso, muitas dessas ferramentas geram uma enorme quantidade de arquivos que requerem processamento de outro tipo de ferramentas para extrair conhecimento dos dados coletados. Assim, os atuais sistemas de segurança, como o *Security Information and Event Management* (SIEM), não possuem um desempenho satisfatório, enquanto 82% das ameaças de segurança ocorrem em minutos, uma intrusão pode levar até 8 meses para ser detectada [Verizon Enterprise 2016]. É essencial que o tempo de detecção seja o mínimo possível para que a prevenção da intrusão seja eficaz [Wu et al. 2014].

Os ataques de segurança vêm se aperfeiçoando e a simples análise e filtragem de pacotes pelo cabeçalho IP e porta TCP deixaram de ser efetivas, porque o tráfego atacante procura se esconder das ferramentas de segurança falsificando o IP de origem e alterando dinamicamente a porta. Nesse contexto, uma alternativa promissora para classificar tráfegos e detectar ameaças é o uso de técnicas de aprendizado de máquina. Essas técnicas são adequadas para grandes massas de dados, pois, com mais amostras para treinar o classificador, os métodos tendem a terem maior acurácia [Mayhew et al. 2015]. No entanto, com grandes quantidades de dados, os métodos de aprendizado de máquina podem apresentar altas latência devido ao longo tempo de processamento. Essa alta latência é uma restrição para métodos de aprendizado de máquina que devem analisar os dados e detectar ameaças o mais rápido possível. Neste contexto, o processamento de fluxo em tempo real permite a análise imediata de diferentes tipos de dados e, consequentemente, beneficiam o monitoramento de tráfego para detecção de ameaças de segurança. Plataformas de processamento distribuído de fluxo de código aberto vêm recentemente sendo propostas para processar grandes massas de dados em fluxos com baixa latência. Para tanto, essas plataformas são a base para o desenvolvimento de aplicativos sob-medida para cada caso.

Este artigo propõe a ferramenta CATRACA[1] que utiliza a tecnologia de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV) e que possui uma infraestrutura que combina virtualização, computação em nuvem e processamento de fluxo distribuído para monitorar o tráfego de rede e detectar ameaças. O objetivo é fornecer uma ferramenta de detecção de ameaças precisa, escalável e em tempo real, detecção de ameaças em tempo real de forma precisa e escalável, capaz de atender picos de uso, fornecendo uma alta qualidade de serviço. O monitoramento de tráfego e a detecção de ameaças como uma função de rede virtualizada apresentam duas vantagens principais: a capacidade de se autoadaptar a diferentes volumes de tráfego e a flexibilidade de instalação e migração de sensores na rede para reduzir a latência no monitoramento [Andreoni Lopez et al. 2014]. A ferramenta captura e processa grandes volumes de dados, as técnicas de aprendizado de máquinas classificam o tráfego em normal ou ameaça. Por fim, o conhecimento extraído dos fluxos é apresentado em uma interface web.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 discute a arquitetura e o funcionamento da ferramenta. A Seção 4 descreve a demonstração da ferramenta a ser realizada. Por fim, a Seção 5 conclui o trabalho.

2. Trabalhos Relacionados

Existem algumas propostas que utilizam a ferramenta de processamento de fluxo *Storm* para realizar a detecção de anomalias em tempo real. Du *et al.* utilizam a ferramenta *Flume* e

[1] A ferramenta, a documentação e a licença estão disponíveis em https://gta.ufrj.br/catraca.

 © 2017 Sociedade Brasileira de Computação

Storm para fazer o monitoramento de tráfego a fim de detectar anomalias. A proposta é fazer a detecção através do algoritmo de k-NN [Du et al. 2014]. O artigo apresenta alguns resultados de desempenho, mas carece de avaliação da acurácia da detecção e a ferramenta não recebe dados de múltiplas fontes. O trabalho de Zhao *et al.* usa os *softwares Kafka* e *Storm*, assim como o trabalho anterior, para a detecção de anomalias em redes [Zhao et al. 2015],caracterizando os fluxos no formato *NetFlow*. He *et al.* propõem uma combinação das plataformas de processamento distribuído *Hadoop* e *Storm*, em tempo real, para a detecção de anomalias. Nessa proposta, uma variante do algoritmo k-NN é usada como algoritmo de detecção de anomalias [He et al. 2015]. Os resultados mostram um bom desempenho em tempo real, porém sem utilizar nenhum processo de reação e prevenção de ameaças. Mylavarapu *et al.* propõem usar o *Storm* como plataforma de processamento de fluxos na detecção de intrusão [Mylavarapu et al. 2015].

A proposta Stream4Flow utiliza o Apache Spark com a pilha ElasticStack para fazer monitoramento de redes [Jirsik et al. 2017]. O protótipo serve como visualização dos parâmetros da rede. Stream4Flow, no entanto, não possui nenhuma inteligência para realizar detecção de anomalias. Dos Santos *et al.* utilizam uma combinação do Snort IDS e OpenFlow para criar o Of-IDPS. O Snort IDS é usado como uma ferramenta de detecção, enquanto as ações do OpenFlow realizam a mitigação ou a prevenção de ataques detectados [Santos et al. 2014]. Uma evolução do Of-IDPS foi proposta para desenvolver um sistema de computação autônoma para criar automaticamente regras de segurança em comutadores de Rede Definida por Software (SDN) [dos Santos et al. 2016]. As regras são criadas aplicando um algoritmo de aprendizado de máquina para alertas do Snort IDS e *logs* do OpenFlow.

O projeto OpenSOC: The Open Security Operations Center [Santos 2015] é um projeto de desenvolvimento colaborativo que integra diversos *softwares* de código aberto que objetiva uma ferramenta de análise de segurança extensível e escalável. Assim, o OpenSOC é um arcabouço de segurança analítica para monitorar grandes massas de dados usando processamento distribuído de fluxos. O OpenSOC foi descontinuado e deu origem ao projeto Apache Metron [Apache Software Foundation 2017] que é uma evolução do OpenSOC e propõe uma nova arquitetura que visa facilitar a adição de novas fontes de dados, e explorar melhor o paralelismo da ferramenta Storm.

A ferramenta CATRACA proposta, assim como o Metron, também foi inspirada no OpenSOC e objetiva a monitoração de grandes volumes de dados usando processamento de fluxo. A ferramenta CATRACA é implementada como uma função virtualizada de rede (VNF) no ambiente Open Plataform for Network Function Virtualization (OPNFV). CATRACA foca em técnicas de captura dos pacotes em tempo real, de seleção de características, de aprendizado de máquina e possui um mecanismo de ação para bloqueio imediato de fluxos considerados maliciosos. Assim, a ferramenta CATRACA atua como uma função virtualizada de rede de detecção e prevenção de intrusão que reportam resumos de fluxos e que pode ser encadeada com outras funções virtualizadas de rede conforme definido nos padrões de encadeamento de funções de rede (*Service Function Chainning* - SFC) e cabeçalhos de serviço de rede (*Network Service Header* - NSH). Além disso, o enriquecimento dos dados com metadados de geolocalização dos endereços de rede, a marcação de tempo e a correlação de eventos permitem a melhor compreensão da origem e o destino dos ataques através de uma interface simples para a visualização do conhecimento inferido a partir dos dados.

3. A Arquitetura da Ferramenta CATRACA

A arquitetura da ferramenta CATRACA, mostrada na Figura 1, é composta por três camadas: captura, processamento e visualização. A **camada de captura**, é responsável pela cap-

© 2017 Sociedade Brasileira de Computação

tura dos pacotes, através do espelhamento de tráfego, pela biblioteca *libpcap*. Uma aplicação, escrita em Python e baseada no *flowtbag*[2], abstrai os pacotes em fluxos, que são definidos como uma sequência de pacotes que possuem a mesma quíntupla IP de origem, IP de destino, porta de origem, porta de destino e protocolo, durante uma janela de tempo. Ao todo, 46 características de fluxo são extraídas e publicadas em uma fila de um serviço produtor/consumidor de mensagens *Kafka*, com baixa latência, para serem consumidos pela camada de processamento.

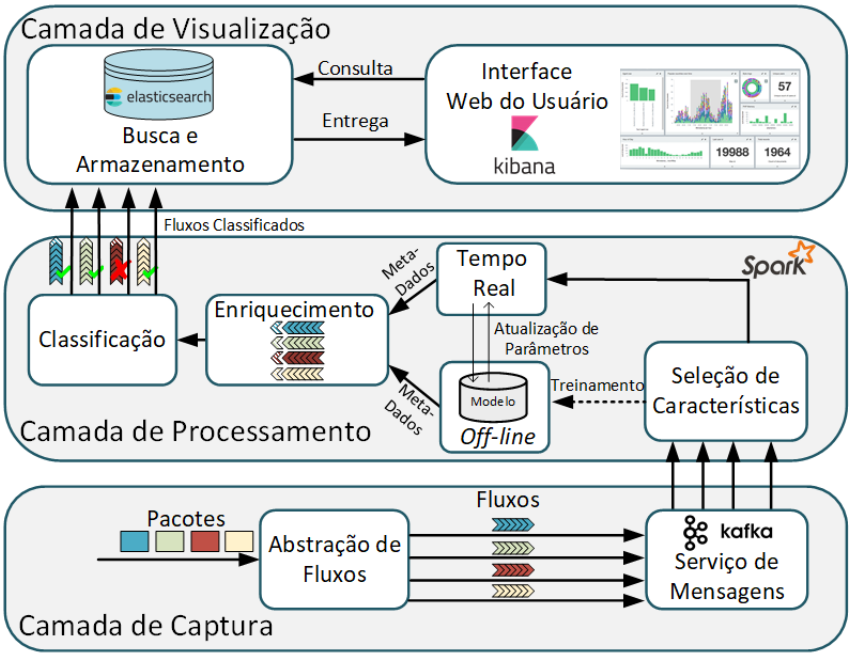


Figura 1: A arquitetura em camadas da ferramenta CATRACA: a camada de captura, a camada de processamento e a camada de visualização.

A **camada de processamento** é instanciada em uma nuvem dedicada para a classificação que possui o *Spark* como núcleo principal de processamento. A plataforma `Spark` foi escolhida entre as diferentes plataformas de processamento de fluxos por possuir o melhor comportamento em relação a tolerância a falhas [Andreoni Lopez et al. 2016], tornando a ferramenta CATRACA mais robusta em caso de falha de nós de processamento. O *Spark* é implementado em um aglomerado de máquinas no modelo mestre/escravo, em que os escravos possuem a capacidade de expansão e redução de recursos, tornando o sistema escalável. Uma vez que os fluxos chegam na camada de processamento, um algoritmo seleciona as características mais importantes para a classificação dos ataques [Andreoni Lopez et al. 2017a]. Na etapa de processamento, os dados são enriquecidos através de distintas informações como a localização geográfica dos IPs analisados. Em seguida, os fluxos são classificados em maliciosos ou bem-comportados através do algoritmo de aprendizado baseado em árvores de decisão.

Finalmente, a **camada de visualização** é implementada utilizando a pilha de *software Elastic*[3]. A pilha *Elastic* permite a visualização personalizada de eventos em tempo real. Assim, a saída da camada de processamento é enviada ao componente *elasticsearch* que fornece um rápido serviço de busca e armazenamento. A interface do usuário, que é executada no componente *Kibana* da mesma pilha de *software*, comunica-se com *elasticsearch* através de consultas.. O *Kibana* processa os resultados das consultas e gera a visualização final.

[2]Estatisticas de fluxos *flowtbag*: https://dan.arndt.ca/projects/flowtbag/
[3]*Elastic Stack*: https://www.elastic.co/

© 2017 Sociedade Brasileira de Computação

3.1. A Classificação de Tráfego Proposta

A classificação inicia com o pré-processamento de seleção das características mais importantes dos fluxos [Andreoni Lopez et al. 2017a]. Em seguida, a ferramenta pode operar em modo de tempo diferenciado ou de tempo real. A classificação de tráfego em tempo diferenciado (*off-line*) consiste no processamento de mini-lotes da plataforma *Spark*. Nesse modo, conjuntos de dados de grandes volumes são carregados em um sistema de arquivos distribuído, como por exemplo o *Hadoop Distributed File System* (HDFS). O conjunto de dados é separado em um conjunto de treinamento e um conjunto de teste em uma proporção de 70% para o treinamento e 30% para o teste. Logo, o *Spark* realiza o processamento através da técnica *map-reduce*. Um algoritmo de aprendizado de máquinas é treinado para obter o modelo de classificação.

O algoritmo de classificação por árvore de decisão é implementado na ferramenta, devido à sua velocidade de treinamento aliada à sua alta acurácia e precisão [Lobato et al. 2016]. A árvore de decisão é um algoritmo guloso que executa um particionamento binário recursivo do espaço de recursos. Cada folha é escolhida selecionando a melhor separação de um conjunto de divisões possíveis, para maximizar o ganho de informações em um nó da árvore. A divisão em cada nó da árvore é escolhida a partir do $argmax_d GI(CD, d)$, onde $GI(CD, d)$ é o ganho de informação quando uma divisão d é aplicada a um conjunto de dados CD. O ganho de informação GI da ferramenta CATRACA é a impuridade de Gini, $\sum_{i=1}^{C} f_i(1 - f_i)$, que indica o quão separadas estão as classes, em que f_i é a frequência da classe i em um nó e C é o número exclusivo de classes. Uma vez obtido, o modelo é armazenado no sistema de arquivos e carregado para ser usado no modo de classificação de tráfego em tempo real (*on-line*). No entanto, é também possível validar o modelo com o conjunto de treinamento de 30% obtido anteriormente.

A Tabela 1 mostra a matriz de confusão da avaliação do conjunto de dados de segurança [Lobato et al. 2016]. A matriz de confusão especifica claramente a taxa de falsos positivos e outras métricas de cada classe no conjunto de dados de teste. As linhas representam os elementos que de fato pertencem à classe real e as colunas os elementos que foram classificados como pertencentes à classe. Portanto, os elementos na diagonal em destaque dessa matriz representam o número de elementos que são corretamente classificados. A Tabela 2 mostra as

Tabela 1: Matriz de Confusão na Árvore de Decisão.

	Normal	**DoS**	**Varredura**
Normal	64134	0	1550
DoS	0	2557067	0
Varredura	13	0	31254

Tabela 2: Métricas de Avaliação *Off-line*

Métrica	**Acurácia**	**Erro**	**Precisão**	***Recall***	**F1 *Score***
Valor	99.94%	0.0005 %	99.94%	99.94%	99.94%

métricas complementares à matriz de confusão. Observando os valores de Acurácia e Precisão é possível ver o bom desempenho do algoritmo de árvores de decisão na classificação *off-line*. Comparando-se as Tabelas 1 e 2, verifica-se que o algoritmo apresentou uma alta acurácia praticamente em todas as classes, com uma baixa taxa de falsos positivos. Outra forma de ver a taxa de falsos positivos é observar os valores que estão fora da diagonal principal na Tabela 1.

© 2017 Sociedade Brasileira de Computação

O treinamento do sistema é realizado a partir de um conjunto de dados criado no laboratório GTA/UFRJ, contendo sete tipos de negação de serviço (DoS) e nove tipos de varredura de portas [Lobato et al. 2016]. O conjunto de dados contém cerca de 95 GB de dados de captura de pacotes, resultando em 154.187 fluxos.

Após a obtenção do modelo de classificação a partir da base histórica, pode-se avaliar a acurácia da ferramenta com dados chegando em tempo real. A operação da ferramenta CATRACA em tempo real utiliza o módulo *streaming* da plataforma *Spark*. Assim, os pacotes abstraídos em fluxos, capturados em diferentes máquinas virtuais na nuvem, são processados conforme chegam à plataforma *Spark*. Quando um fluxo chega na ferramenta de detecção, é resumido em características utilizando o algoritmo de seleção [Andreoni Lopez et al. 2017a], a fim de reduzir o tempo de processamento. Assim, o vetor de características selecionadas é avaliado no modelo obtido no processamento *off-line*. Após a extração dos dados analíticos dos fluxos, os resultados são armazenados em um banco de dados para uma análise posterior. Os dados armazenados possuem as informações coletadas durante a detecção de ameaças e podem ser reprocessados *offline* para calcular os parâmetros a serem usados no modelo em tempo real. Para tornar o sistema mais preciso, há uma retroalimentação, uma vez que os parâmetros calculados *offline* com dados históricos ajustam o modelo de processamento para a detecção de ameaças em tempo real.

3.2. A Visualização em Tempo Real dos Dados Enriquecidos

A visualização dos dados enriquecidos ocorre através de uma interface *web* simples e amigável para permitir ao usuário monitorar as diferentes configurações de parâmetros da rede em tempo real. O visualizador de código aberto *Kibana*, um componente da pilha *Elastic*, foi utilizado para o desenvolvimento da interface *web*, pois permite a visualização dos dados de forma simples e rápida aliada ao desempenho de processamento de consultas com grandes volumes de dados com baixa latência. A Figura 2 mostra alguns dos diferentes cenários que podem ser visualizados no painel de controle, como, por exemplo, as portas de destino/origem mais acessadas, os endereços IP destino mais utilizados, o tamanho médio dos fluxos nas direções de ida e volta, a quantidade de fluxos analisados, entre outros. Vale ressaltar a visualização dos ataques em andamento através de um mapas que retratam a origem, o destino e número de ocorrências. Isso é possível devido ao enriquecimento dos dados através da correlação com metadados de geolocalização no módulo de processamento. Assim, tanto os dados como as ameaças são visualizados em tempo real. Além disso, todos os dados são armazenados com estampa de tempo, permitindo o processamento dos dados através de séries temporais.

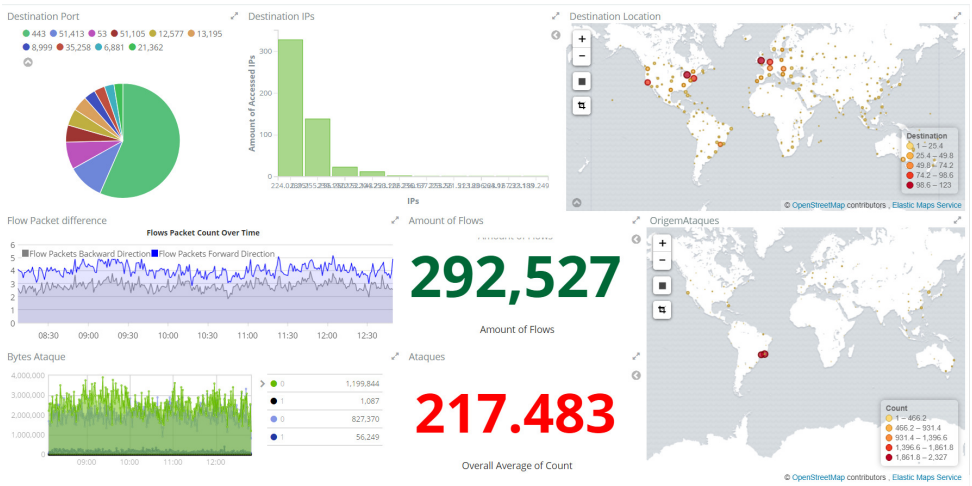


Figura 2: Visualização do painel de controle da ferramenta.

 © 2017 Sociedade Brasileira de Computação

4. A Demonstração do Ferramenta CATRACA

A demonstração da ferramenta CATRACA é efetuada pela análise e a classificação em tempo real do tráfego que atravessa uma máquina virtual, que será executada em um computador no SBSeg[4]. A captura dos pacotes será enviada ao nó central, nó *master*, localizado na nuvem OPNFV no Grupo de Teleinformática e Automação (GTA/UFRJ), no Rio de Janeiro. O nó presente no GTA é responsável por receber os fluxos provenientes do *Kafka*, e gerenciar distribuí-los para os nós escravos do aglomerado *Spark*.

A demonstração evidencia a acurácia da ferramenta quando é capturado tráfego normal e quando é injetado tráfego malicioso através de uma interface de rede na máquina virtual monitorada. O classificador é previamente treinado com os parâmetros do modelo da árvore de decisão. Um conjunto de dados de uma operadora de telecomunicações [Andreoni Lopez et al. 2017b] da cidade do Rio de Janeiro é usado para gerar tráfego na máquina virtual. A topologia da demonstração é mostrada na Figura 3.

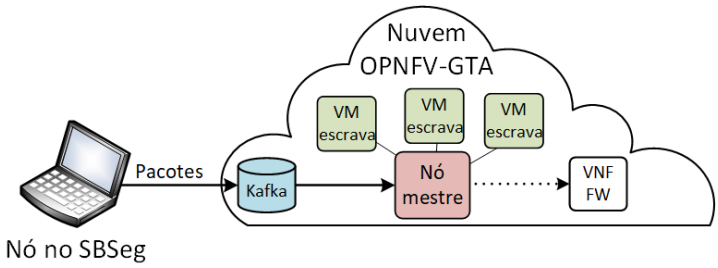


Figura 3: Topologia do experimento a ser realizado no salão de ferramentas do SBSeg.

5. Conclusão

A ferramenta CATRACA propôs uma função virtualizada de rede para detecção de intrusão em tempo real e em tempo diferenciado. A detecção de ameaças é realizada usando algoritmos de aprendizagem em máquina executados em uma plataforma de processamento de fluxo. A ferramenta proposta executa a análise de ameaças sobre o tráfego entrante em tempo real ou sobre uma base histórica em tempo diferenciado. Para aumentar a velocidade de análise e melhorar a eficiência em grandes volumes de dados, a ferramenta utiliza um algoritmo rápido, não supervisionado, para seleção das características mais importantes e, consequentemente, redução de dimensionalidade dos dados de entrada. O algoritmo de seleção de características reduz até 100 vezes do tempo de processamento, quando comparado com algoritmos tradicionais de seleção e redução de características [Andreoni Lopez et al. 2017a]. A ferramenta executa sobre plataforma de código aberto OPNFV como uma função virtual de rede e exibe o conhecimento extraído dos dados enriquecidos através de uma interface gráfica amigável para a visualização de diferentes análises e da localização geográfica da origem e destino das ameaças em tempo real.

A ferramenta pode ser obtida em `http://www.gta.ufrj.br/catraca`, onde consta o manual do usuário, que detalha os procedimentos de instalação e uso da ferramenta, a documentação, que permite compreender o projeto de *software* e obter mais detalhes sobre o código da ferramenta, além de outras informações úteis. Como trabalhos futuros, pretende-se oferecer diferentes algoritmos de aprendizado de máquina e monitorar tráfego em redes reais.

[4]A demonstração requer um computador com acesso à Internet para ser configurado com os *softwares* de teste.

© 2017 Sociedade Brasileira de Computação

Referências

Andreoni Lopez, M., Figueiredo, U. d. R., Lobato, A. G. P. e Duarte, O. C. M. B. (2014). BroFlow: Um Sistema Eficiente de Detecção e Prevenção de Intrusão em Redes Definidas por Software. *XII WPerformance (XXXIV CSBC)*, páginas 1919–1932.

Andreoni Lopez, M., Lobato, A. G. P. e Duarte, O. C. M. B. (2016). A Performance Comparison of Open-Source Stream Processing Platforms. Em *IEEE GLOBECOM*, páginas 1–6, Washington, USA. IEEE.

Andreoni Lopez, M., Lobato, A. G. P., Mattos, D. M. F., Alvarenga, I. D., Duarte, O. C. M. B. e Pujolle, G. (2017a). Um Algoritmo Não Supervisionado e Rápido para Seleção de Características em Classificação de Tráfego. Em *XXXV SBRC'2017*, Belém- Pará, PA,.

Andreoni Lopez, M., Silva, R., Alvarenga, I., Mattos, D. e Duarte, O. C. M. B. (2017b). Coleta e Caracterização de um Conjunto de Dados de Tráfego Real de Redes de Acesso em Banda Larga. Em *XXII WGRS'17*.

Apache Software Foundation (2017). Apache Metron. `https://cwiki.apache.org/confluence/display/METRON/About+Metron`. Acessado em 29/08/2017.

dos Santos, L. A. F., Campiolo, R., Monteverde, W. A. e Batista, D. M. (2016). Abordagem autonômica para mitigar ciberataques em LANs. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2016*.

Du, Y., Liu, J., Liu, F. e Chen, L. (2014). A real-time anomalies detection system based on streaming technology. Em *Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 2, páginas 275–279. IEEE.

He, G., Tan, C., Yu, D. e Wu, X. (2015). A real-time network traffic anomaly detection system based on storm. Em *Proceedings - 2015 7th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2015*, volume 1, páginas 153–156.

Hu, P., Li, H., Fu, H., Cansever, D. e Mohapatra, P. (2015). Dynamic defense strategy against advanced persistent threat with insiders. Em *IEEE Conference on Computer Communications (INFOCOM)*, páginas 747–755.

Jirsik, T., Cermak, M., Tovarnak, D. e Celeda, P. (2017). Toward Stream-Based IP Flow Analysis. *IEEE Communications Magazine*, 55(7):70–76.

Lobato, A. G. P., Andreoni Lopez, M. e Duarte, O. C. M. B. (2016). Um Sistema Acurado de Detecção de Ameaças em Tempo Real por Processamento de Fluxos. Em *SBRC'2016*, páginas 572–585, Salvador, Bahia.

Mayhew, M., Atighetchi, M., Adler, A. e Greenstadt, R. (2015). Use of machine learning in big data analytics for insider threat detection. Em *IEEE MILCOM*, páginas 915–922.

Mylavarapu, G., Thomas, J. e TK, A. K. (2015). Real-Time Hybrid Intrusion Detection System Using Apache Storm. Em *17th International Conference on High Performance Computing and Communications*, páginas 1436–1441. IEEE.

Santos, L. A. F., Campiolo, R. e Batista, D. M. (2014). Uma Arquitetura Autonômica para Detecçao e Reação a Ameaças de Segurança em Redes de Computadores. Em *III WoSiDA'14*, páginas 1–4.

Santos, O. (2015). Big data analytics and netflow. Em *Network Security with NetFlow and IPFIX: Big Data Analytics for Information Security*. Acessado em 29/08/2017.

Verizon Enterprise (2016). 2016 Data Breach Investigations Report. http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf.

Wu, K., Zhang, K., Fan, W., Edwards, A. e Yu, P. S. (2014). RS-Forest: A Rapid Density Estimator for Streaming Anomaly Detection. Em *IEEE International Conference on Data Mining (ICDM)*, páginas 600–609.

Zhao, S., Chandrashekar, M., Lee, Y. e Medhi, D. (2015). Real-time network anomaly detection system using machine learning. Em *11th International Conference on the Design of Reliable Communication Networks (DRCN)*, páginas 267–270. IEEE.

 © 2017 Sociedade Brasileira de Computação

Provisionamento automatizado de servidores para competições de segurança da informação

Lucas Magalhães[1], **Antônio Carlos F. Petri**[1], **Gabriel de S. Alves**[1], **Cesar Augusto C. Marcondes**[1], **Paulo Matias**[1]

[1]Departamento de Computação
Universidade Federal de São Carlos (UFSCar)
São Carlos, SP – Brasil

whoisroot@openmailbox.org, {falcaopetri, g4briel.4lves}@gmail.com,
{marcondes, matias}@dc.ufscar.br

Abstract. *Promoting Capture-The-Flag (CTF) competitions requires large operational costs, due to the number of participants scale and problems computing requirements. In problems that involve server exploitation, it is important to provide guarantees that each participant solution do not interfere with others. Thus, to save resources, we propose an automated provisioner that allocates LXD containers to competitors that have achieved a minimum score, in addition we integrate the provisioner to the OpenStack API. Finally, the solution is fully operational at UFSCar Private Cloud and we plan to adopt it during the 2017 Pwn2Win International CTF.*

Resumo. *Promover competições de Capture-The-Flag (CTF) pode envolver um grande custo operacional, pois a complexidade da infraestrutura escala com o número de participantes e de problemas propostos. Em problemas que envolvam explorar servidores, é necessário garantir que as ações de um participante não impactem na resolução de outros. Para economizar recursos, desenvolvemos um provisionador automático que aloca contêineres LXD apenas para competidores que obtenham previamente uma pontuação mínima em problemas que não envolvam explorar servidores, além de integrar-se à API do OpenStack. A solução encontra-se totalmente operacional na nuvem privada da UFSCar, e pretendemos adotá-la na edição de 2017 do CTF internacional Pwn2Win.*

1. Motivação

Um elemento crítico de uma estratégia efetiva de cibersegurança é dispor de pessoas treinadas nas questões mais recentes relacionadas à segurança da informação. Alguns países, como os Estados Unidos, têm se preparado de forma expressiva contra ataques cibernéticos [ABI Research 2015], no entanto, ainda há uma limitação na quantidade e na qualidade dos profissionais, especialmente quando consideramos habilidades mais sofisticadas, tais como segurança por construção, inteligência de ameaças, ou perícia forense após um ataque [Evans and Reeder 2010].

Com o objetivo de reduzir a carência do mercado em profissionais de cibersegurança, empresas, escolas, universidades e instituições militares ao redor do mundo têm promovido competições de "capture a bandeira" (ou CTF, do inglês *Capture-The-Flag*) para incentivar que mais profissionais envolvam-se com tópicos de cibersegurança.

© 2017 Sociedade Brasileira de Computação

Competições de CTF geralmente são planejadas para proporcionar aos participantes experiência em um amplo espectro de problemas relacionados à segurança de computadores, bem como em técnicas de ataque e defesa que poderiam ser aplicadas em situações reais. Essas competições também podem servir como uma alternativa de baixo custo para recrutar profissionais altamente habilidosos para preencher vagas específicas de emprego. Dentre as habilidades necessárias para participar de CTFs, citamos engenharia reversa, exploração de falhas em binários e em aplicativos *web*, análise forense, criptanálise e programação.

Há dois estilos de competição de CTF: *ataque/defesa* e *jeopardy*. Em competições de *ataque/defesa*, cada equipe recebe uma máquina para defender, conectada a uma rede isolada. A pontuação das equipes é dada com base tanto no sucesso em defender essa máquina como no êxito em atacar as máquinas de outras equipes. Competições de *jeopardy* são mais comuns e geralmente envolvem múltiplas categorias de problemas, cada qual contendo uma variedade de desafios com diferentes pontuações e níveis de dificuldade. Resolver corretamente o problema leva o competidor a obter uma bandeira (*flag*), que deve ser submetida para uma plataforma a fim de computar pontos no placar. As equipes tentam obter a maior pontuação possível ao longo da competição (*e.g.*, num período de 24 horas), mas geralmente não atacam diretamente uma à outra. Ao contrário de uma corrida, esse estilo de jogo estimula as equipes a dedicar tempo para estudar os desafios, e prioriza a quantidade de submissões corretas em vez da velocidade com que são submetidas.

Dentre os desafios propostos em competições de *jeopardy*, há um subconjunto que disponibiliza um arquivo para que os participantes copiem e resolvam o problema localmente em seus computadores. Geralmente enquadram-se nesse grupo os desafios das categorias de engenharia reversa, análise forense e criptanálise. Muitas vezes, há também um subconjunto de problemas que, apesar de requerer ataques a um servidor, não solicitam que o participante tome controle por completo nem altere o estado desse servidor. Por exemplo, para resolver alguns tipos de problema de exploração de falhas em aplicativos *web* vulneráveis a injeção de SQL (*Structured Query Language*), basta realizar consultas somente de leitura (*e.g.,* `SELECT`). Nesse caso, é possível configurar o servidor de forma que a base de dados mantenha-se congelada e não possa ser alterada pelos participantes.

No entanto, a imensa maioria dos problemas de exploração de binários e de aplicativos *web* exige que os participantes executem ações que alteram o estado do servidor ao longo do processo de resolução. Nesses casos, é necessário prover uma infraestrutura separada para cada equipe, a fim de impedir que as demais equipes causem indisponibilidade ou tomem qualquer ação que dificulte a resolução do problema. Por exemplo, em um desafio que envolva tomar controle por completo de um servidor compartilhado entre diversas equipes, a primeira equipe a resolver o problema poderia remover as bandeiras desse servidor, impedindo que outras equipes concluíssem a resolução. A interferência entre diversas equipes também poderia ocorrer de forma acidental, pela simples disputa entre acessos de escrita originados por equipes distintas de forma concorrente.

Para resolver esse problema, diversas competições têm adotado uma infraestrutura baseada em contêineres, na qual cada equipe recebe um contêiner diferente para um determinado desafio. Contêineres são alternativas leves às máquinas virtuais que, no caso do Linux, são implementadas usando os recursos de espaço de nomes e de grupos de controle do sistema operacional. Apesar do núcleo do sistema operacional apresentar uma área de ataque maior que a de um virtualizador, essa tecnologia vem se mostrando

 © 2017 Sociedade Brasileira de Computação

competitiva em diversos cenários de aplicação [Bernstein 2014].

2. Proposta e Principais Funcionalidades

Este trabalho propõe uma ferramenta de software livre capaz de provisionar uma infraestrutura baseada em contêineres para uma competição de CTF em uma nuvem OpenStack pública ou privada disponibilizando, ainda, as seguintes inovações:

- Nossa ferramenta gerencia contêineres sem depender de soluções internamente complexas, como a plataforma Kubernetes. Apesar de estável e de fácil utilização, a plataforma Kubernetes tem demonstrado possuir uma área de ataque demasiadamente ampla para ser adotada em CTFs nos quais os competidores possam vir a tomar controle por completo de alguns dos contêineres. Em competições nas quais foi adotada, ocorreram problemas devido a configurações padrão inseguras, tais como permissão de acesso à API de dentro dos contêineres [Eastes 2017]. Apesar de existirem esforços recentes em prover recursos e orientações de segurança para a plataforma Kubernetes, optamos por uma solução mais simples e menos sujeita a erros. Gerenciamos os servidores por meio de comandos enviados via SSH (*Secure Shell*) e utilizamos somente contêineres providos pelo LXD, projeto cujo foco é justamente ser seguro por padrão, fornecendo uma alternativa mais leve a virtualizadores [Ernst et al. 2016, Ferreira 2017]. Com isso, perdemos alguns recursos da plataforma Kubernetes, como o provimento de infraestruturas com alta disponibilidade (por meio de réplicas do nó mestre). No entanto, argumentamos que esse recurso não é tão importante em CTFs, podendo ser substituído pelo monitoramento da infraestrutura.
- Provisionamos contêineres apenas para as equipes que atingirem uma determinada pontuação mínima nos problemas que não dependam de contêineres, ou seja, aqueles que possam ser resolvidos pelos participantes localmente em seus computadores, ou que tenham sido elaborados de forma a garantir que todas as operações efetuadas em um servidor sejam somente de leitura. Desta forma, economizamos recursos computacionais que seriam alocados para equipes que se inscreveram mas não possuem a intenção de participar seriamente da competição. CTFs divulgados no sítio CTFtime[1], que reúne e avalia por pares diversas competições internacionais, costumam receber centenas de inscrições de equipes. Em edições anteriores da competição Pwn2Win, observamos que apenas pouco mais da metade das equipes inscritas chega a resolver ao menos um dos problemas propostos [Bertochi 2016].
- Nosso provisionador integra-se à *NIZKCTF*, uma plataforma abertamente auditável e resistente a ataques contra seu mecanismo de placar [Matias et al. 2017]. O provisionador monitora continuamente o placar da plataforma *NIZKCTF* para determinar se uma equipe possui pontuação mínima para ter acesso aos contêineres. Além disso, contêineres associados a problemas que já tenham sido resolvidos por uma equipe são automaticamente destruídos, por não serem mais necessários.

3. Importância da Segurança da Infraestrutura de CTFs

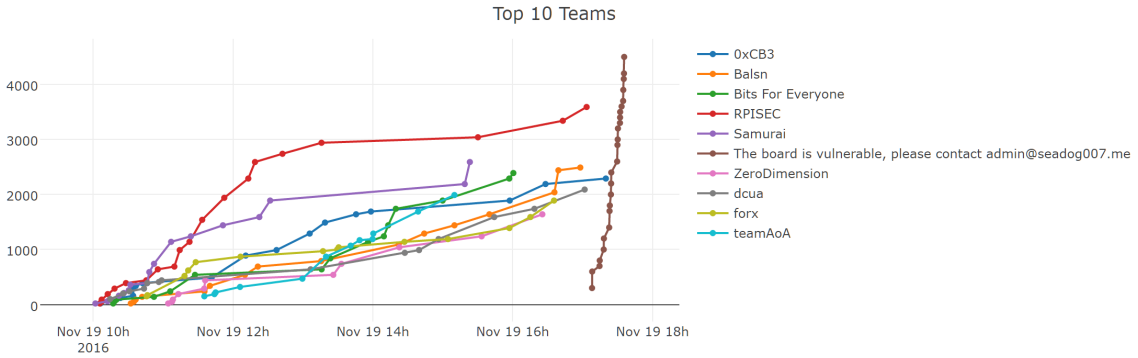
Uma preocupação constante para organizadores de CTFs é proteger de ataques a infraestrutura da competição, que está sujeita às mesmas falhas de segurança que qualquer

[1] `https://ctftime.org`

© 2017 Sociedade Brasileira de Computação

sistema computacional. Devido à elevada competitividade entre os participantes, é comum encontrar equipes atacando a infraestrutura em vez dos desafios propostos pela competição.

Um exemplo dessa situação ocorreu durante o RC3 CTF de 2016. Por um momento durante essa competição, uma equipe denominada "*The board is vulnerable, please contact admin@seadog007.me*" apareceu no placar com 4500 pontos [Yu 2016]. A Figura 1 mostra o registro desse fato. Felizmente, a intenção do participante que explorou a falha era apenas reportar a vulnerabilidade para os administradores. Entretanto, era perfeitamente possível explorá-la para fins maliciosos.



Place	Team	Score
1	The board is vulnerable, please contact admin@seadog007.me	4500
2	RPISEC	3590
3	Samurai	2590

Figura 1. Durante o RC3 CTF de 2016, um competidor explorou falhas no mecanismo de placar para reportar a vulnerabilidade aos administradores da competição.

Um dos componentes de uma competição de CTF mais visados em ataques é a plataforma central que recebe submissões de respostas (bandeiras) e gerencia o placar. Visando proteger esse mecanismo nas competições em que for adotada, a ferramenta apresentada neste artigo foi desenvolvida para integrar-se à *NIZKCTF*, uma plataforma para CTFs que propomos recentemente [Matias et al. 2017], que baseia sua segurança em provas criptográficas em vez de confiar na segurança do software ou dos servidores que o hospedam. Em lugar de submeter diretamente as respostas (bandeiras) para a plataforma, o competidor envia uma prova de conhecimento zero de que obteve a resposta correta. Essa prova criptográfica não contém nenhuma informação que possa auxiliar outras equipes a resolver o problema, e é tornada pública para que qualquer observador externo possa auditar a competição. A Figura 2 ilustra a arquitetura geral da implementação da plataforma *NIZKCTF*, que utiliza somente serviços disponibilizados gratuitamente (até limites da ordem de 1 milhão de requisições) pela nuvem pública da Amazon.

Além da preocupação em adotar uma plataforma central segura, a ferramenta proposta neste artigo foi projetada para provisionar uma infraestrutura que resista aos modelos de adversário descritos ao final da seção a seguir.

 © 2017 Sociedade Brasileira de Computação

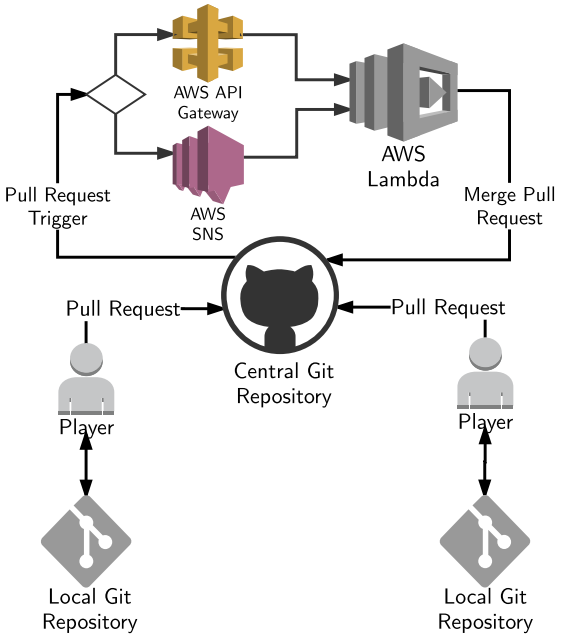


Figura 2. Visão geral da plataforma *NIZKCTF*, à qual o provisionador integra-se. Os competidores modificam repositórios Git locais e criam *pull requests* para o repositório central. Uma função *AWS Lambda* é disparada para integrar as alterações ao repositório central caso as mudanças sejam válidas.

4. Infraestrutura Provisionada e Modelos de Adversário

A Figura 3 apresenta a infraestrutura provisionada pela solução proposta neste artigo. Como forma de provisionamento inicial, utilizamos a ferramenta Ansible para preparar um certo número de máquinas virtuais para receber contêineres de desafios, além de uma máquina virtual específica para abrigar contêineres de VPN (*Virtual Private Network*).

Na rede gerenciada pelo componente Neutron do OpenStack, a ferramenta proposta por esse artigo aloca uma *vlan* para cada equipe que atingir a pontuação mínima necessária para ter acesso aos contêineres. Todas as máquinas virtuais possuem acesso a todas as *vlans*, porém internamente a essas máquinas conectamos cada *vlan* apenas ao contêiner alocado para a respectiva equipe.

Os contêineres de VPN são configurados com certificados TLS (*Transport Layer Security*) emitidos por uma autoridade comum, gerenciada pela própria ferramenta. Os times acessam essas instâncias por meio de um roteamento NAT (*Network Address Translation*) reverso. Cada equipe recebe o número da porta que é internamente roteada para seu contêiner de VPN, juntamente com um usuário e uma senha aleatória para conexão.

Contêineres contendo um mesmo desafio são provisionados sempre na mesma máquina virtual. Como utilizamos ZFS para armazenar os dados dos contêineres, essa organização permite aproveitar ao máximo os recursos de cópia na escrita (*copy-on-write*) desse sistema de arquivos. Ao provisionar um desafio para uma nova equipe, os blocos de armazenamento não são clonados, apenas referenciados. No decorrer da competição, apenas blocos que desviem do conteúdo original causarão um aumento do espaço em disco ocupado na máquina virtual.

© 2017 Sociedade Brasileira de Computação

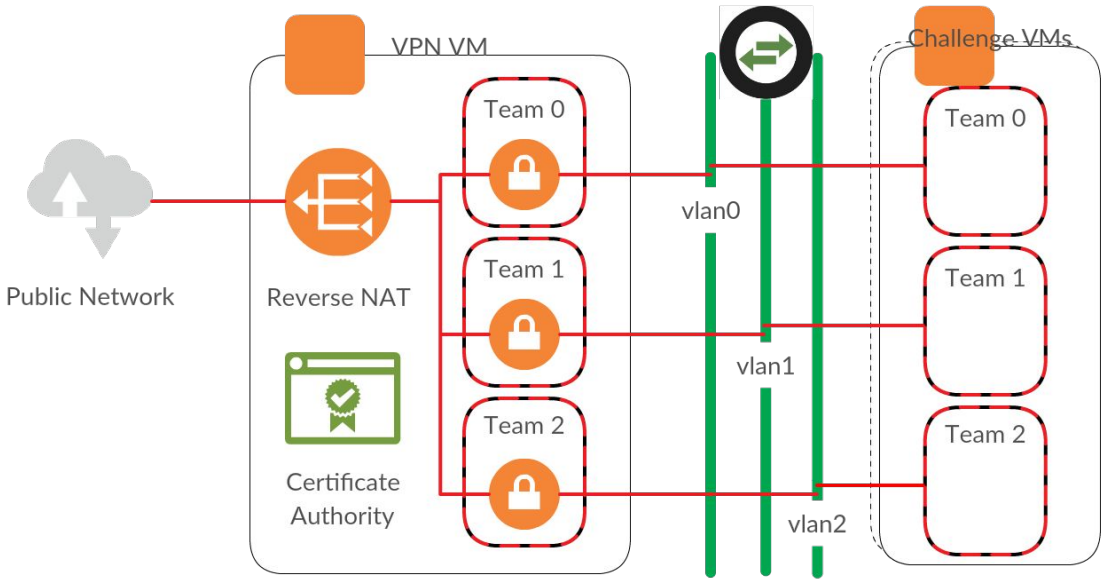


Figura 3. Infraestrutura provisionada pela ferramenta proposta. Cada time é alocado em um contêiner distinto, dentro de máquinas virtuais diferentes para cada problema. As equipes acessam a rede interna da competição via VPN.

Além disso, a organização proposta provê propriedades interessantes de segurança quando adotados os seguintes modelos de adversário:

- Um adversário que possua acesso a uma falha de dia zero (*zero day*) que permita escapar de contêineres, e pretenda obter as respostas (bandeiras) de problemas que não tenha resolvido: Neste caso, o adversário não conseguirá escapar de um problema para outro, assumindo que este não tenha acesso a uma falha de dia zero do virtualizador.
- Um adversário que possua acesso a uma falha de dia zero (*zero day*) que permita escapar de contêineres, e pretenda apagar ou corromper dados dos contêineres de outros times: Neste caso, o adversário precisa ao menos começar a resolver cada um dos problemas antes de perpetuar o ataque. Caso tenha obtido acesso para executar código remoto em apenas uma das máquinas virtuais, somente conseguirá explorar a falha de dia zero nessa máquina.

5. Arquitetura da Ferramenta

O processo principal da ferramenta é implementado por um único *script* escrito em linguagem Python. Esse *script* monitora constantemente o arquivo de submissões aceitas gerado pela plataforma *NIZKCTF* e, por meio da função `compute_target_state`, calcula o estado final desejado, ou seja, quais contêineres devem estar alocados em cada máquina virtual, dada a lista de desafios resolvidos por cada time.

Uma vez calculado o estado final, o *script* executa as seguintes funções, em ordem:

- `start_vms`: assegura que as máquinas virtuais que abriguem contêineres que precisem estar disponíveis estejam ligadas;
- `handle_container_transition`: assegura que os contêineres que precisem estar disponíveis existam dentro das máquinas virtuais, e destrói contêineres que não precisem estar disponíveis;

 © 2017 Sociedade Brasileira de Computação

- `stop_idle_vms`: desliga máquina virtuais que não estejam abrigando nenhum contêiner.

Cada transição (criação ou destruição) de contêiner é efetuada por um *script* em linguagem *shell* previamente instalado pela ferramenta Ansible em cada máquina virtual. A ferramenta chama esses *scripts* por meio de comandos SSH enviados com o auxílio da biblioteca Paramiko.

O *script* que provisiona o contêiner de VPN é especial pois configura as credenciais de acesso da equipe à rede interna da competição e, portanto, precisa comunicá-las de volta à equipe. Para isso, o provisionador adiciona a um repositório Git uma mensagem encriptada com a chave pública da equipe, permitindo que apenas a equipe a que se destina tenha acesso ao conteúdo, sem exigir que o provisionador tenha acesso a contatos diretos ou informações da equipe que não sejam do conhecimento de todos que acompanham a competição.

Todas as operações do provisionador são executadas em *thread pools*, com o objetivo de alcançar melhor desempenho (reduzindo as esperas de entrada/saída) e de permitir impor tempos máximos de execução (*timeouts*).

6. Homologação

A ferramenta proposta neste artigo foi previamente homologada, em pequena escala, em uma competição promovida durante a Semana de Computação (SeComp) do Departamento de Computação da UFSCar.

Durante essa competição, 4 equipes alcançaram pontuação suficiente para receberem acesso aos contêineres. Foram observados apenas problemas pontuais na solução de provisionamento aqui proposta, tais como a inadequação da configuração de *keepalive* do OpenVPN para a rede de alguns dos participantes que estavam atrás de roteadores NAT. Todos os problemas observados puderam ser corrigidos ainda no decorrer da competição.

7. Disponibilidade

O provisionador e projetos correlacionados estão disponíveis como software livre sob a licença MIT. Todos os códigos e documentação estão disponíveis nos repositórios da competição Pwn2Win no GitHub: `https://github.com/pwn2winctf`.

- Documentação de instalação da plataforma *NIZKCTF* (pré-requisito para realizar um CTF com o provisionador): `https://github.com/pwn2winctf/nizkctf-tutorial/blob/master/GitHub.md`
- Documentação de instalação do provisionador: `https://github.com/pwn2winctf/NIZKCTF-provisioning/blob/master/README.md`

8. Requisitos de Hardware

O provisionador foi testado em uma máquina virtual com as seguintes características:

- 1 núcleo de CPU de 2 GHz
- 2 GB de memória RAM
- 5 GB de disco

© 2017 Sociedade Brasileira de Computação

9. Demonstração Planejada para o SBSEG

Para o Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg), pretendemos instalar o provisionador proposto neste artigo em um computador pessoal (PC) ou *laptop* com acesso à Internet a ser exposto no SBSEG.

Configuraremos o provisionador para monitorar o repositório GitHub controlado por uma instância da plataforma *NIZKCTF* previamente configurada na Amazon. Posteriormente, disponibilizaremos alguns desafios para que os participantes do Simpósio possam simular sua participação em uma competição. Configuraremos o provisionador para instanciar esses desafios em máquinas virtuais na nuvem da UFSCar. Em outras palavras, o computador fisicamente exposto na SBSEG apenas controlará máquinas virtuais hospedadas em uma infraestrutura externa.

Alguns dos desafios terão a resposta (bandeira) incluída em seu próprio enunciado, a fim de permitir que mesmo aqueles que não tenham à disposição tempo para estudar e resolver um problema de CTF possam avaliar o funcionamento do provisionador. Desta forma, ao resolver ao menos um desafio que não tenha um servidor atrelado a si, o expectador da demonstração receberá credenciais para acessar a VPN hospedada na nuvem da UFSCar. Em seguida, ao resolver problemas atrelados a servidores, observará que os contêineres correspondentes são destruídos, e passa a não ser mais possível acessá-los.

Referências

ABI Research (2015). Global cybersecurity index & cyberwellness profiles. Technical report, International Telecommunications Union, Geneva, CH.

Bernstein, D. (2014). Containers and cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Bertochi, A. (2016). Pwn2Win CTF 2016 – Bastidores. https://ctf-br.org/2016/04/pwn2win-ctf-2016-bastidores.

Eastes, B. (2017). Capturing all the flags in BSidesSF CTF by pwning our infrastructure. https://hackernoon.com/capturing-all-the-flags-in-bsidessf-ctf-by-pwning-our-infrastructure-3570b99b4dd0.

Ernst, D., Bermbach, D., and Tai, S. (2016). Understanding the container ecosystem: A taxonomy of building blocks for container lifecycle and cluster management. In *IEEE Second International Workshop on Container Technologies and Container Clouds*, Berlin, Germany. IEEE.

Evans, K. and Reeder, F. (2010). A human capital crisis in cybersecurity: Technical proficiency matters. Technical report, Center for Strategic and International Studies, Washington, DC, USA.

Ferreira, U. J. S. (2017). Análise de tecnologias de virtualização e hardware de baixo custo para infraestutura de nuvem de pequeno porte. https://memoria.ifrn.edu.br/handle/1044/940.

Matias, P., Barbosa, P., Cardoso, T., Mariano, D., and Aranha, D. (2017). NIZKCTF: A non-interactive zero-knowledge capture the flag platform. https://arxiv.org/abs/1708.05844.

Yu, J. (2016). Seadog GitHub repository: RC3-CTF-2016-scoreboard. https://github.com/seadog007/RC3-CTF-2016-scoreboard.

 © 2017 Sociedade Brasileira de Computação

SBSeg 2017

XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

6 A 9 DE NOVEMBRO - BRASÍLIA - DF

sbseg2017.redes.unb.br

Realização

Apoio

Patrocínio

Agência Brasileira do ISBN
ISBN 978-857669422-9