black hat BRIEFINGS

AUGUST 6-7, 2025

MANDALAY BAY / LAS VEGAS

Watching the Watchers

Exploring and Testing Defenses of Anti-Cheat Systems

Sam Collins, Marius Muench, Tom Chothia

This talk

This talk is about **anti-cheats as software defenses.**

In this context:
- Cheats & Cheaters act as attackers
- Anti-Cheats & games act as defenders

Do expect …

- Cool software defenses
- Windows kernel internals
- To learn why a computer is almost never as secure as when playing Fortnite

Do not expect …

- Comparisons of anti-cheats to spyware
- Bypasses of anti-cheat systems
- Development tips for cheats

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Who Are We?

Sam

- PhD Student @ UoB,
- Man At The End Attacks & Reverse Engineering
- Game Dev but all my games are impossible to beat without cheating

Marius

- Assistant Prof @ UoB
- Baseband hacking, Reverse Engineering, & Low-Level Security
- Hacked the RP2350

Tom

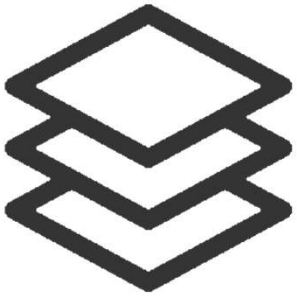
- Professor @ UoB
- Taught game hacking to his students for the last 5 years
- Hacked Apple Pay, Visa, Square, Bank of America, pacemakers, e-passports.

Setting the Scene

Image by Gary Jamroz

The scene - A harsh planet, on which continual combat leads to the evolution of super soldiers/monsters.

Why Anti-Cheats?



Full-Stack Defence
Software, Hardware, Firmware, Networking

Protection vs Privilege:
Kernel, Hypervisor, and Beyond

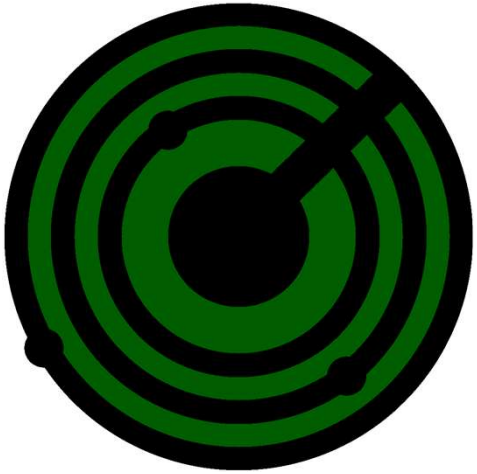
Mysterious Arcane Tricks
Invisible memory & underhanded windows hooking

Hands on Testing:
Playing Video Games at Work :P

Selected Titles

VALORANT	RAINBOW SIX SIEGE	FORTNITE	APEX LEGENDS	COUNTER STRIKE 2
18.6 Million (Monthly Players)	**~6-8 Million** (Monthly Players)	**110 Million** (Monthly Players)	**18 Million** (Monthly Players)	**~24 Million** (Monthly Players)
~$3.1 Billion (Lifetime Revenue)	**~$3.8 Billion** (Lifetime Revenue)	**~$26 Billion** (Lifetime Revenue)	**~$3.4 Billion** (Lifetime Revenue)	**~$6.7 Billion** (Lifetime Revenue)
Free	**Free**	**Free**	**Free**	**Free**

What Cheats Do



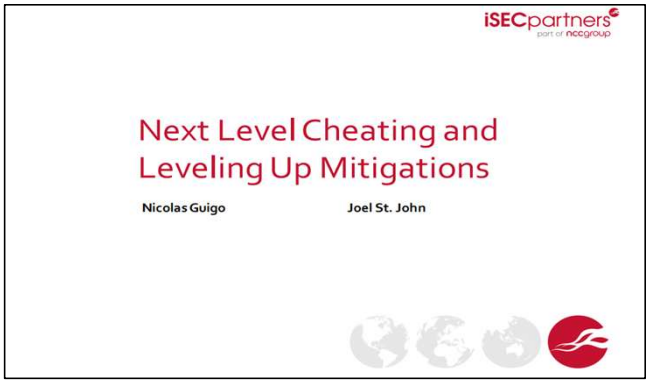
ESP
Extra-Sensory-Perception

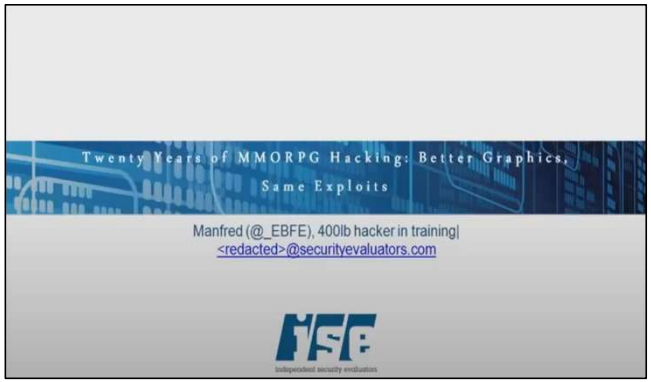
- Lets you see things you shouldn't
- Requires access to the game memory
- Shown in an app or overlay

Aimbot

- Does the shooting for you
- Requires access to the game memory
- Executed by artificial mouse clicks

Prior Art





And of course a lot of cheat forums :)

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

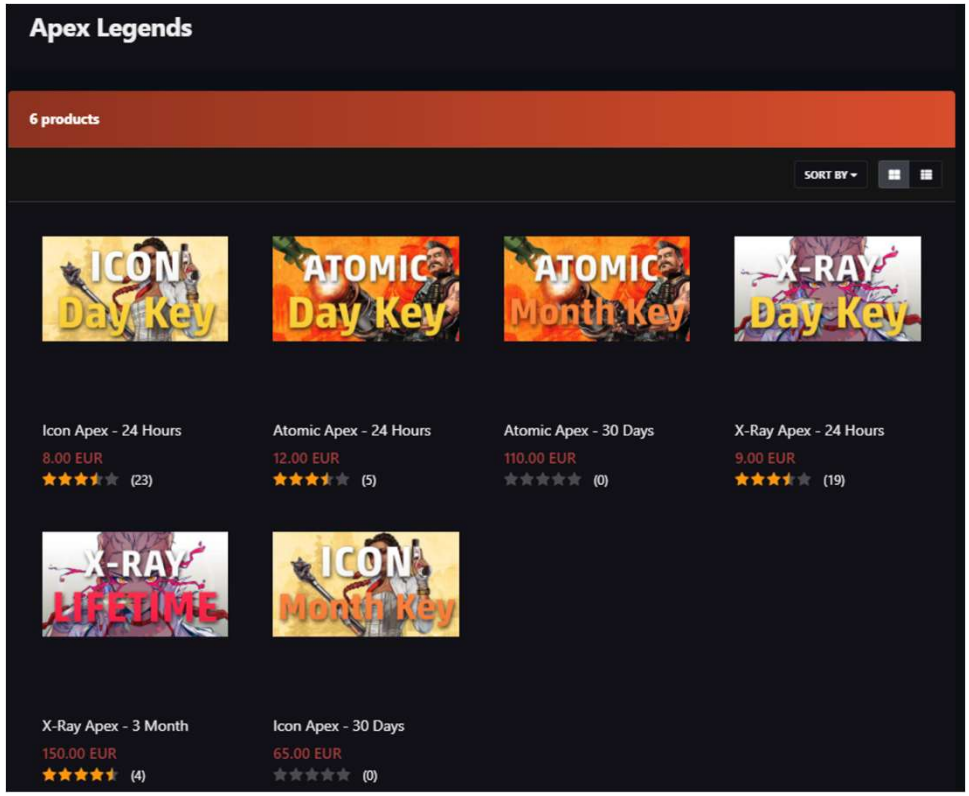
Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

A Huge Market for Game Cheats



We monitored **80 cheat selling sites** over **six months**, and make a market dataset avilable.

In most countries, game cheats are not illegal, but sites have been sued for copyright infringement.

Cheats sold on a **subscription model**, e.g., one month access.

Well run sites, with user reviews and credit card payment.

A Huge Market for Game Cheats

At any time, roughly **174,000 people** using cheats from these sites

Prices from **$12 to $220** dollars a month.

Based on standard e-market conversion rates top sites making **~$5,000,000 a year**.

You can make more money with a game cheat than from a bug bounty or from malware!

Site	Avg. mo. Traffic	Avg. mo. Cheat Price	Min. Price	Max. Price
Engine Owning	509,720	$13.80	$10.89	$19.59
Sky Cheats	197,463	$92.43	$35.00	$130.00
Battle Log	194,463	$72.84	$19.90	$145.75
Kernaim	189,338	$41.13	$16.50	$60.00
Lavi Cheats	153,429	$71.08	$29.00	$109.00
Interwebz Cheats	144,838	$21.79	$21.79	$21.79
Aimware	135,784	$19.16	$17.24	$22.99
Ring-1	115,353	$54.00	$29.00	$99.00
Phantom Overlay	87,528	$32.546	$19.96	$43.24

Market Observations

Days

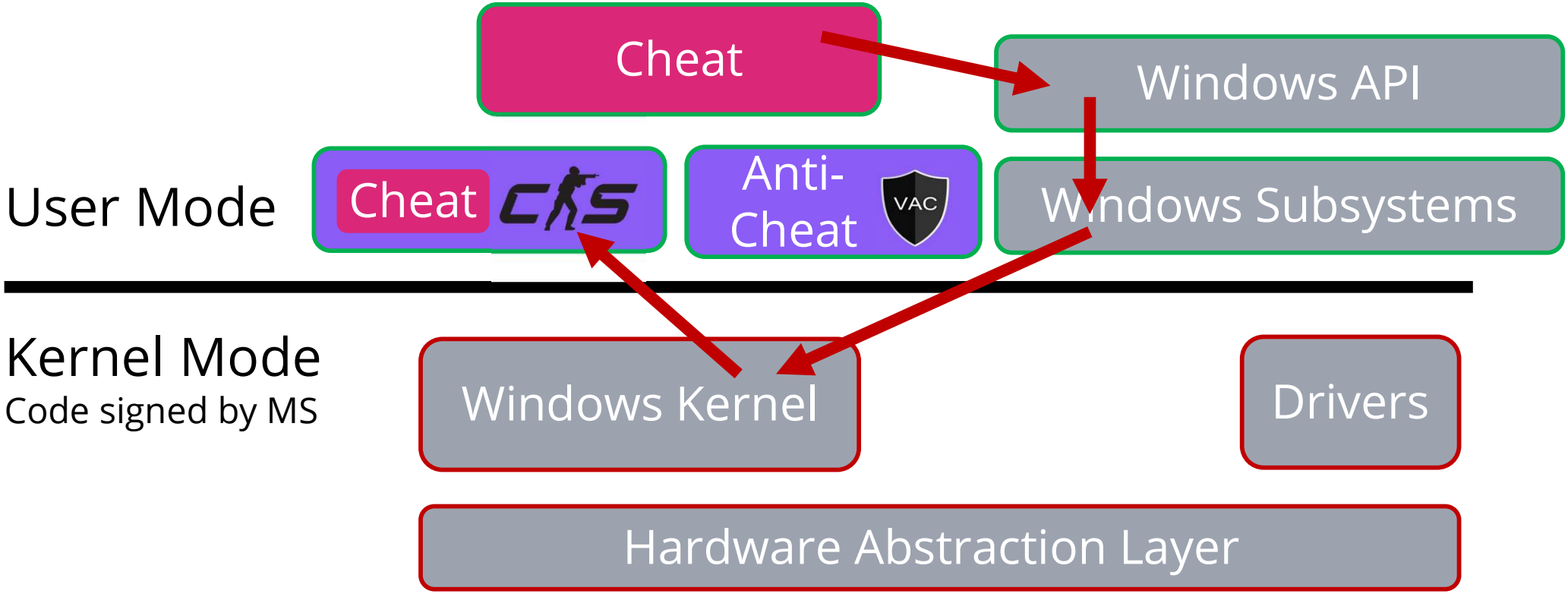
Mean Uptime = 50%

Battle Log - vader	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	—	—	—	—	●	●	●	●	●	●	●	●	●	●	●
Battle Log - fury	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Battle Log - quantum	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	●	●	●	●	●	●	●	·	·	·	·	·	·	·	·	—	—
Lavi Cheats - Yamatos	—	●	—	—	—	—	—	—	—	—	●	●	●	●	—	—	—	●	—	—	—	●	●	●	●	●	●	●	●	●	●	—	●	●
Lavi Cheats - Coffee	●	—	●	●	●	●	●	●	●	●	—	—	—	—	—	—	—	—	●	●	●	—	·	·	·	·	·	·	·	·	·	·	—	—
Lavi Cheats - Grave	—	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	—	—	●	●	●	●	●	●	●	—	—	—	—	—	—	—	—	—
Lavi Cheats - Hyperion	—	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	·	·	·	·	·	·	·	·	·	·	●	●
Sky Cheats - Division	●	●	●	●	●	●	●	●	—	—	—	●	●	●	●	●	●	—	—	—	—	—	—	—	—	—	—	—	—	●	●	—	—	—	—
Sky Cheats - Omega	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	●	●	●	●	●	—	—	●	●
Sky Cheats - Zero	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	—	—	●	●	●	—	—	●	●	●	●	●	—	—	—	—	—	●	●
Sky Cheats - Valkyrie	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	●	●	●	—	—	—	—
Sky Cheats - Tenet	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Private Cheatz - Hyperion	●	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	·
Private Cheatz - Droid	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	·
Private Cheatz - Intel	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	·
Lavi Cheats - Sky	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	●	—	—	—	—	●	●	●	●	●	—	—	●	●	●	—	●
Lavi Cheats - Pro	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	●	●	●	●	●	●	●	●	●	●	●	●

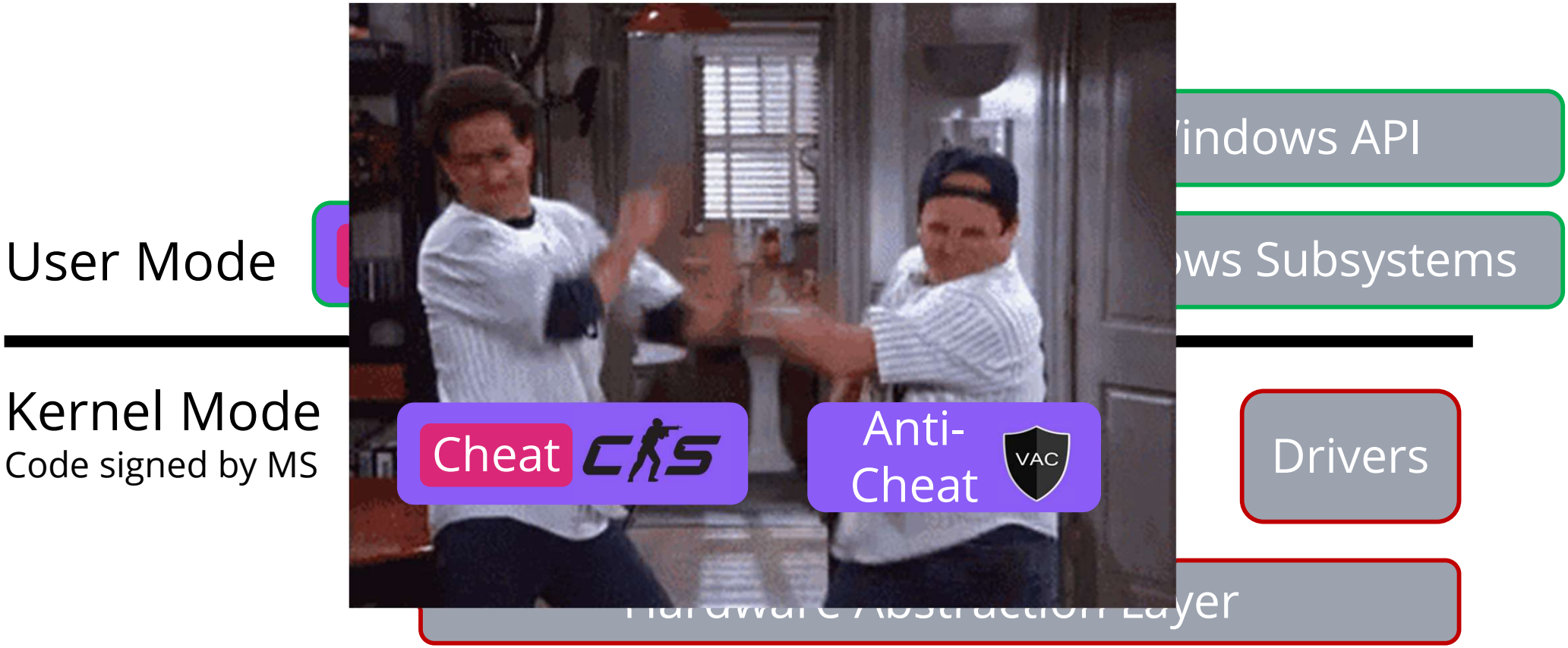
VALORANT

● Cheat Working | — Cheat Not Working | · Cheat not Available

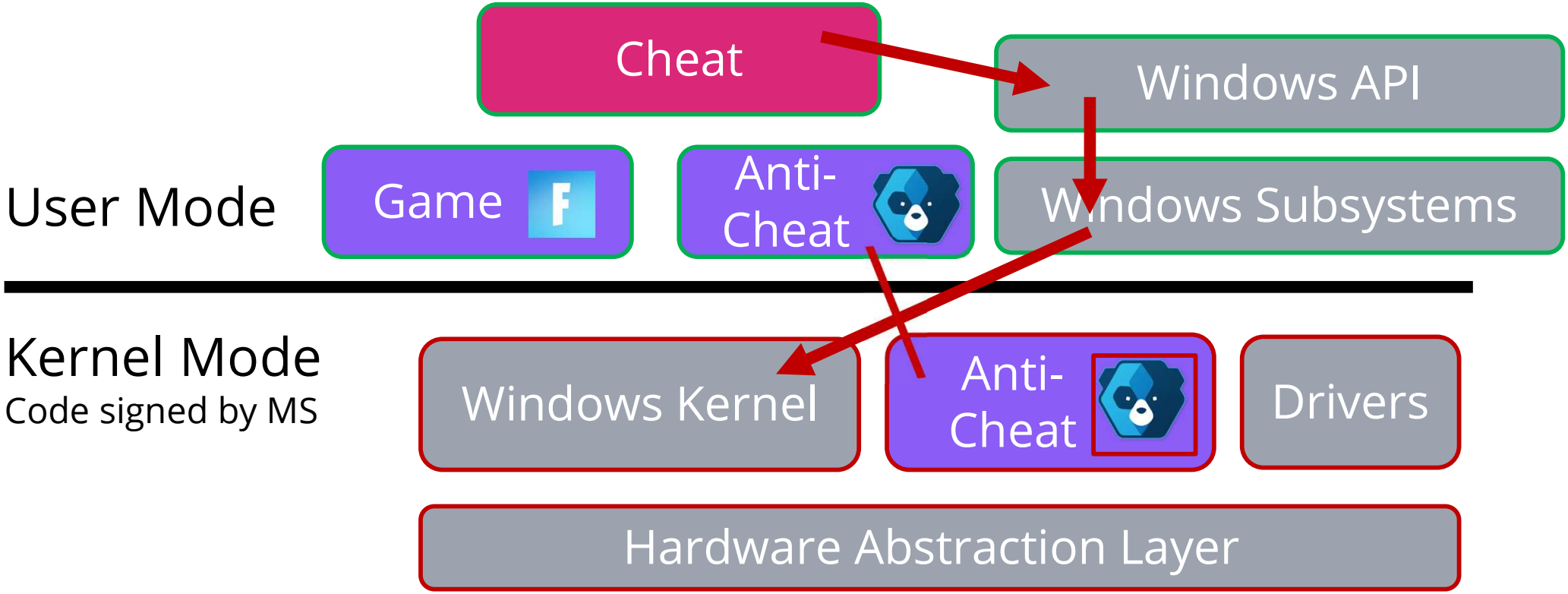
User Level Anti-Cheat



User Level Anti-Cheat



Kernel Level Anti-Cheat



Kernel Level Anti-Cheat

User M

Kernel

Code signe

API

systems

vers

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Cheat Forums – The Good, Bad, and Ugly

Cheat forums are the best and the worst source of information about game hacking and anti-cheats – this talk would not have been possible without this impressive community

Code:

Game Offsets

```
namespace offset
{
    // Main Offsets

    constexpr auto dwEntityList = 0x6190778;          // Entity list
    constexpr auto dwLocalPlayer = 0x26a2[illegible]8; // [illegible]tal player entity handle
    constexpr auto ViewMatrix = 0x11a350;             // [illegible]w matrix
    constexpr auto ViewRender = 0x3C81F10;            // View render (correct)
    constexpr auto OFFSET_NAMELIST = 0x8bc2c80;       // Name list

    // Glow
    constexpr auto OFFSET_HIGHLIGHTSERVERACTIVESTATES = 0x29C; // Highlight active states
    constexpr auto OFFSET_HIGHLIGHTCURRENTCONID = 0x3c8;       // Highlight enable
    constexpr auto OFFSET_HIGHLIGHTVISIBILITYTYPE = 0x3d0;     // Highlight visibility through walls
    constexpr auto OFFSET_HIGHLIGHTSETTINGS = 0x68da910;       // Highlight settings
    constexpr auto OFFSET_GLOW_HIGHLIGHT_ID = 0x29C;           // Glow highlight ID
    constexpr auto OFFSET_HIGHLIGHT_TYPE_SIZE = 0x34;          // Highlight type size
    constexpr auto OFFSET_GLOW_FIX = 0x278;                    // Glow fix
    constexpr auto OFFSET_GLOW_DISTANCE = 0x264;               // Glow distance
    constexpr auto OFFSET_GLOW_THROUGH_WALL = 0x26c;           // Glow through wall

    // Player
    constexpr auto m_vecAbsVelocity = 0x170;          // Absolute velocity (Vector3)
    constexpr auto m_localOrigin = 0x17c;             // Local origin
    constexpr auto m_shieldHealth = 0x01a0;           // Shield health
    constexpr auto m_shieldHealthMax = 0x01a4;        // Max shield health
    constexpr auto m_iHealth = 0x328;                 // Health
    constexpr auto m_iTeamNum = 0x334;    //0x338     // Team number
    constexpr auto m_iMaxHealth = 0x470;              // Max health
    constexpr auto m_lifeState = 0x690;               // Life state
    constexpr auto m_lastvisibletime = 0x1a54;        // Last visible time
    constexpr auto OFFSET_CROSSHAIR_LAST = m_lastvisibletime + 0x08; // Crosshair last t[illegible]
    constexpr auto camera_origin = 0x1fac;            // Camera origin (Vector3)
    constexpr auto m_ammoPoolCapacity = 0x25FC;       // Ammo pool capacity
    constexpr auto VIEW_ANGLES = m_ammoPoolCapacity - 0x14;  // View angles + Ammo Pool
    constexpr auto m_SwayAngle = VIEW_ANGLES - 0x10;  // Sway angle

    // Misc
```

Offsets correct? because esp not work :/

Guides

ASUS Motherboard AMI Bios Spoofing

I have been using this method for around 3 years. I made a small guide that I gave to son[illegible] method to safely ban evade HWID bans in many games.

Tested working on: EAC/BE/Vanguard/ESEA/FACEIT

Since we are directly editing the BIOS ROM, there isn't anything any anti-cheat can do abo[illegible]

This guide is for ASUS motherboards, but it may work on other motherboard brands that u[illegible]

Step One: Dump your BIOS ROM file
The software you need will be depending on your motherboard and BIOS model/version.

I have an ASUS z390p and use "Aptio V AMI Firmware Update Utility"

Here is a video of how the BIOS ROM dumping should look:

19th May 2024, 03:01 PM

adammmmm1
n00bie

Threadstarter

so soldering the dma to ram slots?

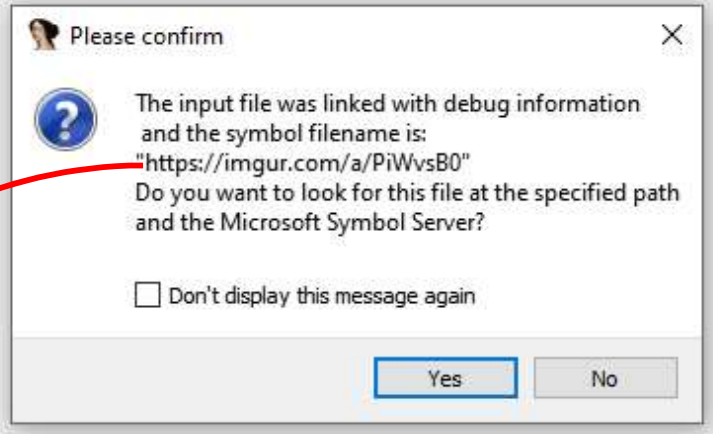
Whatever this is

Riot Vanguard - Development Team

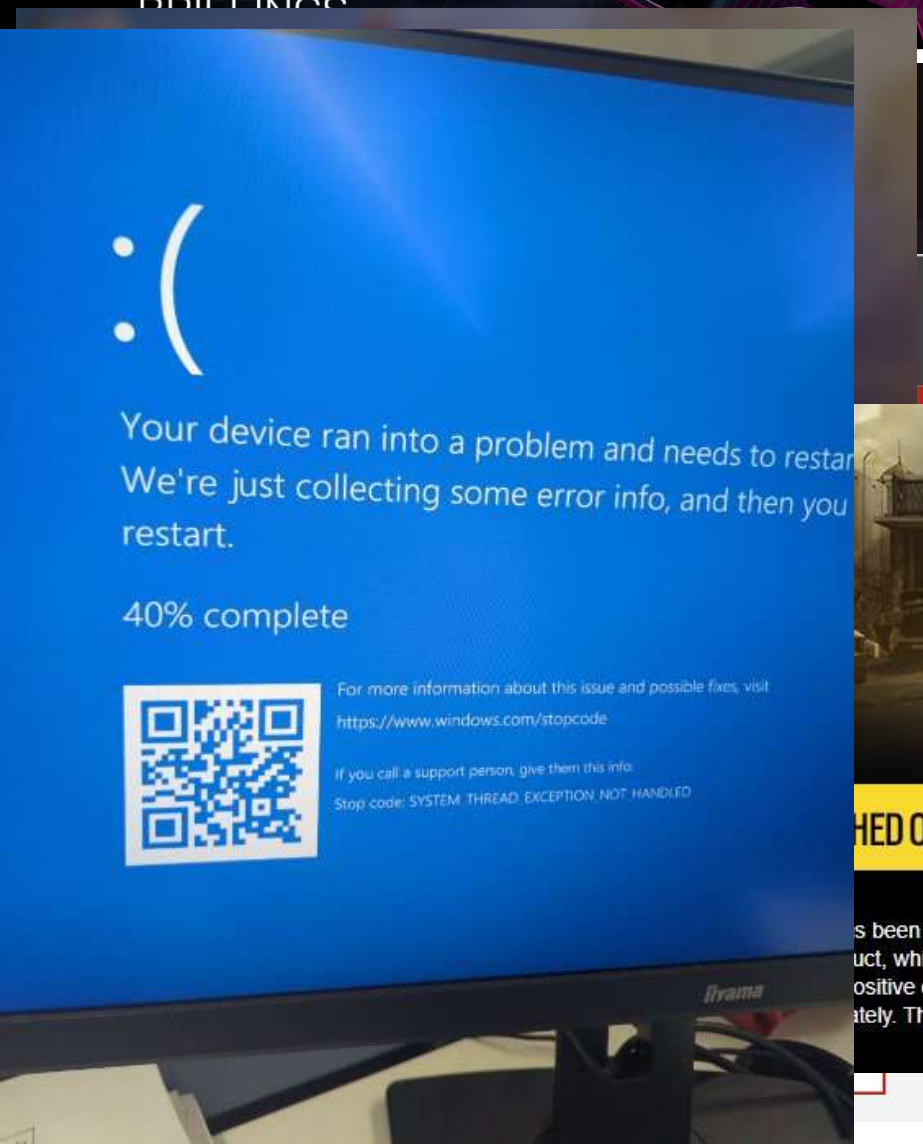
The central Riot Anti-Cheat team circa Feb. 18, 2020.

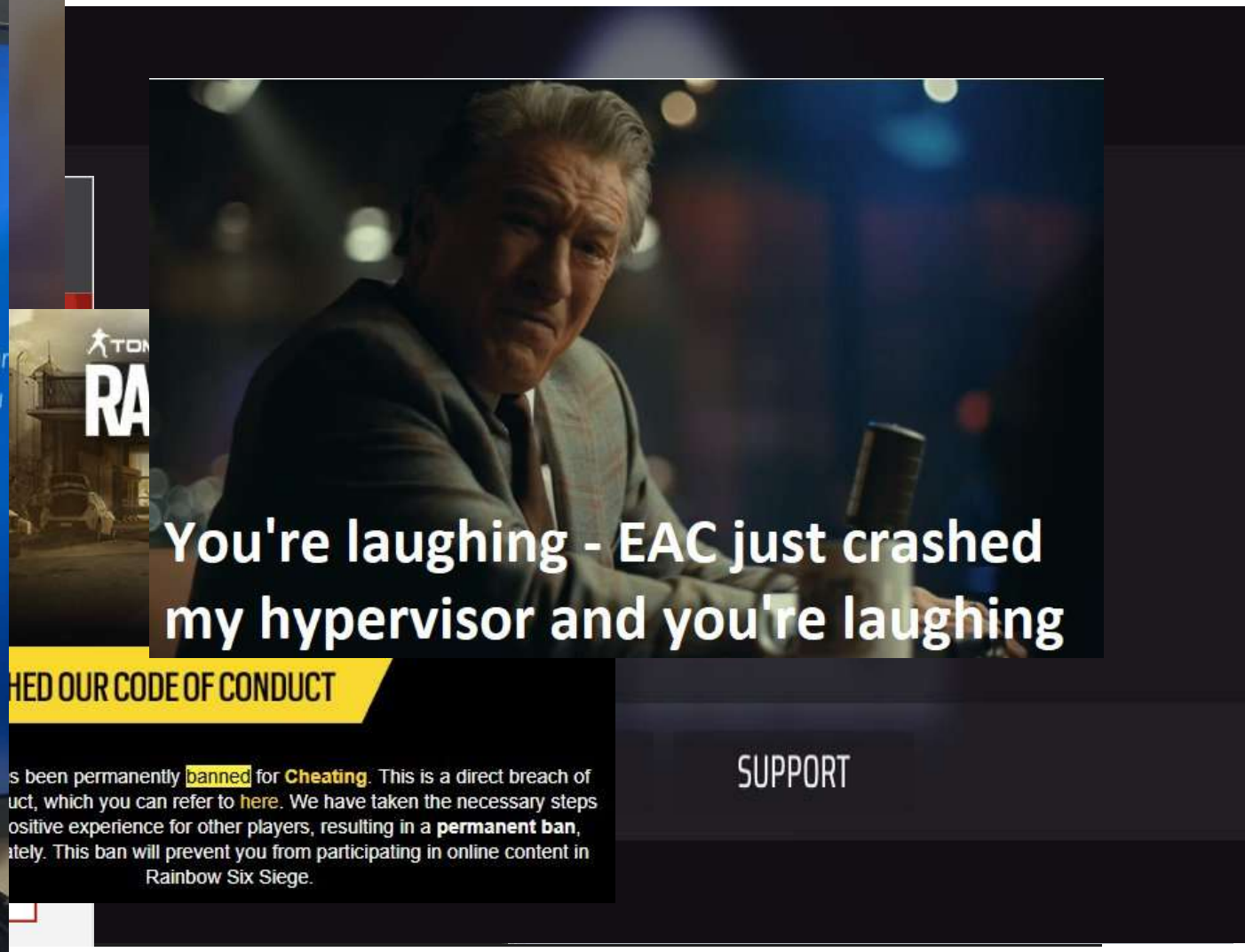
Want to join our gang? Take a look at our careers website (www.riotgames.com/careers) for openings.

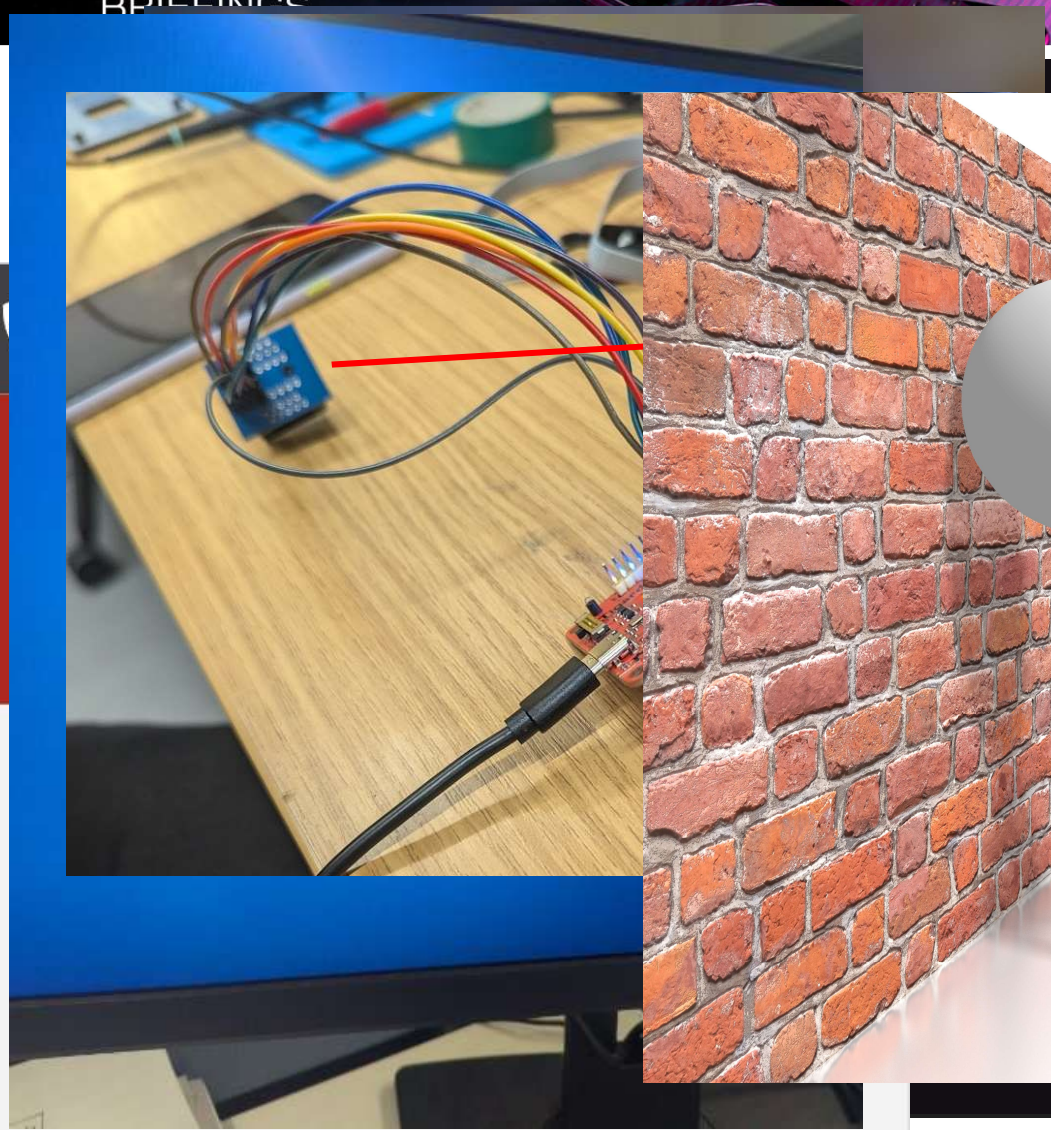
```
c2a8660 00000000`00000002 00000000`00000000 : nt!KeBugCheckEx
c104b19 00000000`00000f4d ffff9d42`00000000 : nt!KiBugCheckDispatch+0x69
0000000 ffffcae4`c7693d40 fffff800`00000000 : nt!KiPageFault+0x478
0000000 ffffc98e`d27a8000 ffffb588`2cef6080 : myfault+0x12d0
1e0fc70 00000000`000000f0 fffff800`0f407b91 : myfault+0x168e
d7ac3c0 ffffc98e`d17afb50 00000000`00000000 : myfault+0x17f1
```

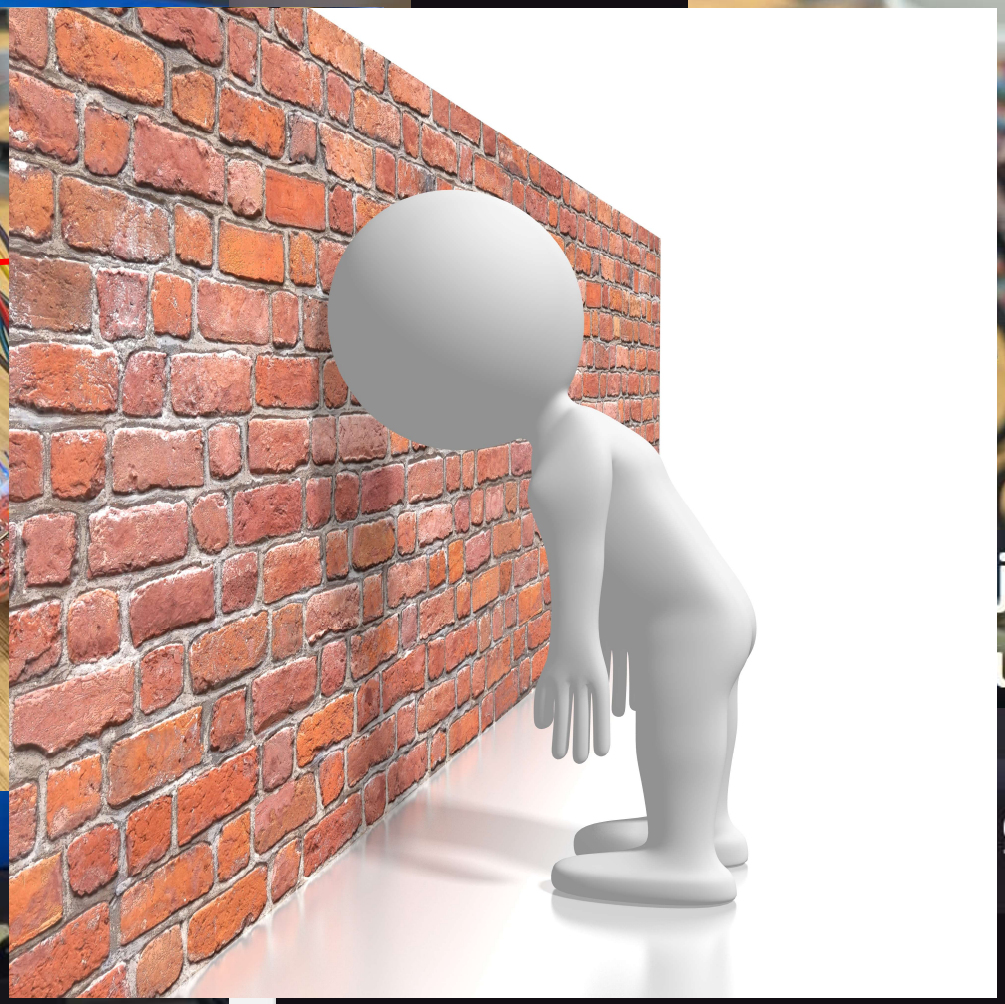


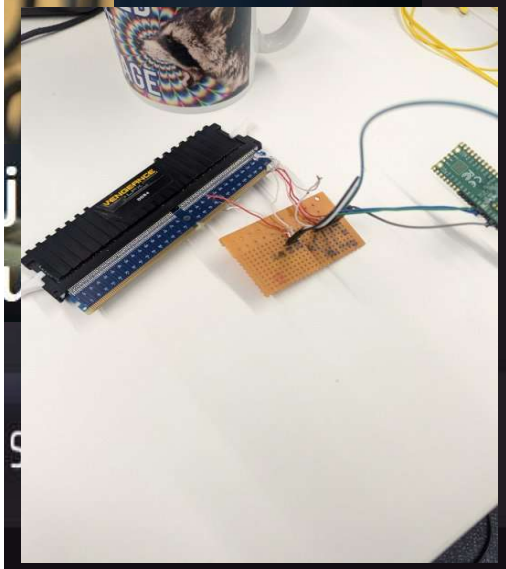

```
nt!DbgBreakPointWithStatus:
fffff807`07e06e40 cc
0: kd> g
Hello, it Vanguard, who dis?
Input>
```


:(
Your device ran into a problem and needs to restar
We're just collecting some error info, and then you
restart.
40% complete
For more information about this issue and possible fixes, visit
https://www.windows.com/stopcode
If you call a support person, give them this info:
Stop code: SYSTEM_THREAD_EXCEPTION_NOT_HANDLED


You're laughing - EAC just crashed
my hypervisor and you're laughing
HED OUR CODE OF CONDUCT
s been permanently banned for Cheating. This is a direct breach of
uct, which you can refer to here. We have taken the necessary steps
ositive experience for other players, resulting in a permanent ban,
ately. This ban will prevent you from participating in online content in
Rainbow Six Siege.
SUPPORT







Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Anti Cheat Defences – The Usual Suspects

Any defense you have heard about is probably used:

- Registered Callbacks
- Signature scanning
- File and memory integrity checks
- Obfuscations and packing
- Anti Debug
- Hooking API calls
- AI detection methods
- Instruction Misalignment
- TPM usage
- Stack walking

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Kernel code protection

All code in the kernel should be signed.

$\neq$

Windows checks that all code loaded
into the kernel via normal APIs is signed.

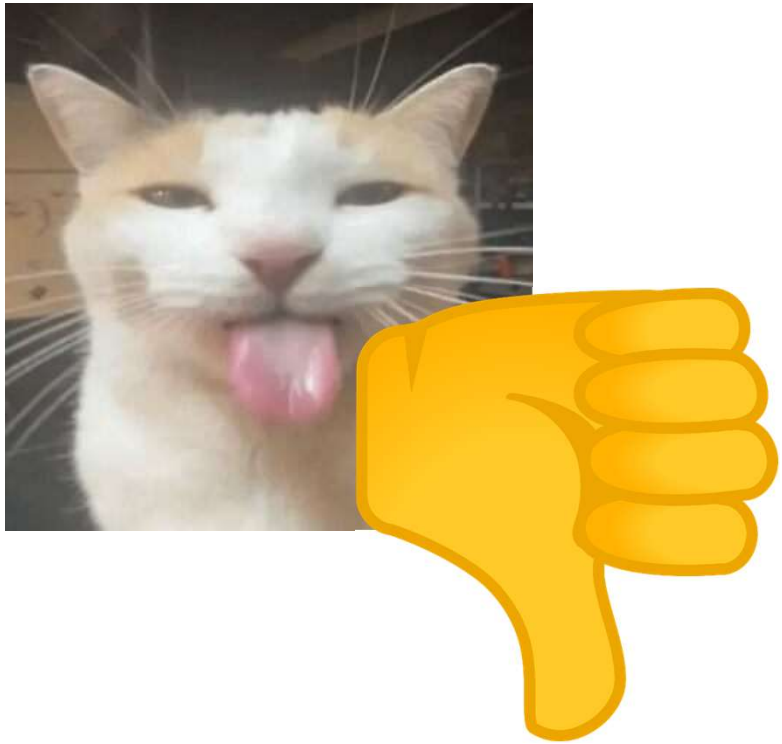
Bring Your Own Vulnerable Driver (BYOVD)

1. Legitimate drivers contain bugs/vulnerabilities

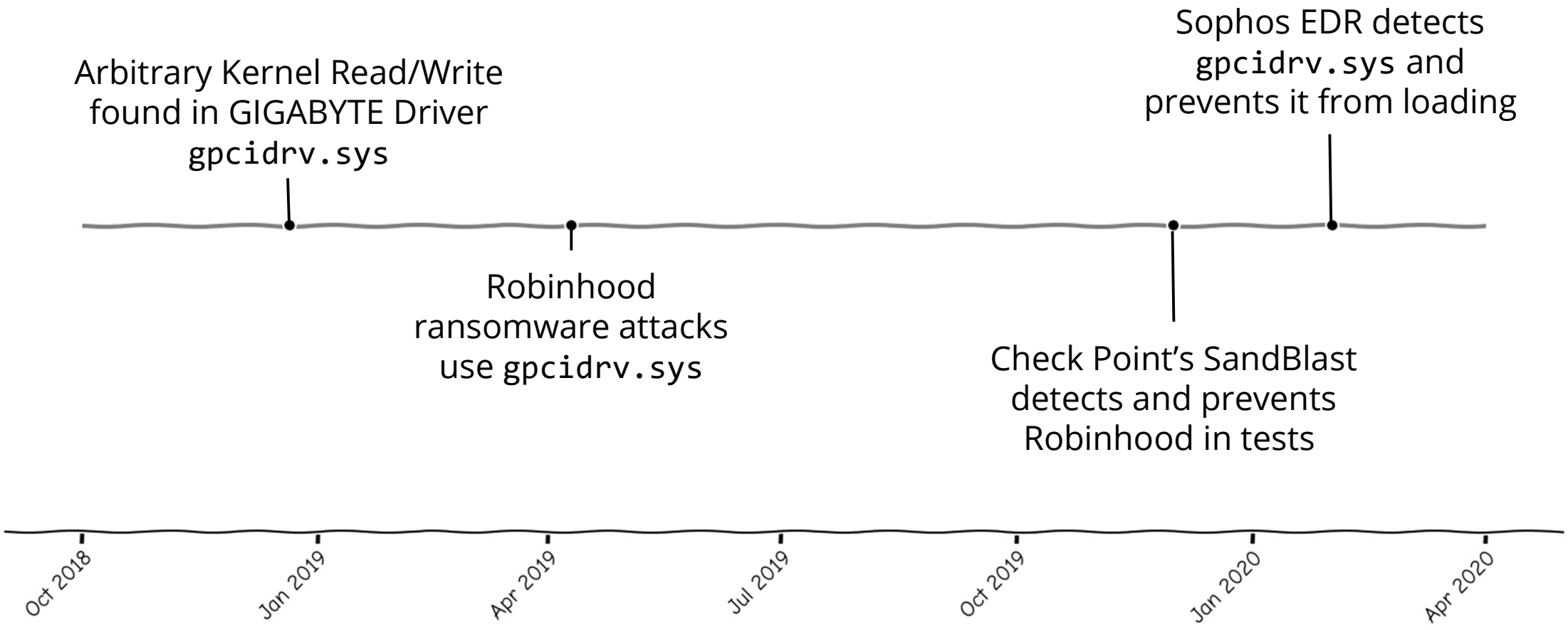
2. Attackers exploit these

3. Unsigned code can now be loaded into the kernel

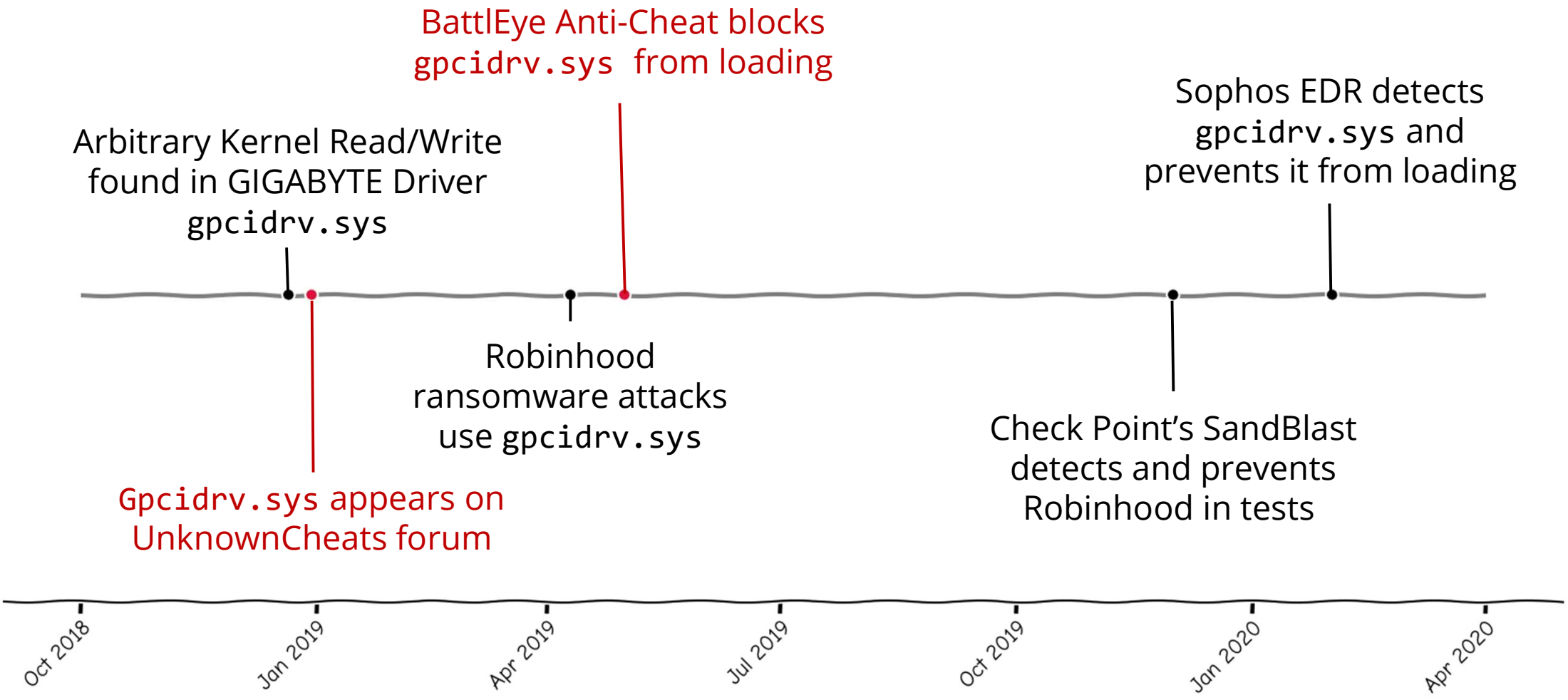
In recent years, this became a popular entry vector for malware



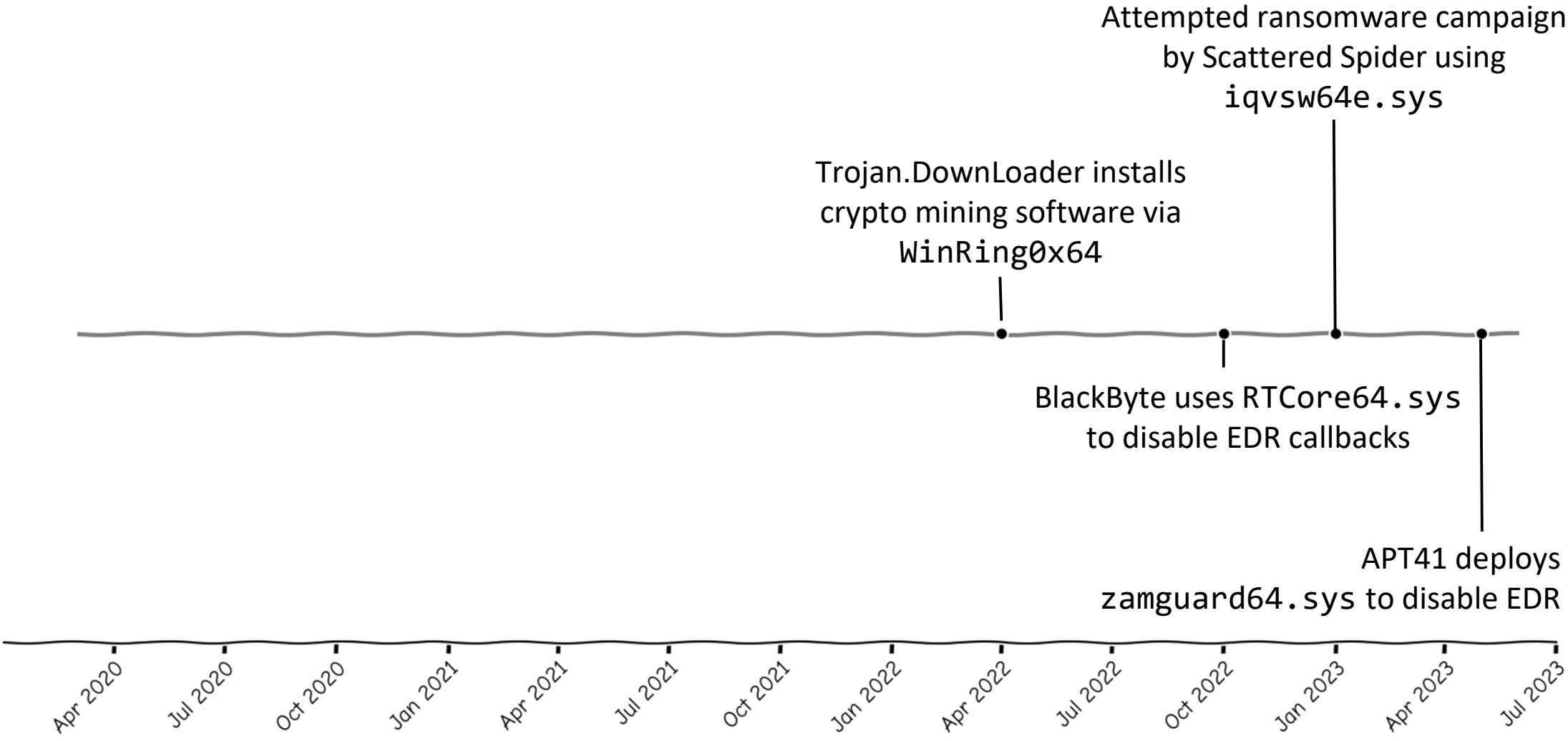
BYOVD – Malware Case Study



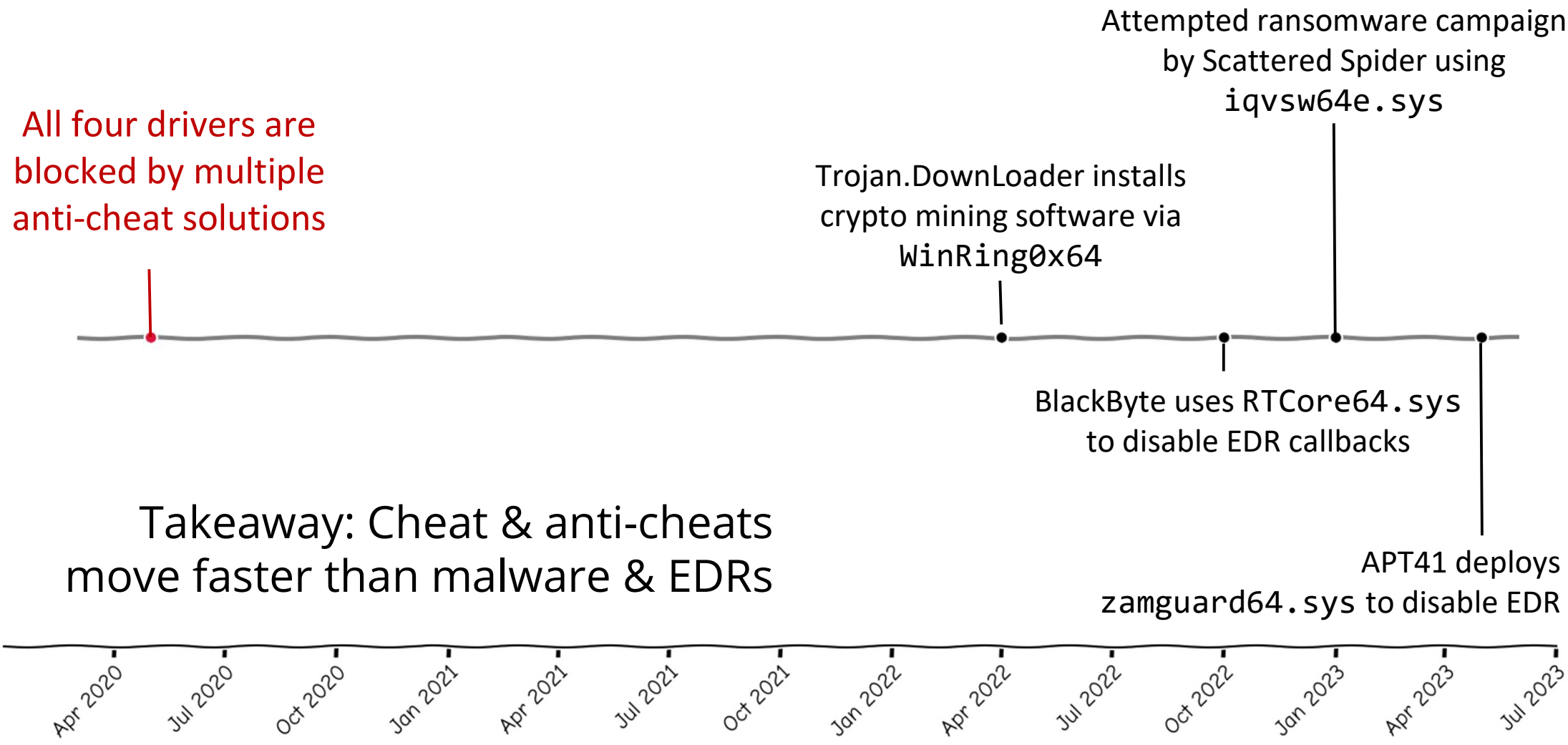
BYOVD – Malware Case Study



BYOVD – Malware Case Study



BYOVD – Malware Case Study



Takeaway: Cheat & anti-cheats move faster than malware & EDRs

How Anti Cheats stop BYOVD

Method A
Load Time Prevention

Block vulnerable drivers from loading altogether

Example - Using object callbacks to intercept handle manipulation behaviour and strip access rights

Method B
Run Time Detection

Walk through suspect areas and scan for malicious code

Example – Scanning through the nonpaged pool space → looking for known behaviour signatures

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Shortcomings of BYOVD Defenses

Method A
Load Time
Prevention

Issue – Cheat can be loaded before the game runs

Method B
Run Time
Scanning

Issue – Slow to run and hurts game performance

- Both methods rely on signatures to detect known drivers/cheats

- How to **detect unknown attacks**?

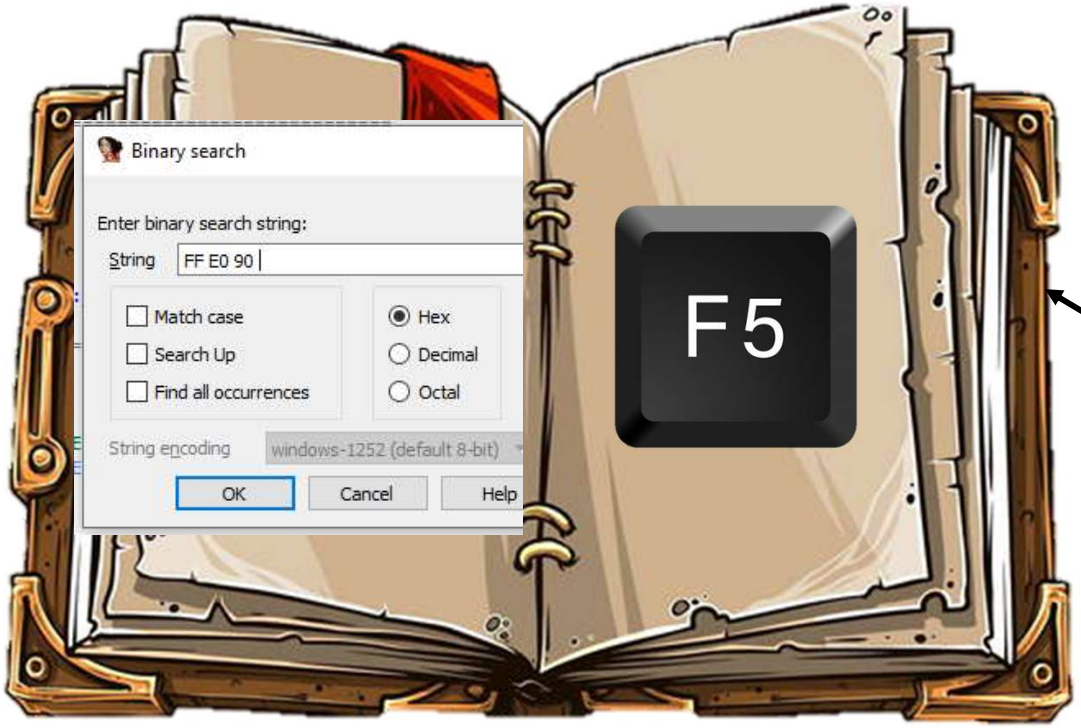
- Some anti-cheats use *arcane* measures

↓

Let's Investigate!

VALORANT

Let's Investigate



Crash dump

Fishing for Hooks

Address	Function	Instruction
.data:000000014006F2A3		db 0FFh ; ÿ
.data:000000014006F2CB		db 0FFh ; ÿ
seg010:00000001400E3FC9	sub_1400E3FAD	jmp rax
seg010:00000001400E560B	sub_1400E551D	jmp rax
seg010:00000001400E89A9	sub_1400E8894	jmp rax
seg010:00000001400E9D4B	sub_1400E9D32	jmp rax
seg010:00000001400EDF5B	sub_1400EDF0B	db 2, 2 dup(0), 0FFh, (
seg010:00000001400F1817	sub_1400F17B5	db 0FFh ; ÿ
seg010:00000001400FA3F6	sub_1400FA39A	jmp rax
seg010:0000000140102AB3	sub_140102979	jmp rax
seg010:0000000140111156	sub_1401110B3	jmp rax
seg010:000000014014A2A7	sub_14014A220	jmp rax
seg010:000000014014BDD3	sub_14014BD3A	jmp rax
seg010:0000000140150B87	sub_140150B5A	jmp rax
seg010:0000000140159785	sub_14015973E	jmp rax
seg010:0000000140179BF9	sub_140179BDB	jmp rax
seg010:000000014017C19A	sub_14017C0DA	jmp rax

Fishing for Hooks

Target 1

Target 2

```
L40                 8024AC8D4800000158EC814855h
                                    ; DATA XREF: sub_140116F91+8
L                   mmword 909090E0FF0000000000000000B84850h
                                    ; DATA XREF: sub_1400E9D1
                                    ; seg010:0000000140E85094
L40                 0AC8D4800000150EC8148565508EC8348h
                                    ; DATA XREF: sub_140195A88+8↓r
                                    ; seg010:0000000141197764↓o ...
                24h ; $
        db      80h ; €
        db        0
        db        0
        db        0
        db        0
        db        0
        db        0
L4006F2C0 xmmword 909090E0FF0000000000000000B84850h
                                    ; DATA XREF: sub_1401319F
                                    ; seg010:0000000141285754
```

Detour 1

Detour 2

```
nt!KiPageFault:
fffff800`0f20dd00 50              push    rax
fffff800`0f20dd01 48b8b0d0a62c00f8ffff mov rax,offset vgk+0x5d0b0
fffff800`0f20dd0b ffe0            jmp     rax
fffff800`0f20dd0d 90              nop
fffff800`0f20dd0e 90              nop
fffff800`0f20dd0f 90              nop
fffff800`0f20dd10 c645ab01        mov     byte ptr [rbp-55h],1
fffff800`0f20dd14 488945b0        mov     qword ptr [rbp-50h],rax
```

```
nt!KiSwInterrupt:
fffff800`0f205050 50              push    rax
fffff800`0f205051 48b83dd1a62c00f8ffff mov rax,offset vgk+0x5d13d
fffff800`0f20505b ffe0            jmp     rax
fffff800`0f20505d 90              nop
fffff800`0f20505e 90              nop
fffff800`0f20505f 90              nop
fffff800`0f205060 90              nop
fffff800`0f205061 90              nop
```

A look at the Targets

KiPageFault

- Windows page fault handler
- Handles:
  - Bad read/write access
  - Page protection violations
  - Executing NX pages

KiSwInterrupt

- Kernel trap handler for software interrupts
- Triggered by the OS for deferred kernel work (DPCs)

Page Fault

```
test    byte ptr [rsp+18h],1
jne     vgk+0x5d111 (fffff800`2ca6d111)
```

If interrupt is from Kernel…

```
test    byte ptr [rsp+8],10h
je      vgk+0x5d111 (fffff800`2ca6d111)
```

And Page Fault code is 4 (executing NX page)

```
mov     rax,cr8
cmp     al,2
ja      vgk+0x5d111 (fffff800`2ca6d111)
```

And IRQL <= 2

```
push    rax
mov     eax,2
mov     cr8,rax
push    rcx
push    rdx
push    r8
push    r9
push    r10
push    r11
mov     rcx,cr2
mov     edx,dword ptr [rsp+30h]
sub     rsp,20h
sti
call    qword ptr [vgk+0x6f368 (fffff800`2ca7f368)]
```

Run CustomErrorHandler(RCX = FaultingAddress, RDX = ErrorCode);

…

Laying the Trap

- Malicious code is often mapped using **MmAllocatePagesForMdl** or **ExAllocatePoolWithTag**
- Both create a safe, non-pagable, area for the code to execute
- On game boot -> page map flags for these target areas is written
- **NX is set for target PPE**, the second level of paging

```
                                        VA ffffca043f131650
PXE at FFFFF67B3D9ECCA0    PPE at FFFFF67B3D994080    PDE at FFFFF   ---DA--KWEV   65021F8988
contains 0A000000063CD863  contains 0A00000006BD0863  contains 8A0                 000000000000
pfn 63cd      ---DA--KWEV  pfn 6bd0      ---DA--KWEV  pfn 1001800                  pfn 1001931
```

Game Boot

```
                                        VA ffffca043f131650
PXE at FFFFF67B3D9ECCA0    PPE at FFFFF67B3D994080    PDE at FFFFF   ---DA--KW-V   65021F8988
contains 0A000000063CD863  contains 8A00000006BD0863  contains 8A0                 000000000000
pfn 63cd      ---DA--KWEV  pfn 6bd0      ---DA--KW-V  pfn 1001800                  fn 1001931
```

Detection Pipeline

Install hook on Page Fault handler

↓

Spray NX on suspect kernel areas

↓

Execution attempt in nonpaged pool

↓

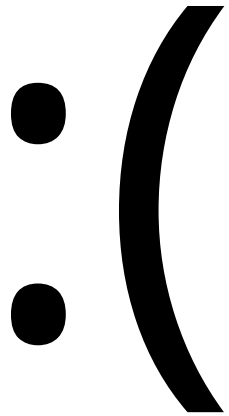
Exception thrown & caught by the custom handler

↓

Defender can analyse & respond

Blue Screen of Death

- Attempt to replicate a page fault hook…
- We install a simple inline hook which returns to the main fault handler…



CRITICAL_STRUCTURE_CORRUPTION

Windows Kernel Patch Protection Blue Screens our Machine :(

- PatchGuard protects critical kernel structures and functions
- It **hides by piggybacking** legitimate kernel entry points
- This way it can execute its checks without exposing a dedicated thread
- **KiSwInterrupt** is one such entry point

TETRANE

Updated Analysis of PatchGuard on Microsoft Windows 10 RS4

A use case of REVEN, the Timeless Analysis Tool

Author : Luc Reginato, @_YouB_

www.tetrane.com

III – Triggering a Check
E - KiSwInterruptDispatch

A look at the Targets

KiPageFault

- Windows page fault handler
- Handles:
  - Bad read/write access
  - Page protection violations
  - Executing NX pages

- Core function - protected by windows kernel patch protection

KiSwInterrupt

- Kernel trap handler for software interrupts
- Triggered by the OS for deferred kernel work (DPCs)
- Piggy backed by windows kernel patch protection

Muting PatchGuard

KiSwInterrupt Hook:

```
test    byte ptr [rsp+10h],1
je      vgk+0x5d147 (fffff800`2ca6d147)
swapgs
push    rcx
push    rdx
push    r8
push    r9
push    r10
push    r11
sub     rsp,20h
xor     ecx,ecx
call    qword ptr [vgk+0x6f370 (fffff800`2ca7f370)]
add     rsp,20h
pop     r11
pop     r10
pop     r9
pop     r8
pop     rdx
pop     rcx
test    byte ptr [rsp+10h],1
je      vgk+0x5d175 (fffff800`2ca6d175)
swapgs
pop     rax
iretq
```

Check interrupt came from the kernel

Save registers & align stack

Unclobber registers & stack

Check privilege and return to interrupted code

```
10: kd> dq vgk+0x6f370
fffff800`2ca7f370  fffff800`0f102260 00000
fffff800`2ca7f380  00000001`00000000 00000
fffff800`2ca7f390  00000000`00000000 00000
fffff800`2ca7f3a0  00000000`00000000 00000
```

Another Function Call

```
nt!HalPerformEndOfInterrupt:
fffff800`0f102260 4053            push    rbx
fffff800`0f102266 488bd9          mov     rbx,rcx
fffff800`0f102269 4885c9          test    rbx,rcx
fffff800`0f10226c 7520            jne     nt!HalPerformEndO
fffff800`0f102275 e826641000      call    nt!guard_dispatch
fffff800`0f10227a 803def74940000  cmp     byte ptr [nt!Halp
```

HalPerformEndOfInterrupt

Tells interrupt controller that the CPU is finished processing an interrupt

Other PatchGuard Smashing

- Vanguard disables PatchGuard entry via KiSwInterrupt with an inline hook
- It also mutes currently running PatchGuard contexts -> queuing infinite waits
- And corrupts DPC structures to break PatchGuard's deferred execution and checks

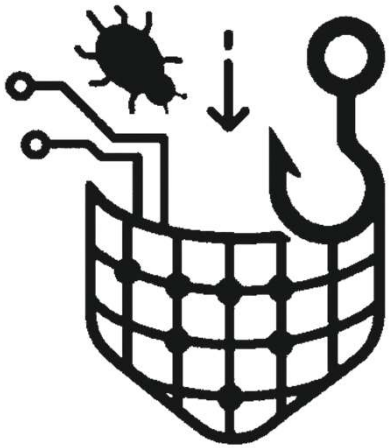
```
> system thread we suspect is running PatchGuard

THREAD ffffb58817d0d040  Cid 0004.02d8
Teb: 0000000000000000
Win32Thread: 0000000000000000
> infinite wait object isn't suspect at all :p
WAIT: (DelayExecution) KernelMode Non-Alertable  ffffffffffffffff  NotificationEvent

> Looking in the stack of KeDelayExecutionThread...

   4.0002d8  ffffb58817d0d040 000000d Blocked
                                       nt!KiSwapContext+0x76
                                       nt!KiSwapThread+0x500
                                       nt!KiCommitThreadWait+0x14f
                                       nt!KeDelayExecutionThread+0x122
                                       vgk+0x1277d3
                                       vgk+0x13fd3d
                                       vgk+0x17a84f
                                       vgk+0x15f4d0
                                       nt!PspSystemThreadStartup+0x55
                                       nt!KiStartSystemThread+0x28

> get your fingers out the PatchGuard pie riot!
```

```
8: kd> dx -id 0,0,ffffca0447c05080 -r1 (*((ntkrnlmp!_KDPC_DATA *)0x1030e4))
(*((ntkrnlmp!_KDPC_DATA *)0x1030e4))                 [Type: _KDPC_DATA]
    [+0x000] DpcList          [Type: _KDPC_LIST]
    [+0x010] DpcLock          : Unable to read memory at Address 0x1030f4
    [+0x018] DpcQueueDepth    : Unable to read memory at Address 0x1030fc
    [+0x01c] DpcCount         : Unable to read memory at Address 0x103100
    [+0x020] ActiveDpc        : Unable to read memory at Address 0x103104
```

Other PatchGuard Smashing

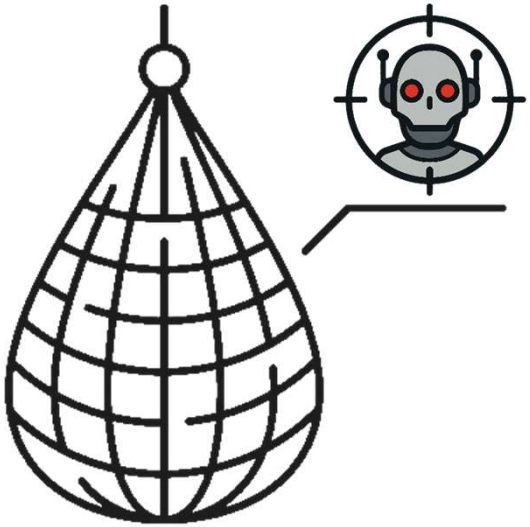
- Vanguard disables PatchGuard entry via KiSwInterrupt with an inline hook
- It also mutes currently running PatchGuard contexts → queuing infinite waits
- And corrupts DPC structures to break PatchGuard's deferred execution and checks

Defence Recap



Install NX net and page fault hook

Suppress Windows Patch Protection



Mapped Code falls straight into the net

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

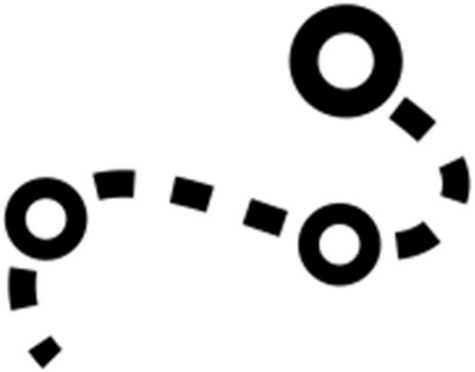
Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

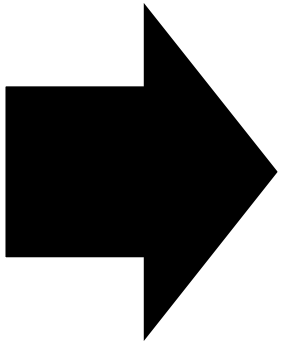
Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

A long-time issue

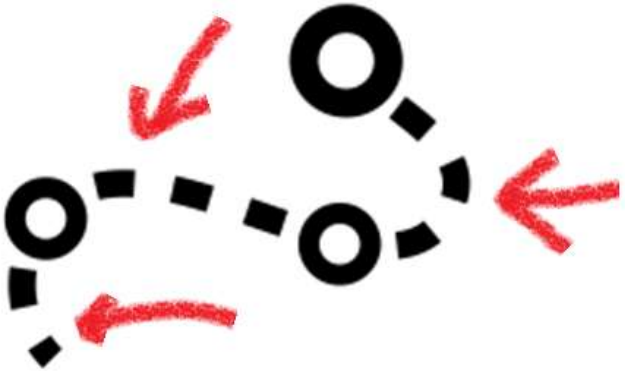


Cheats rely on **offsets** and **pointer paths** to know where important values or functions are located



ModuleBase
+ 0xFA
+ 0x103
+ 0x20
- 0x7
Health

Effect of Updates

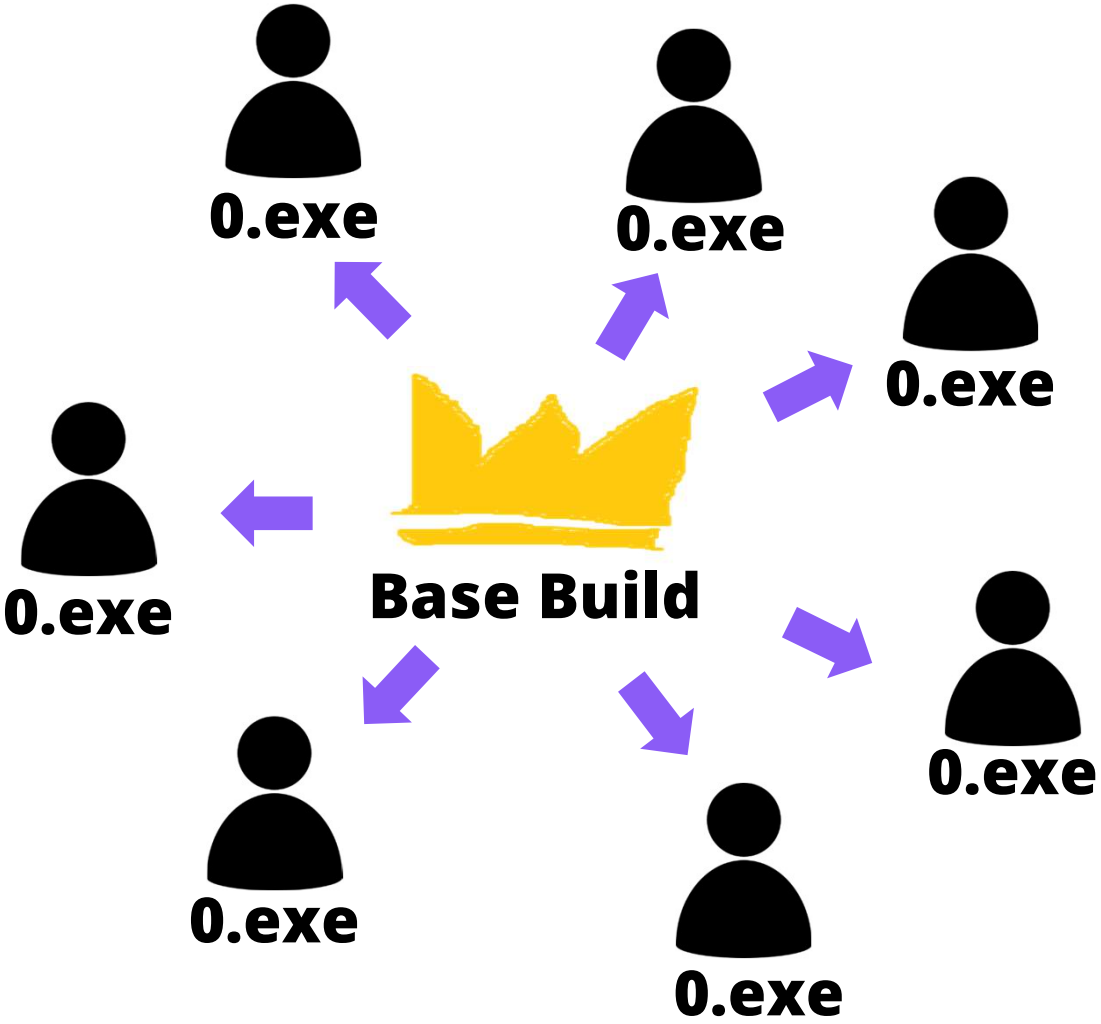


When a game gets updated/rebuilt the **pointer paths change** and must be freshly reversed

ModuleBase
+ 0xFA
+ 0x103
+ 0x20
- 0x7
CatPictures

What if this could be done for everyone all the time?

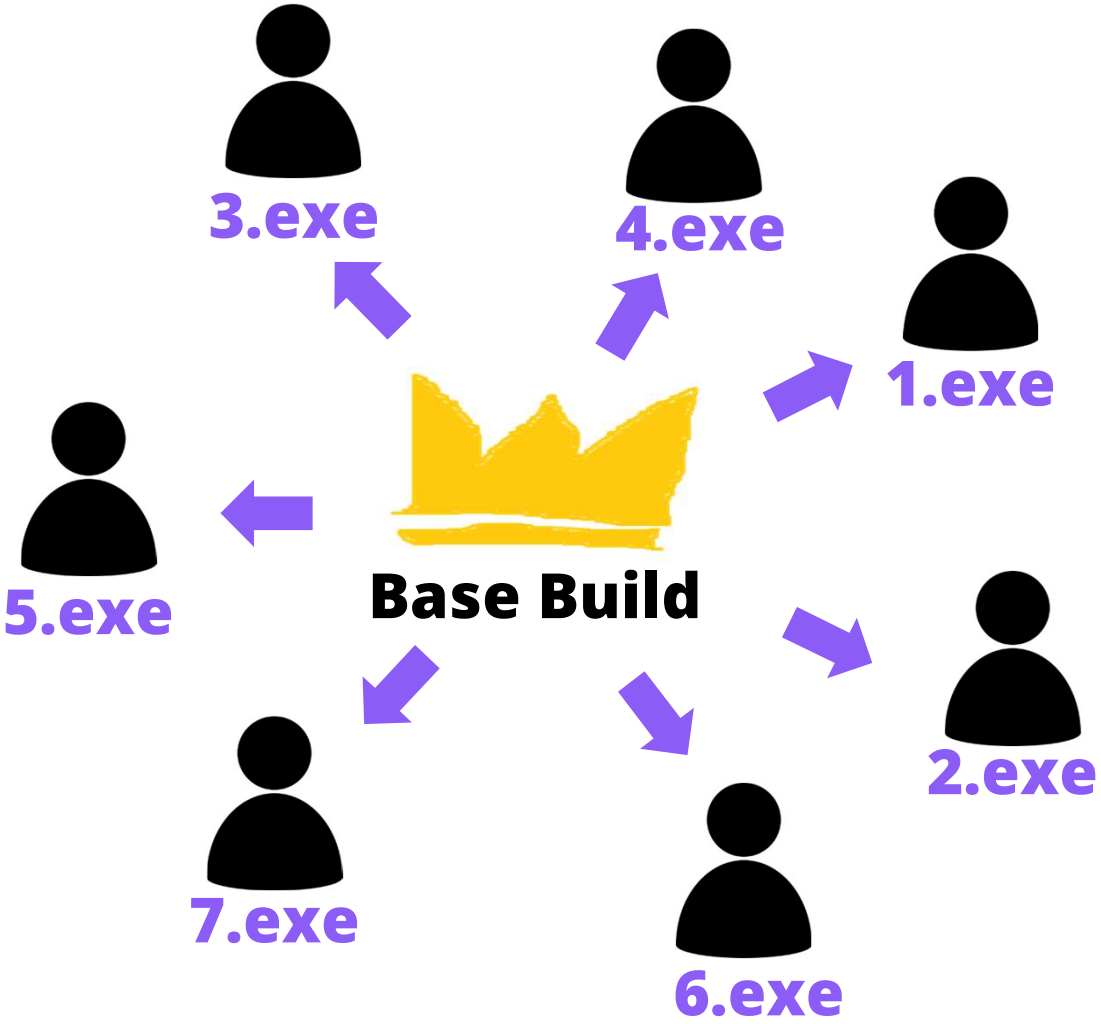
QB System



The Process

- Each client initially gets a base build

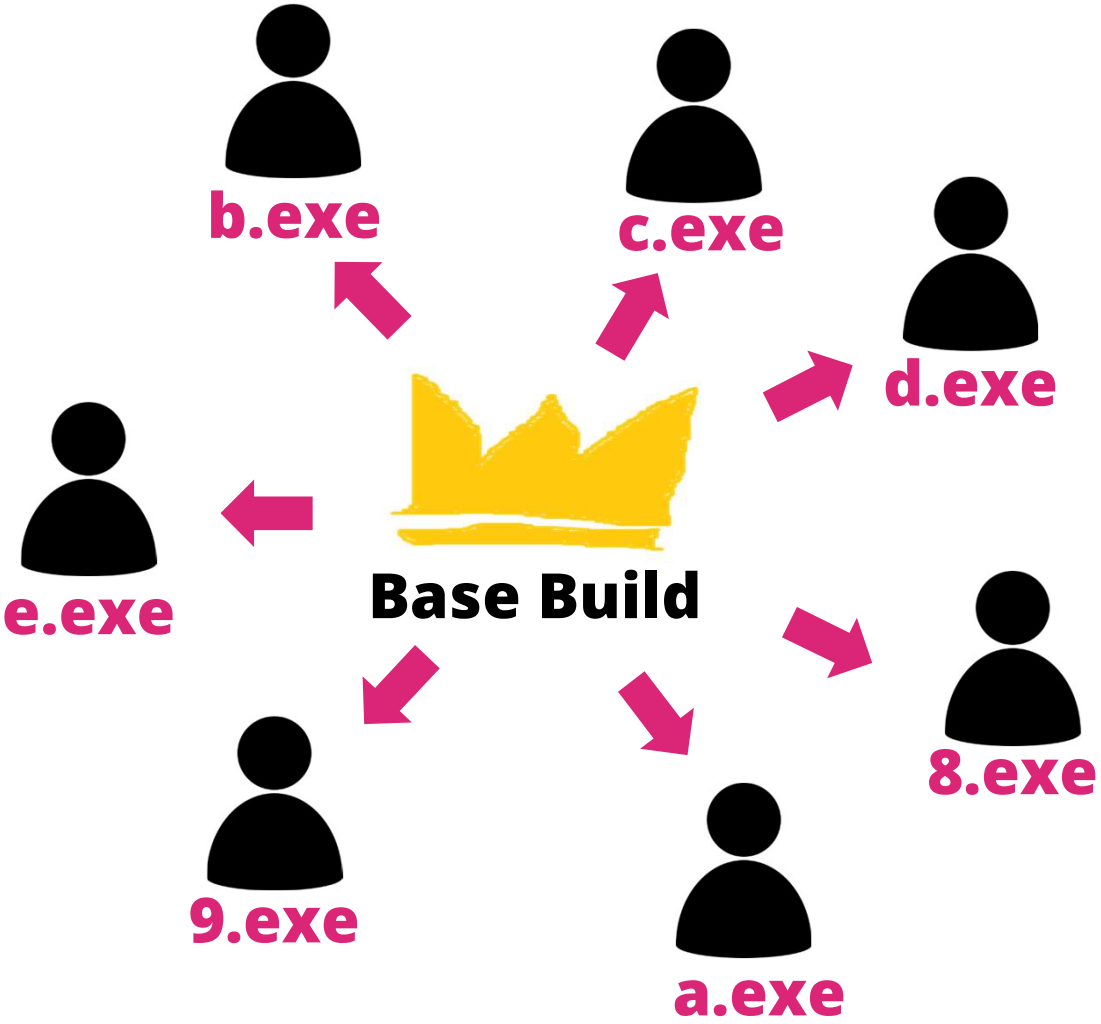
QB System



The Process

- Each client initially gets a base build
- First time run → patch is delivered

QB System

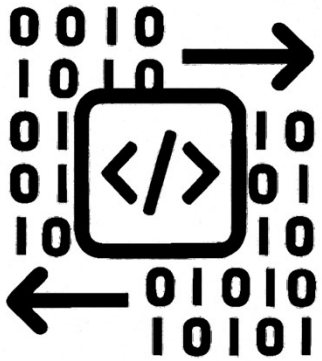


The Process

- Each client initially gets a base build
- First time run → patch is delivered
- Patch repeated at semi-regular intervals

What Changes

Offsets

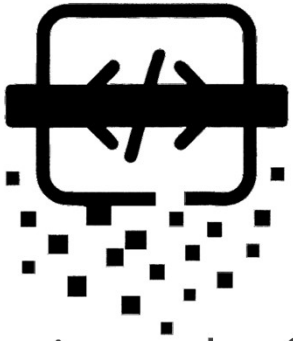


Specific memory offsets are shifted per build

Encryption

Decryption routines use unique keys and logic per build

Obfuscation



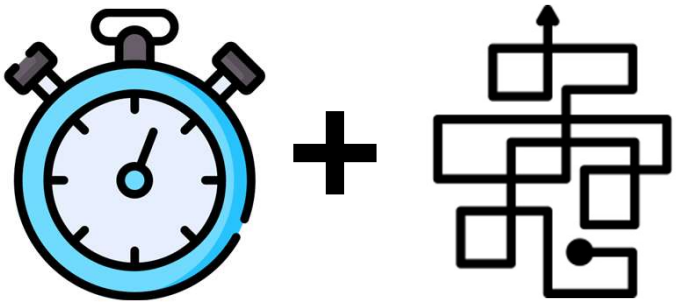
Code is reshuffled across builds, making static signature scanning unreliable

The Effect

Offsets are now unique to each build leaving two options for cheat developers:

1. Provide a unique cheat per unique build on the game (time consuming)

2. Develop cheats which signature scan or wrap key functions (time consuming and hard)

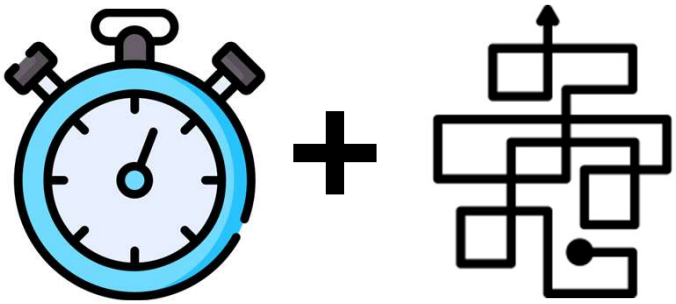


The Effect

Offsets are now unique to each build leaving two options for cheat developers:

1. Provide a unique **attack** per unique build on the target (time consuming)

2. Develop **attacks** which signature scan or wrap key functions (time consuming and hard)



Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

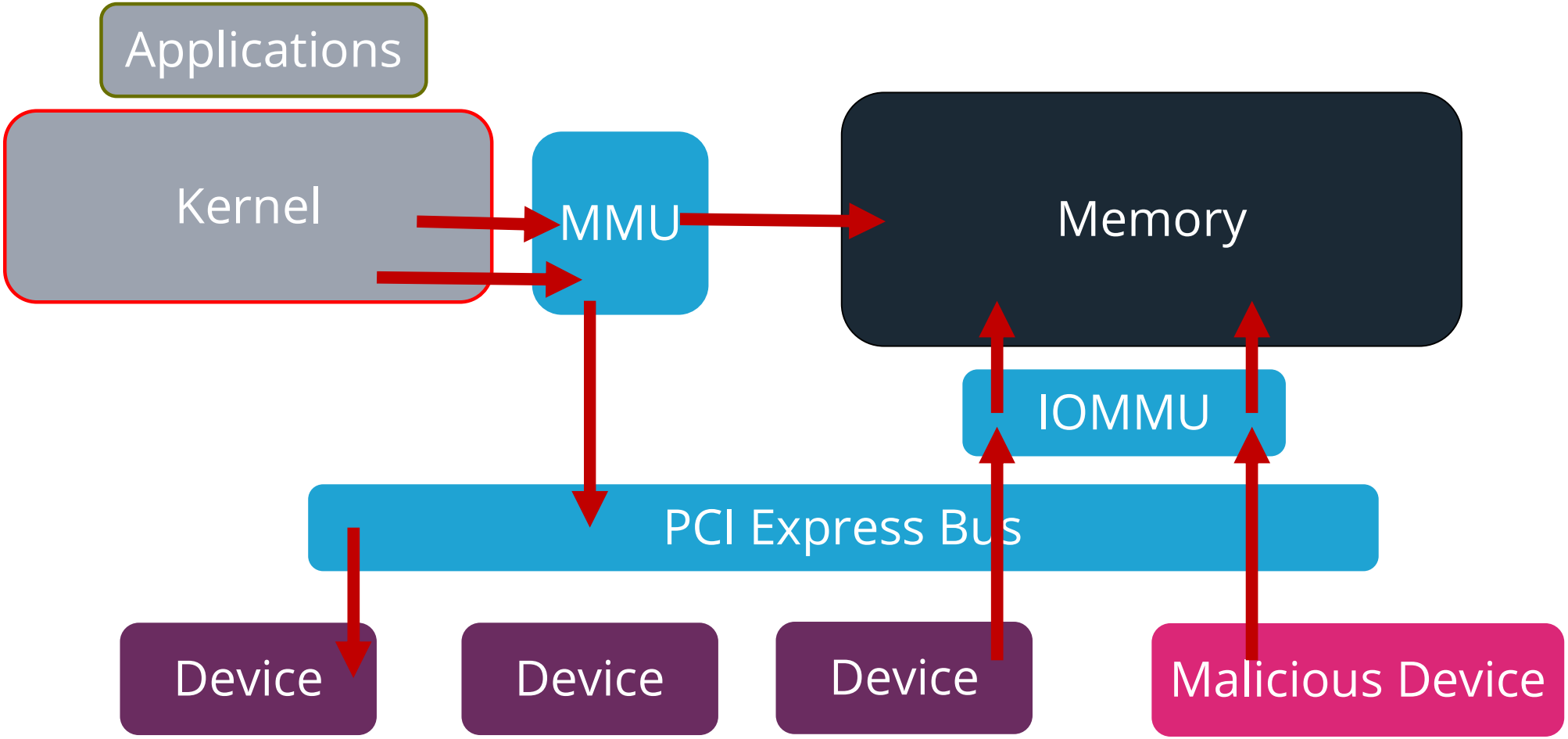
Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheat
- The next battleground
- Takeaways

Introduction to Memory Access



Direct Memory Access – Attack Examples

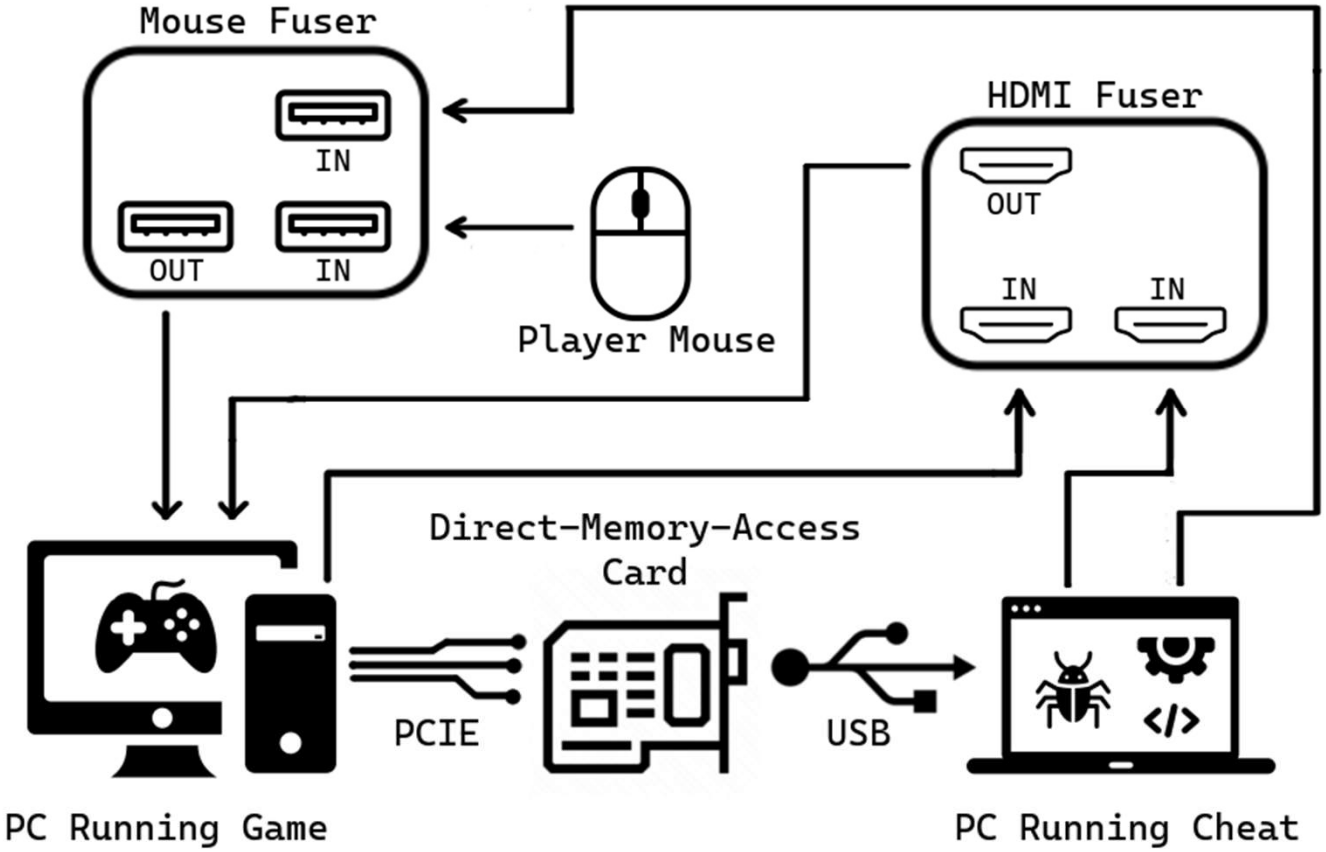
Bypass the lock screen on Windows 10

Connect a USB-C device and dump password (Thunderclap attack)

Cloud provider dumping memory of protected machine

Cheat at video games.

Direct Memory Access – Game Cheats



Detecting DMA Attacks

Anti-cheat scans all PCI devices

By walking the config space, simple checks can be done on serials, vendor IDs, etc.

Known DMA firmware can be flagged

Anything that instantly looks like a DMA card is disabled

Detecting DMA Attacks

DMA cards need to get sneaky

DMA cheats change their firmware to look innocent e.g., a network card.

Configuration Space – Vendor IDs, Supported Capabilities, Serials

Base Address Registers – Responding to reads/writes correctly (behaviour)

Interrupts – Messaged Signal Interrupts behave correctly

Detecting DMA Attacks

I'm a network card

Intel

Uhhhhhh, 0xFFFFFFFF?

What kind?

Do some networking

cy@

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Protecting Secrets in Memory

- Info stealing malware scans memory for credentials and credit card numbers.
- Easy Anti-Cheat and Vanguard have cool ways making important values in memory significantly harder to find.
- We present Vanguard's memory protection method

VALORANT

Security Research

I StealC You: Tracking the Rapid Changes To StealC

Agent Tesla Malware

Agent Tesla is an example of an advanced remote access trojan (RAT)

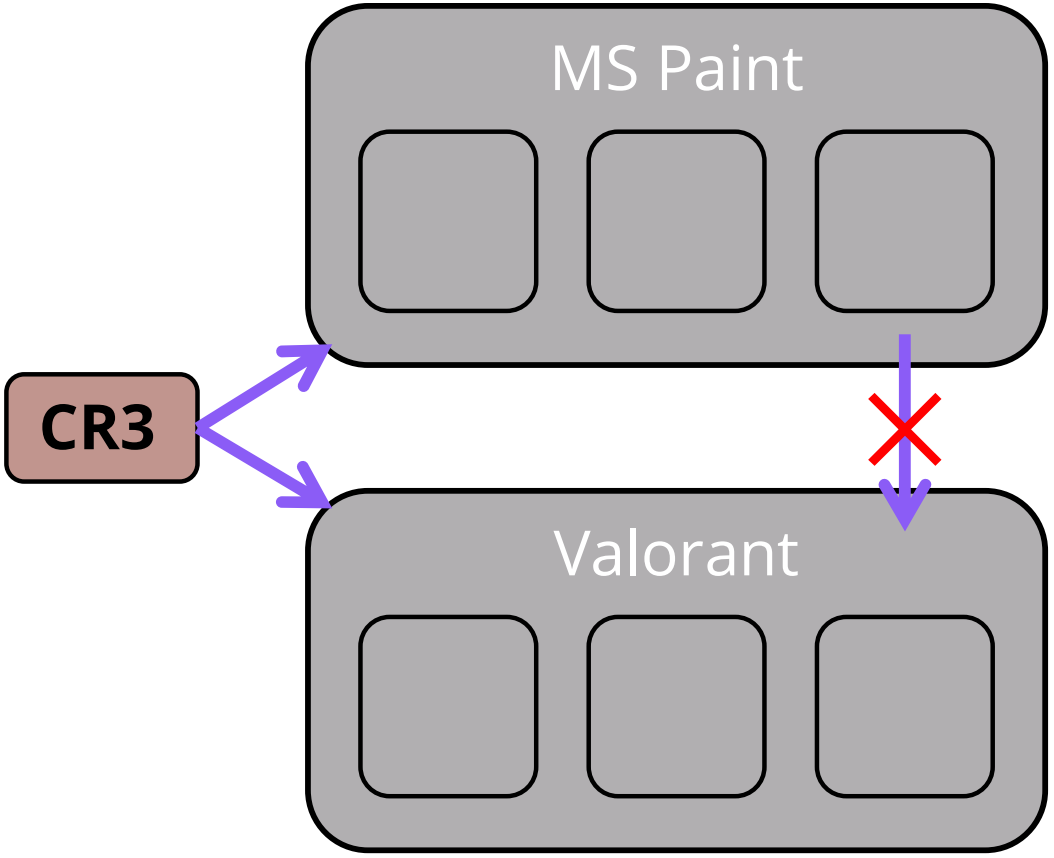
Credential theft • May 21 • 15 min read

Lumma Stealer: Breaking down the delivery techniques and capabilities of a prolific infostealer

By Microsoft Threat Intelligence, Microsoft Digital Crimes Unit and Microsoft Security Experts

Process Isolation

- Each Windows process runs in its own **virtual address space**
- This ensures one process cannot directly access another's memory
- The **CR3 register** holds the base address of the page map (**PML4**) for the current process
- Switching process = loading a new CR3 → changes the view of memory



Hooking the Scheduler

- Riot Vanguard **hooks the context switch** post operation
- When the context is changed, vanguard checks the properties of the new context
- Based off the result of these checks, **CR3 is written**

HalClearLastBranchRecordStack

```
10: kd> u 0xfffff8002ca6d0a0
vgk+0x5d0a0:
fffff800`2ca6d0a0 488bce          mov     rcx,rsi
fffff800`2ca6d0a3 e9e855ffff      jmp     vgk+0x52690
fffff800`2ca6d0a8 cc              int     3
fffff800`2ca6d0a9 cc              int     3
fffff800`2ca6d0aa cc              int     3
fffff800`2ca6d0ab cc              int     3
fffff800`2ca6d0ac cc              int     3
fffff800`2ca6d0ad cc              int     3
```

Context Switch Hook

Detour:

```
  push    r15
  mov     rbp,rsp
  sub     rsp,30h
  movzx   eax,byte ptr [vgk+0x7c179 (fffff800`2ca8c179)]
  lea     rbx,[vgk (fffff800`2ca10000)]
  mov     rdi,rcx
00 mov     rax,qword ptr [rbx+rax*8+7C188h]
  xor     rax,qword ptr [vgk+0x7c180 (fffff800`2ca8c180)]
  call    rax
  mov     rax,cr3
  cmp     rax,qword ptr [vgk+0x7c148 (fffff800`2ca8c148)]
  jne     vgk+0x533b1 (fffff800`2ca633b1)  Branch


  movzx   eax,byte ptr [vgk+0x78e31 (fffff800`2ca88e31)]
  mov     rcx,rdi
00 mov     rdx,qword ptr [rbx+rax*8+78E40h]
  xor     rdx,qword ptr [vgk+0x78e38 (fffff800`2ca88e38)]
  call    rdx
  cmp     rax,qword ptr [vgk+0x7c1b8 (fffff800`2ca8c1b8)]
  jne     vgk+0x533b1 (fffff800`2ca633b1)  Branch


  cmp     byte ptr [vgk+0x7c200 (fffff800`2ca8c200)],0
  je      vgk+0x52a61 (fffff800`2ca62a61)  Branch


  cmp     byte ptr [vgk+0x7c201 (fffff800`2ca8c201)],0
  je      vgk+0x533b1 (fffff800`2ca633b1)  Branch


  lea     rcx,[vgk+0x7c2b8 (fffff800`2ca8c2b8)]
  xor     bl,bl
  call    qword ptr [vgk+0x601a8 (fffff800`2ca701a8)]
  mov     r8d,dword ptr [vgk+0x7c1d8 (fffff800`2ca8c1d8)]
  cmp     r8d,200h
  je      vgk+0x52757 (fffff800`2ca62757)  Branch
```

```
xor     edx,edx
test    r8d,r8d
je      vgk+0x52759 (fffff800`2ca62759)  Branch


mov     rax,qword ptr [vgk+0x7c1e0 (fffff800`2ca8c1e0)]
cmp     rdi,qword ptr [rax+rdx*8]
je      vgk+0x52757 (fffff800`2ca62757)  Branch


inc     edx
cmp     edx,r8d
jb      vgk+0x52741 (fffff800`2ca62741)  Branch


jmp     vgk+0x52759 (fffff800`2ca62759)  Branch


mov     bl,1


lea     rcx,[vgk+0x7c2b8 (fffff800`2ca8c2b8)]
call    qword ptr [vgk+0x601c0 (fffff800`2ca701c0)]
test    bl,bl
je      vgk+0x533b1 (fffff800`2ca633b1)  Branch
```

...

Custom Handler

Context Switch Hook – Pseudocode

If:

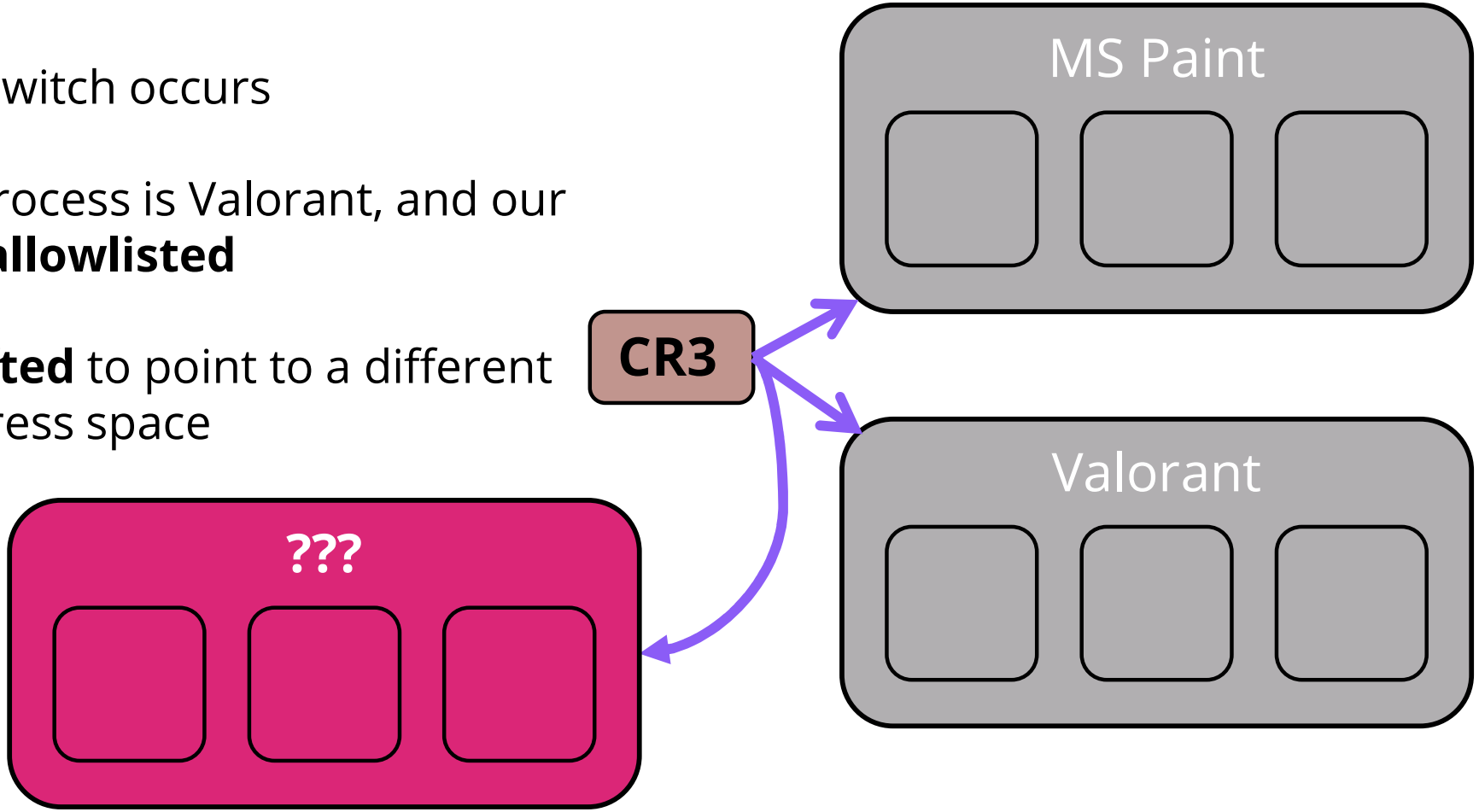
1. The new address space is for Valorant
2. The new thread belongs to the Valorant process
3. The thread belongs to a predefined allowlist

Then:

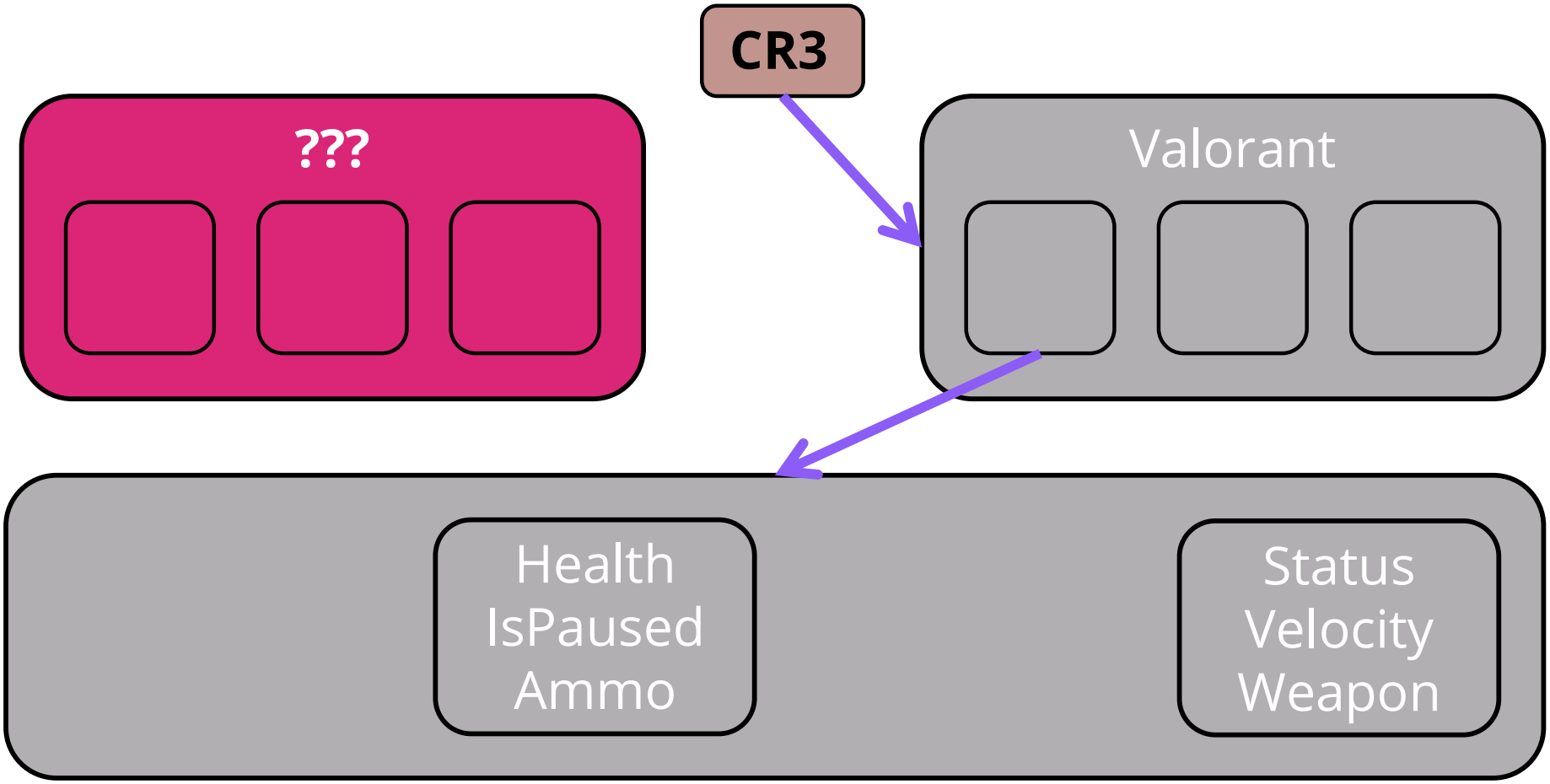
Jump to custom handler -> switch to secret CR3

Process Isolation

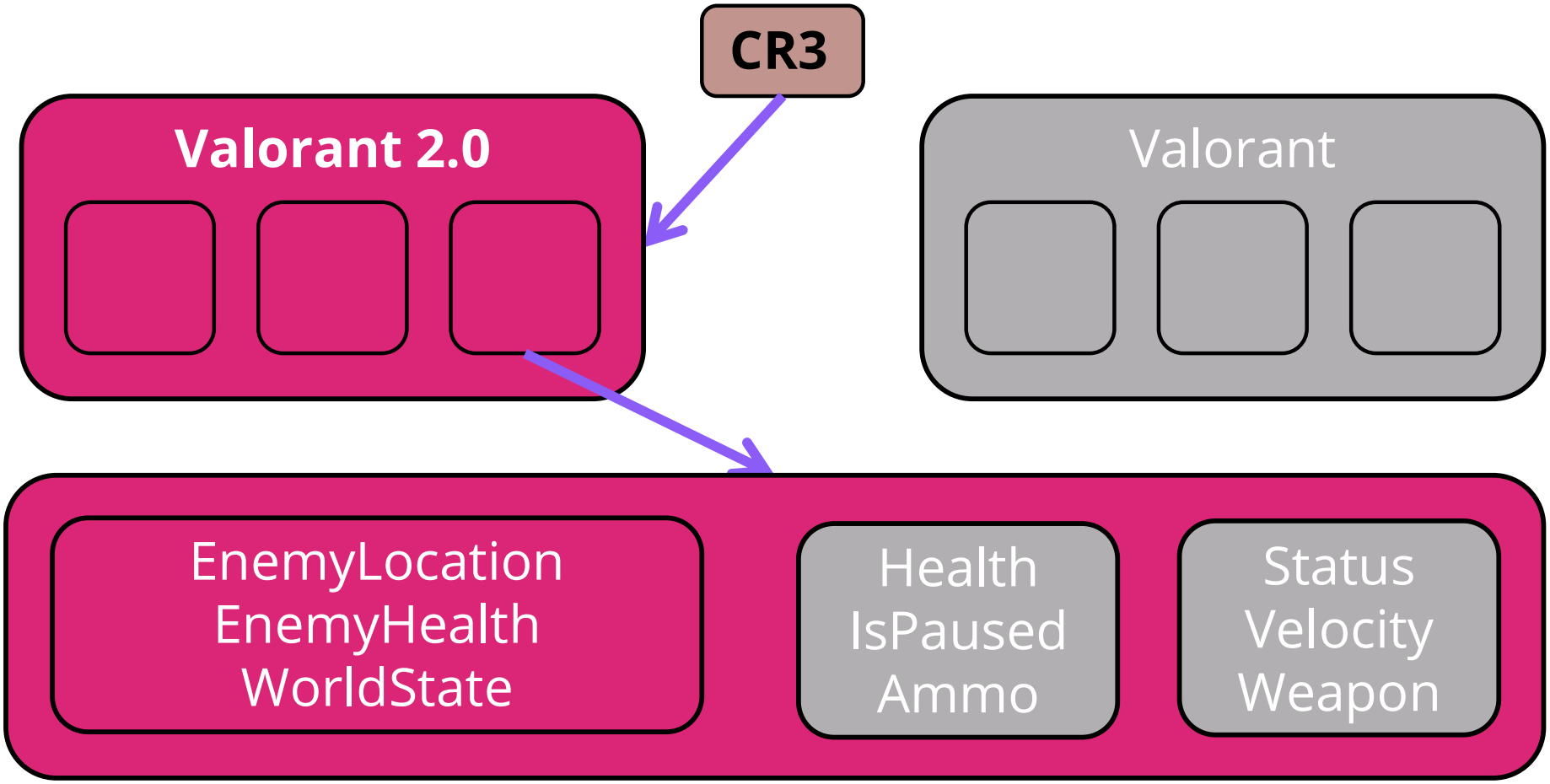
- A context switch occurs
- Our new process is Valorant, and our **thread is allowlisted**
- **CR3 is shifted** to point to a different PML4/address space



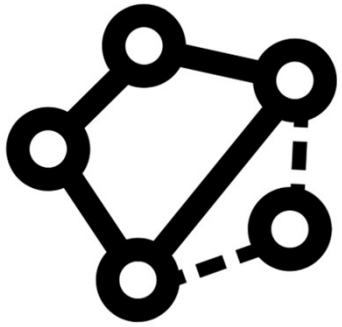
Process Isolation



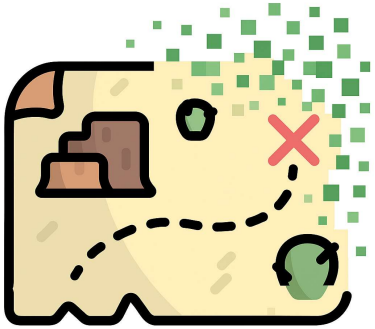
Process Isolation



Defence Recap



Augment the scheduling system



Redirect trusted threads to a different page map

Creating an invisibility cloak for memory!

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

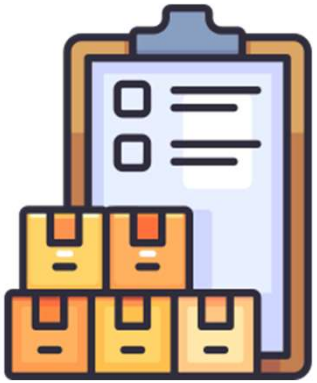
Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

Measurable Factors

Anti-Cheat Strength

Measured via grey box testing



Cheat Availability

Scraped from cheat selling sites

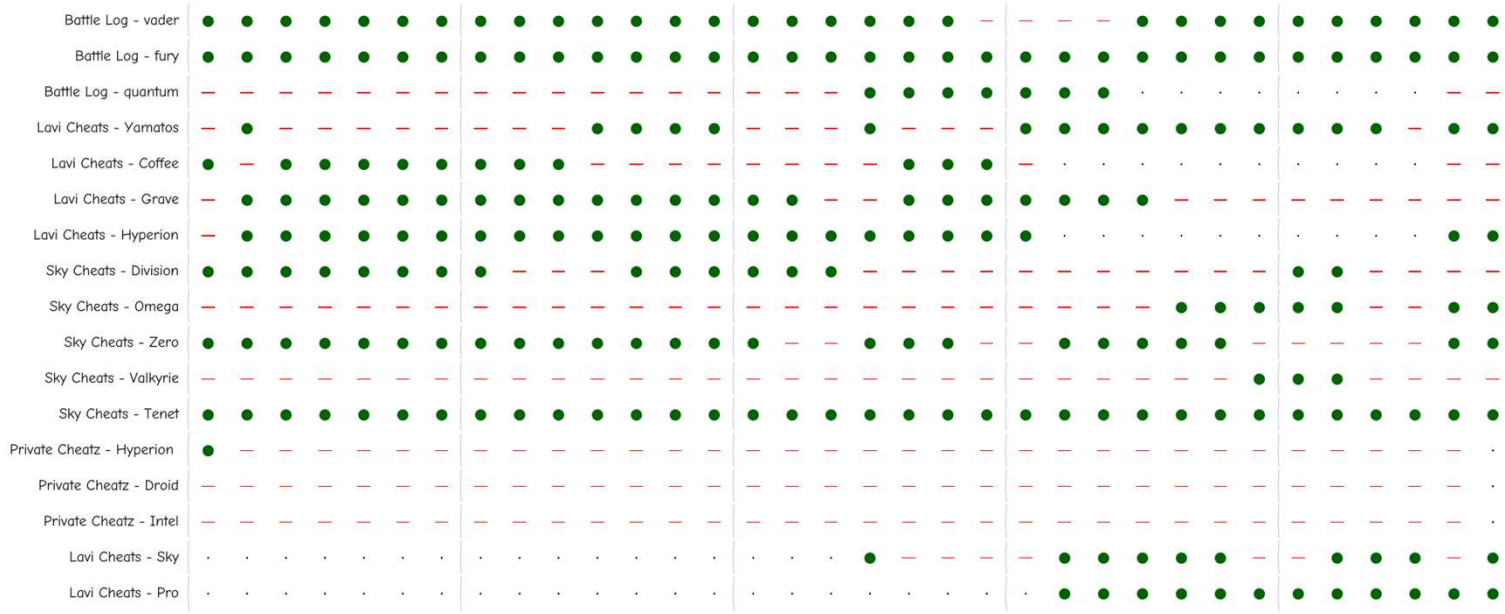
Cheat Price

Scraped from cheat selling sites

Game Popularity

Average players in a month (PC)

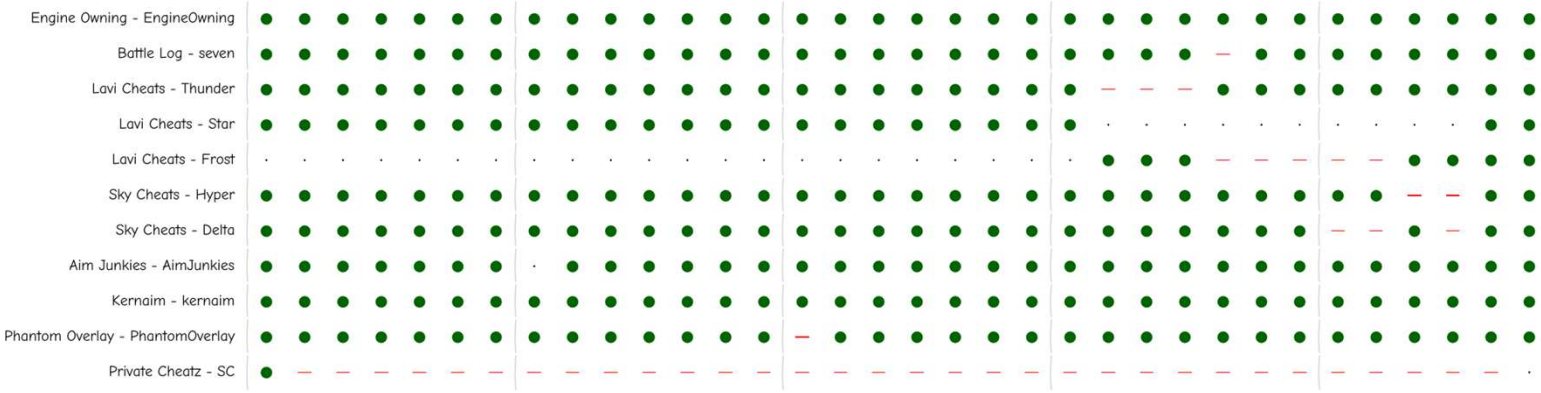
Market Observations



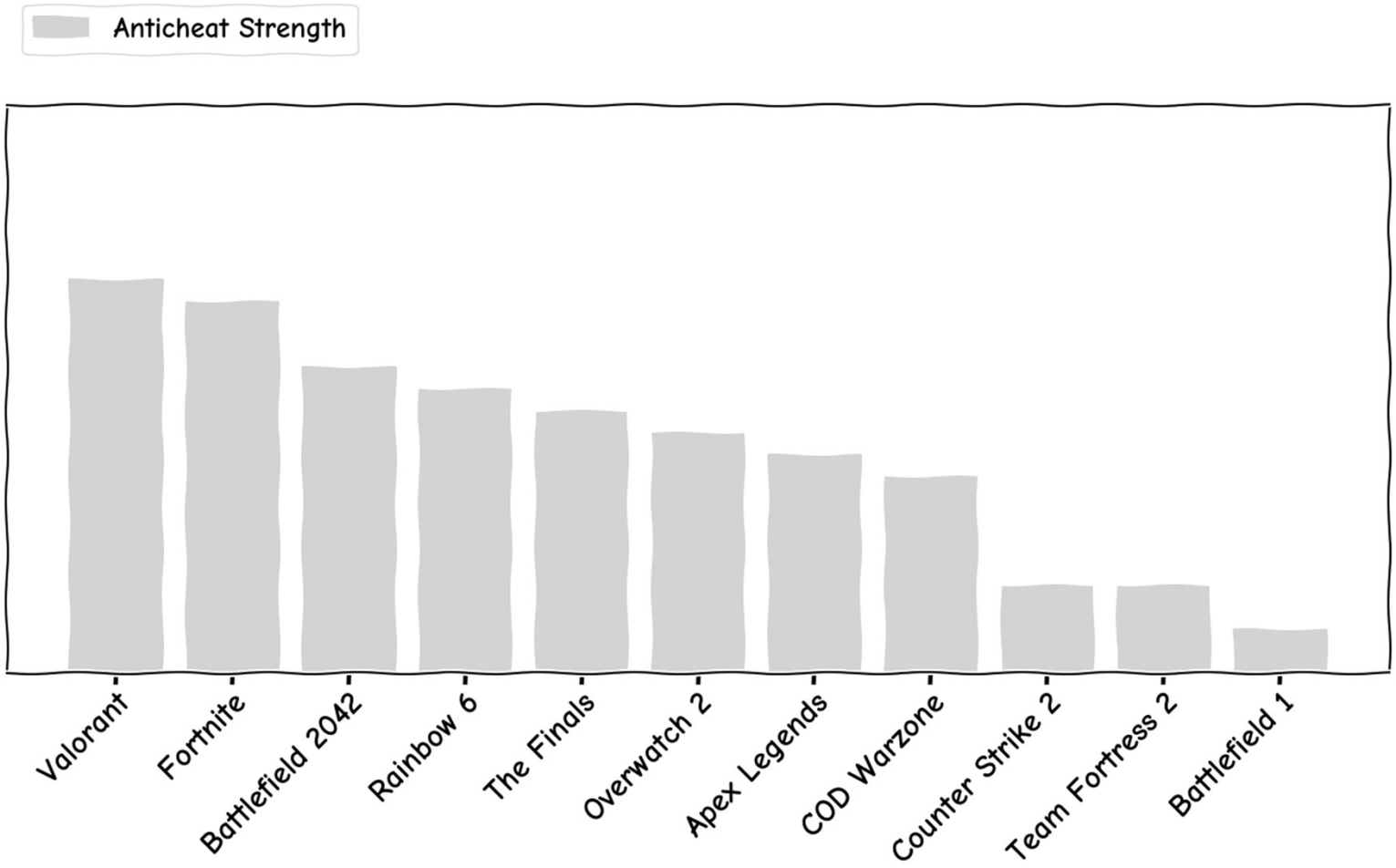
VALORANT

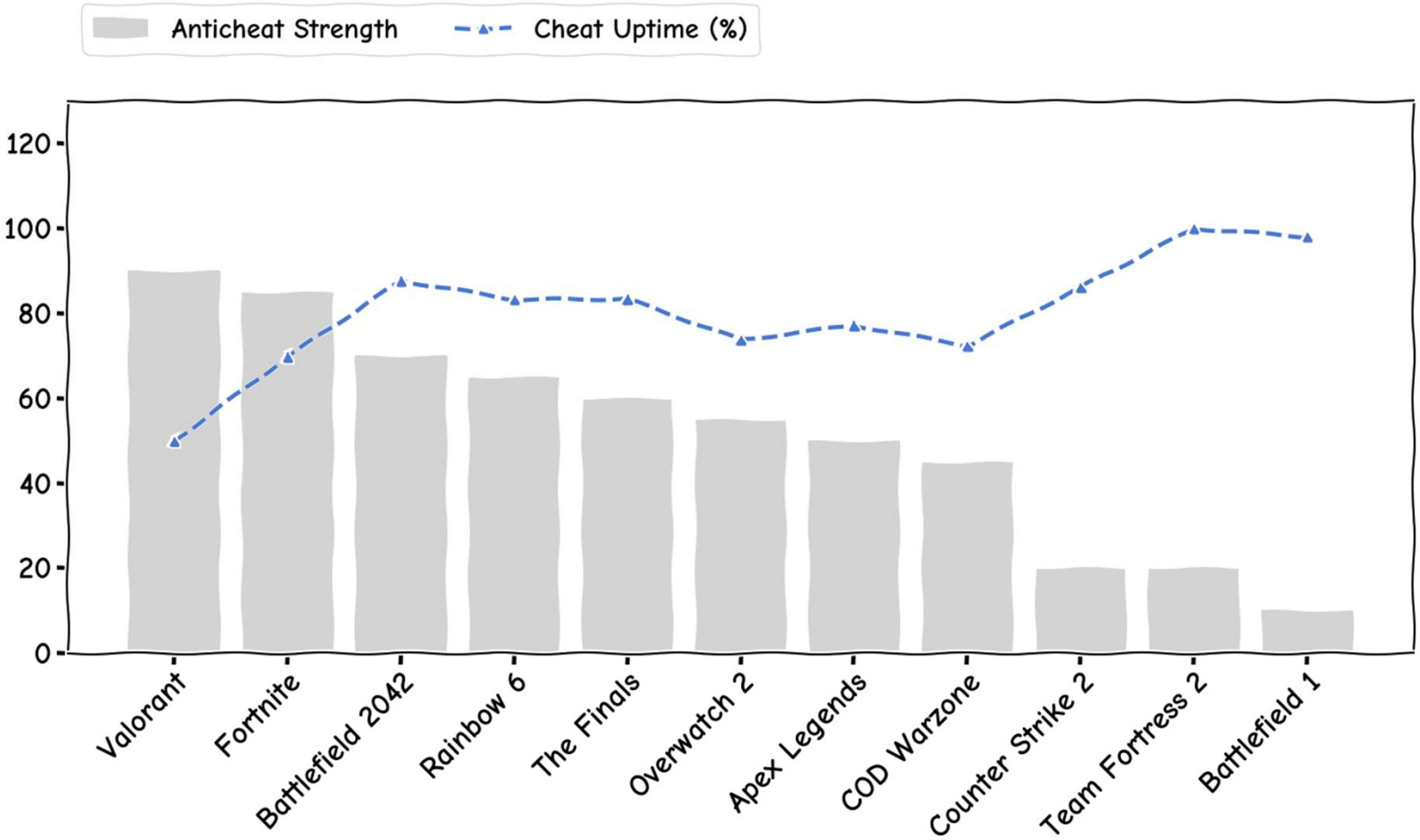
Mean Cheat Uptime = 50%

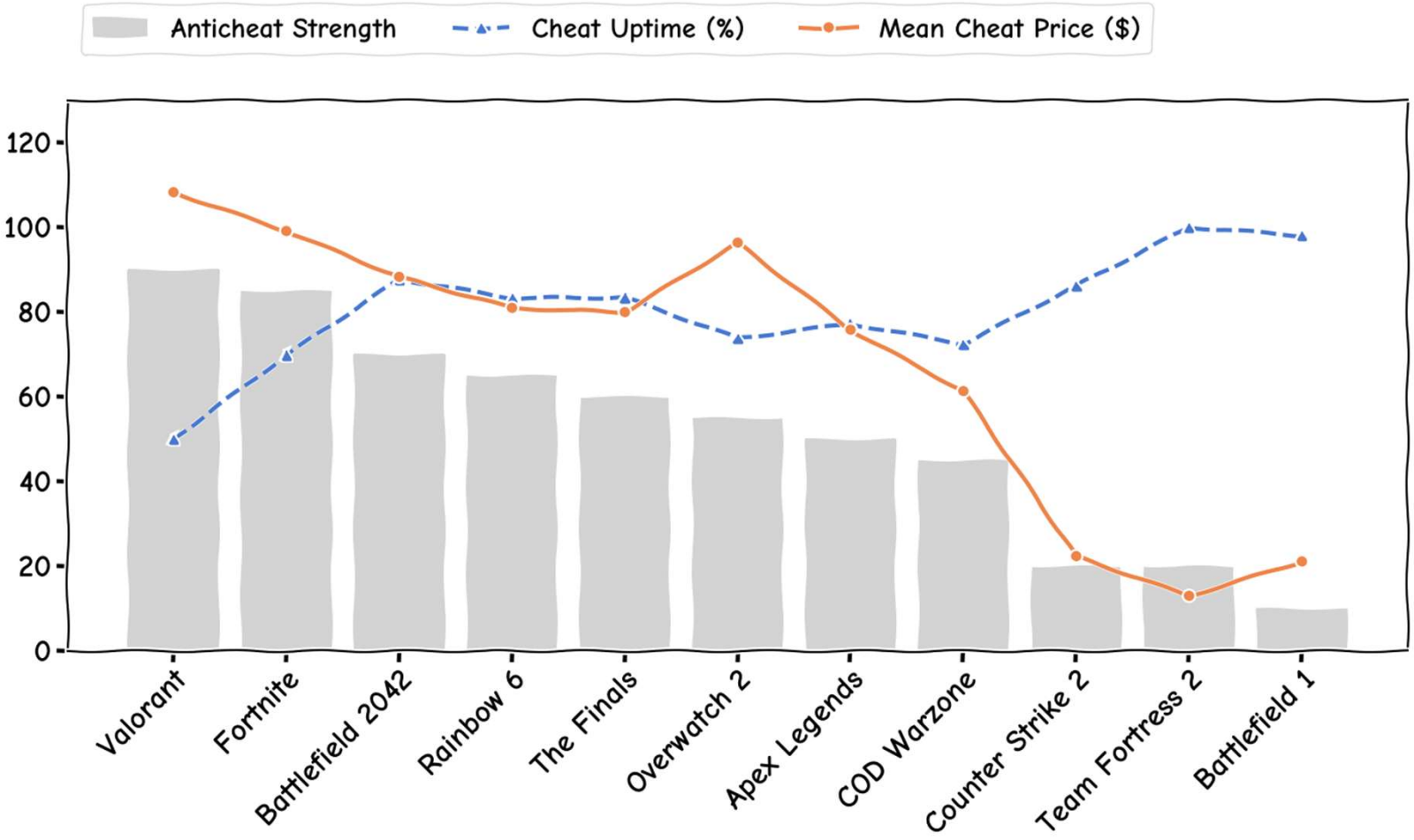
Mean Cheat Uptime = 86.2%

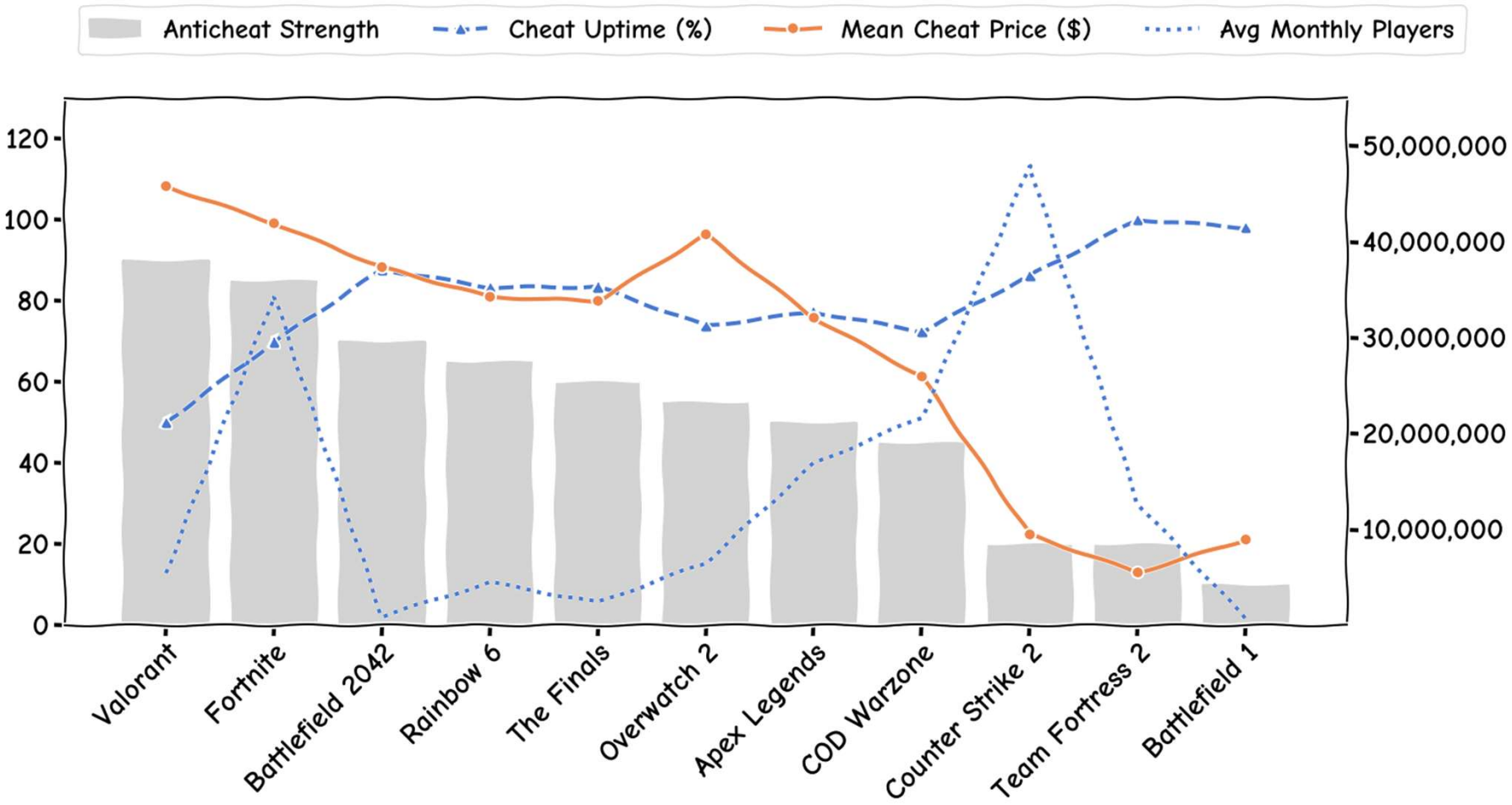


● Cheat Working | — Cheat Not Working | · Cheat not Available


Anticheat Strength
Valorant
Fortnite
Battlefield 2042
Rainbow 6
The Finals
Overwatch 2
Apex Legends
COD Warzone
Counter Strike 2
Team Fortress 2
Battlefield 1


Anticheat Strength
Cheat Uptime (%)
120
100
80
60
40
20
0
Valorant
Fortnite
Battlefield 2042
Rainbow 6
The Finals
Overwatch 2
Apex Legends
COD Warzone
Counter Strike 2
Team Fortress 2
Battlefield 1


Anticheat Strength
Cheat Uptime (%)
Mean Cheat Price ($)
120
100
80
60
40
20
0
Valorant
Fortnite
Battlefield 2042
Rainbow 6
The Finals
Overwatch 2
Apex Legends
COD Warzone
Counter Strike 2
Team Fortress 2
Battlefield 1


Anticheat Strength
Cheat Uptime (%)
Mean Cheat Price ($)
Avg Monthly Players
120
100
80
60
40
20
0
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
Valorant
Fortnite
Battlefield 2042
Rainbow 6
The Finals
Overwatch 2
Apex Legends
COD Warzone
Counter Strike 2
Team Fortress 2
Battlefield 1

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

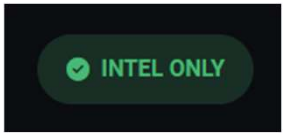
Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

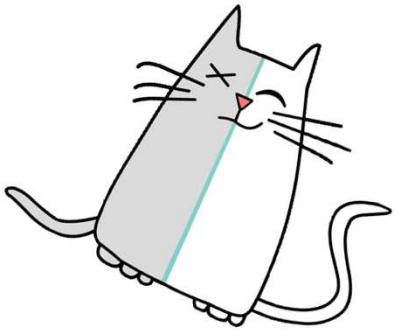

Usermode
Kernel
Hypervisor

Into Hyper-space

Cheats



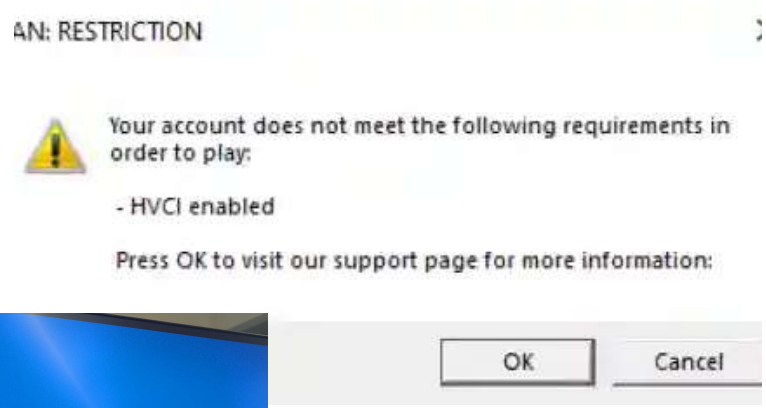
- **Supported CPU:** Intel only! (AMD not supported)
- **Supported OS:** Windows 10 Windows 11*
- **Publisher:** Ubisoft Montreal

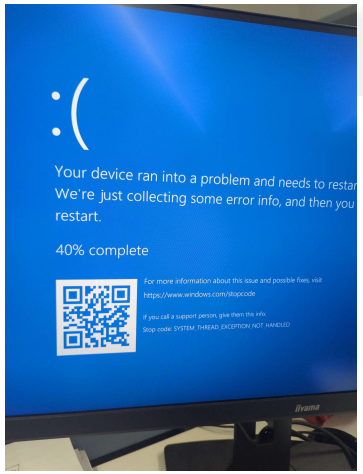


Us

Kernel
+
Hardware
=
Hypervisor
Read/Write

Anti-Cheats





”if they start requiring virtualization-based security to be on…we will leverage those features that protect Windows for us”

Talk Roadmap

Part I: Cheats & Anti-Cheats

- Introduction
- The world of game cheats
- Experiences with investigating anti-cheats

Part II: A Treasure Chest of Defenses

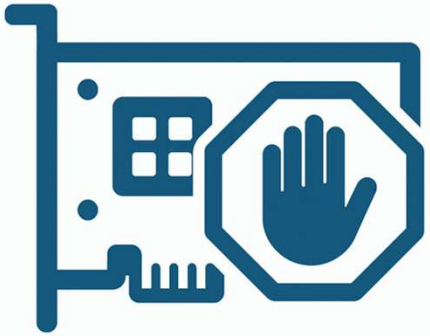
- Mitigating BYOVD
- Windows kernel hardening
- Software diversification
- Detecting rogue hardware
- Hiding memory

Part III: Insights & Takeaways

- Impacts of anti-cheats
- The next battleground
- Takeaways

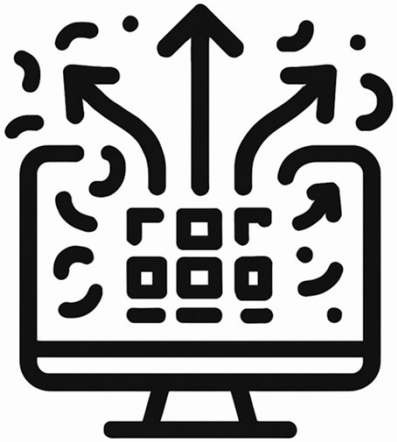
Cool Defences Deployed by Anti-cheats

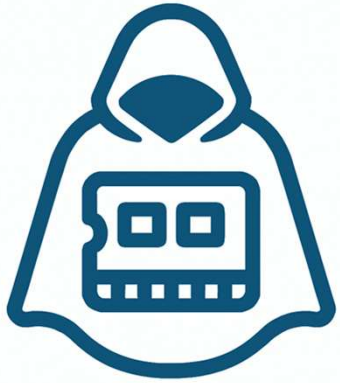
Detecting unsigned code in the kernel.



Stopping rouge hardware and DMA attacks.

Practical software diversification





A cloak of invisibility for memory.

Takeaways – BlackHat Sound Bytes

Anti-cheats implement some of best software defences.

A system is never as safe as when a user is playing Fortnite or Valorant.

If game devs can implement these defences, then so can we!

More information, updates, and code are available at:

https://game-research.github.io/

Questions?