Get high as a Threat Actor

Rootkits and Kernel security

Marcelo Toran
September 2024

Who am I?

Red Team edition
"Careful, it can break stuff"

Mountain bike edition
"Careful, it can get broken"

Marcelo Toran
Red Teamer @ Swiss Re
(@spamv)

Talk expectations vs reality

What is this about?

- Try to understand **what Threat Actors do in Kernel** space
- Learn about the different **Kernel security features**
- Develop a **methodology to “navigate”** through these security boundaries
- **Detection** methods and **mitigation** possibilities

Who is this talk for?

Missions of the day

MS KERNEL SECURITY FEATURES ★★★★★

MS PREINSTALLED DRIVERS 0-DAY ★★★☆☆

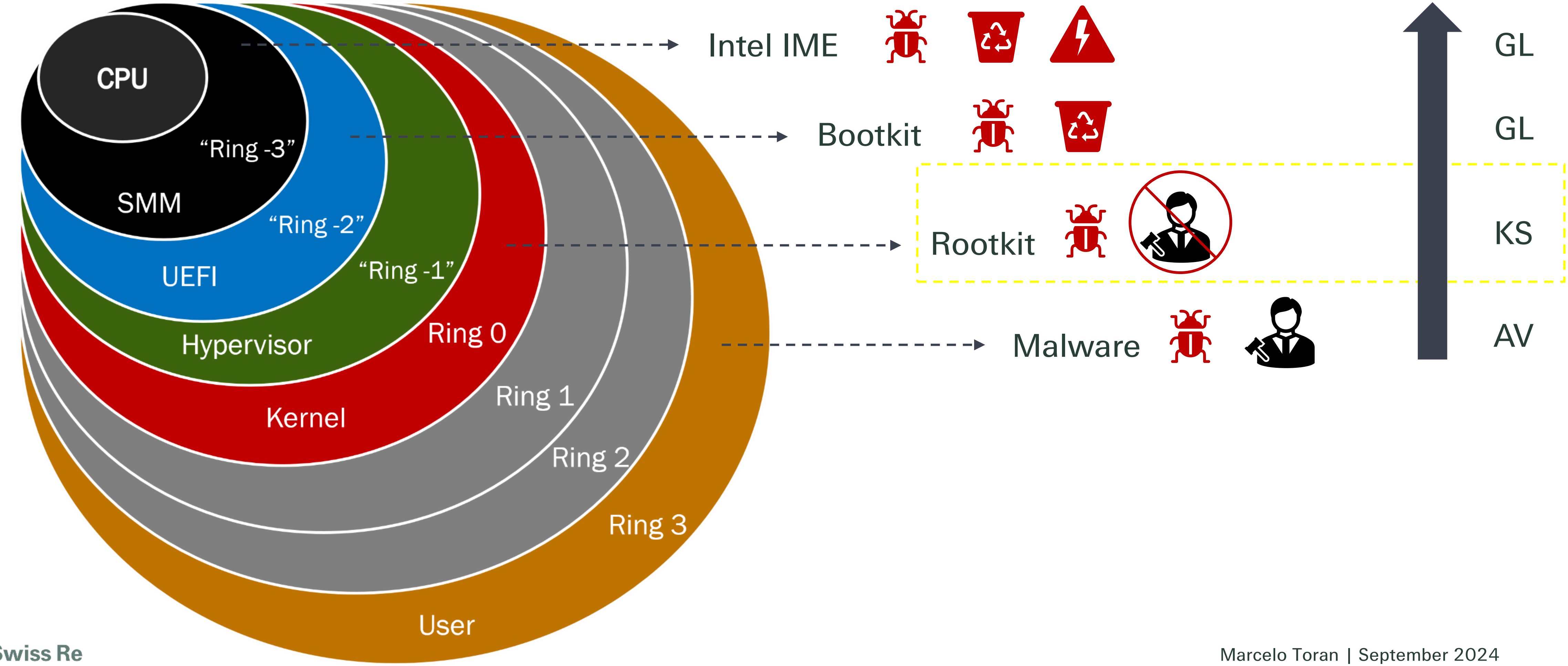
MS PREINSTALLED DRIVERS N-DAY ★★☆☆☆

BYOVD N-DAY ★☆☆☆☆

DETECTION ☆☆☆☆☆

MS Kernel Security Features ★★★★★

Rootkits environment overview



Why Kernel Rootkits are so interesting for Threat Actors?

- Hide files → Hide rootkit (driver)
- Hide registries → Hide rootkit (service)
- Hide process → Hide rootkit (service)
- Kernel shellcode execution → Freedom to interact with the system
- Disable EDRs → Lower detection capabilities
- Limit software execution → Lower detection capabilities
- Disable ETW → Lower detection capabilities
- Network traffic manipulation → Lower detection capabilities / Sensitive information gathering
- Steal token → Privilege escalation
- Disable Updates / Windows Recovery → Difficult infection clean up
- BIOS/UEFI modification → OS clean up persistence
- Callbacks hooking → General OS malfunction

Kernel security features

Driver Signed Enforcement (DSE)

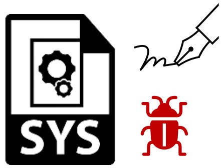
Kernel Patch Protection (KPP) – Patch Guard

Virtualized-based Security (VBS)

- Kernel Data Protection (KDP)
- Secure Kernel Patch Guard (SKPG) – Hyper Guard
  - Kernel Control Flow Guard (KCFG)
  - Kernel Control-Flow Enforcement Technology (KCET)

Kernel security features

Driver Signed Enforcement (DSE)



- Only **digitally signed** drivers are allowed
- DSE enabled by default since Windows Vista (2007)

- **Bypass** to load unsigned drivers:
  - C:\Windows\System32\CI.dll → Code Integrity DLL responsible for validating the signatures
  - It is possible to **patch CI.dll** exported function (**CiOptions**) to return the value as if every driver would be signed.

Kernel security features

Driver Signed Enforcement (DSE) → Only digitally signed drivers are allowed

Kernel Patch Protection (KPP) – Patch Guard

Virtualized-based Security (VBS)

- Kernel Data Protection (KDP)
- Secure Kernel Patch Guard (SKPG) – Hyper Guard
  - Kernel Control Flow Guard (KCFG)
  - Kernel Control-Flow Enforcement Technology (KCET)

Kernel security features

Kernel Patch Protection (KPP) – Patch Guard

- Enabled by default in Windows 10/11
  - **PatchGuard** performs **periodic integrity checks** on the Kernel. It ensures that the Kernel code and key structures have not been altered.
  - DSE CiOptions patch would be detected by KPP

- **Bypass**:
  - Race condition, after patching the function (and loading the driver), quickly **revert to the original state** before KPP integrity check triggers.

Kernel security features

- Driver Signed Enforcement (DSE) → Only digitally signed drivers are allowed
- Kernel Patch Protection (KPP) – Patch Guard → Performs periodic integrity checks on the Kernel code.
- Virtualized-based Security (VBS)
  - Kernel Data Protection (KDP)
  - Secure Kernel Patch Guard (SKPG) – Hyper Guard
  - Hypervisor Code Integrity (HVCI)
    - Kernel Control Flow Guard (KCFG)
    - Kernel Control-Flow Enforcement Technology (KCET)

Kernel security features

- **VBS** is enabled by default in Windows 10/11 (if Intel VT-x or AMD-V found)
- Protected by Secure Boot (UEFI lock)
- Provides a **hypervisor protected environment** running on a **second Secure Kernel** (VLT1) which can't be touched by the traditional Kernel (VLT0) running in ring-0

Virtualized-based Security (VBS)

Kernel Data Protection (KDP)

Secure Kernel Patch Guard (SKPG) – Hyper Guard

Kernel security features

- **VBS** is enabled by default in Windows 10/11 (if Intel VT-x or AMD-V found)
- Protected by Secure Boot (UEFI lock)
- Provides a **hypervisor protected environment** running on a **second Secure Kernel** (VLT1) which can't be touched by the traditional Kernel (VLT0) running in ring-0

Virtualized-based Security (VBS)

Kernel Data Protection (KDP)

Secure Kernel Patch Guard (SKPG) – Hyper Guard

- Enabled when VBS enabled
- **KDP** protects **read-only data** in the Kernel, making sure that specific data structures or memory regions cannot be modified. Those functions are **protected on demand by MmProtectDriverSection** API.
- **Bypass**:
    - Access other **deeper** (not exported) **functions**

Kernel security features

- **VBS** is enabled by default in Windows 10/11 (if Intel VT-x or AMD-V found)
- Protected by Secure Boot (UEFI lock)
- Provides a **hypervisor protected environment** running on a **second Secure Kernel** (VLT1) which can't be touched by the traditional Kernel (VLT0) running in ring-0

Virtualized-based Security (VBS)

Kernel Data Protection (KDP)

Secure Kernel Patch Guard (SKPG) – Hyper Guard

- Periodically performs **integrity checks** within the Kernel (in **VTL1**) to verify that critical Kernel structures have not been tampered.
- **Not enable by default**
- Performance impact

- Enabled when VBS enabled
- **KDP** protects **read-only data** in the Kernel, making sure that specific data structures or memory regions cannot be modified. Those functions are **protected on demand by MmProtectDriverSection** API.
- **Bypass**:
  - Access other **deeper** (not exported) **functions**

Kernel security features

- Driver Signed Enforcement (DSE) → Only digitally signed drivers are allowed
- Kernel Patch Protection (KPP) – Patch Guard → Performs periodic integrity checks on the Kernel code.
- Virtualized-based Security (VBS)
  - Kernel Data Protection (KDP) → Protects read-only data in Kernel, making sure that those structures can't be modified.
  - Secure Kernel Patch Guard (SKPG) – Hyper Guard → Periodically performs integrity checks within the Kernel (in VTL1)
  - Hypervisor Code Integrity (HVCI) →
    - Kernel Control Flow Guard (KCFG) →
    - Kernel Control-Flow Enforcement Technology (KCET) →

BYOVD N-DAY ★☆☆☆☆

- Bring Your Own Vulnerable Driver (BYOVD) N-DAY → known publicly released vulnerabilities

- Purpose
  - Exploit driver-specific vulnerabilities
  - **Deploy unsigned drivers**
  - Execute Kernel functions

- Attack steps
  - Find a vulnerable driver
  - Comply with security policies
  - Deploy the driver on the target system
  - Exploit the vulnerability

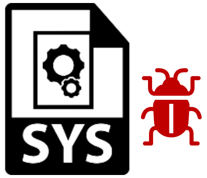
- Approach to bypass Kernel security features

Attack steps: Find a vulnerable driver

www.github.com | www.unknowncheats.me | www.loldrivers.io | www.virustotal.com

Find known exploits

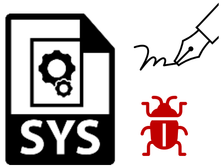
Find known vulnerable driver



N-DAY

Attack steps: Comply with security policies

- Digitally **signed**



- Not in MS vulnerable drivers **Blocklist**

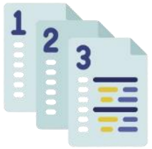
driversipolicy.p7b

+

=

github.com/mattifestation/WDACTools

github.com/trailofbits/HVCI-loldrivers-check



- EDR **static detection**

Attack steps: Deploy the driver on the target system

- Admin access rights / privilege escalation

Attack steps: Exploit the vulnerability

- **IOCTL** (Input/Output Control) misuse
  - Kernel **physical memory Read / Write**
  - Expose Windows Native API functions
  - **PreviousMode** overwrite

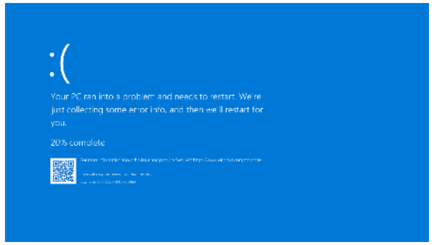
- Time-of-Check to Time-of-Use (**TOCTOU**)

- Buffer overflow

Demo approach:

C:\Windows\System32\CodeIntegrity\driversipolicy.p7b


```xml
<Deny ID="ID_DENY_D_0008" Hash="0289FE12E675101CEE03934C1AF5CB73069A12170A88BD051E31A292B97F701B" />
<Deny ID="ID_DENY_D_0009" Hash="02A7E085631ECFE031B76AFA883A266C850ED61B" />
<Deny ID="ID_DENY_D_000A" Hash="03358C32022F0C9811D40A0BDF2BD58CAB170442" />
<Deny ID="ID_DENY_D_000B" Hash="034DB61D844ADA030BB9173F4B7448DD6CC355B4429266F5ABBA6AFDB481128B" />
<Deny ID="ID_DENY_D_000C" Hash="0358BCBA83349CB23EA44D5C36B9E22ADAEC8D94" />
<Deny ID="ID_DENY_D_000D" Hash="038F39558035292F1D794B7CF49F8E751E8633DAEC31454FE85CCCBEA83BA3FB" />
<Deny ID="ID_DENY_D_000E" Hash="0391107305D76EB9DDF1A5B3B3C50DA361E8AB35B573DBD19BF9383436B9303E" />
<Deny ID="ID_DENY_D_000F" Hash="03F0DD3124EC3A4BB6D30865A488F54E74DED699" />
<Deny ID="ID_DENY_D_0010" Hash="04ACDCE5167AF1D3F4F2AAEEBA143D7B610B275E29850F52C6DBA62E6359272D" />
<Deny ID="ID_DENY_D_0011" Hash="05234D1A267C9B6C1754272658FBEBB22633CAC0" />
<Deny ID="ID_DENY_D_0012" Hash="053E36AF7ECDDB09ED3C1844F43B10DE2156EDD1" />
<Deny ID="ID_DENY_D_0013" Hash="05736AB8B48DF84D81CB2CC0FBDC9D3DA34C22DB67A3E71C6F4B6B3923740DD5" />
<Deny ID="ID_DENY_D_0014" Hash="057E6A58E3515E56EAB85CCB8EC5086552B7DE98C886C37F6A5284C002615565" />
<Deny ID="ID_DENY_D_0015" Hash="05AC1C64CA16AB0517FE85D4499D08199E63DF26" />
```


LPE + EXE + SYS + SYS + != 

LPE

SaharaDriver
- Loader to bypass Kernel security features

Vulnerable signed driver
- Kernel Read
- Kernel Write

Unsigned driver with all the malicious artifacts

Not present in MS drivers blocklist (last update)

No BSOD

RTCore64.sys
MD5:
AA9ADCF640
08E13D7E68B
56FDD307EAD

GPU

Driver loader weaponization

	SaharaDriver Loader	Enforced by Windows
Driver Signed Enforcement (DSE)	Implemented	By default
Kernel Patch Protection (KPP) – Patch Guard	Implemented	By default
Virtualized-based Security (VBS)		
Kernel Data Protection (KDP)	Implemented	If VBS enabled
Secure Kernel Patch Guard (SKPG) – Hyper Guard	No bypass known	No

Hypervisor Code Integrity (HVCI)

Kernel Control Flow Guard (KCFG)

Kernel Control-Flow Enforcement Technology (KCET)

HVCI disabled for this demo

https://blog.xpnsec.com/gcioptions-in-a-virtualized-world/

Demo setup:

Windows 11 +  + Windows Defender + **VBS**

Demo requirements:

Demo (Houdini + Blaster)

01

Objectives Demo 2

- Execute some typical Threat Actor attack vectors on Kernel side

- Bypass Kernel security features (**DSE, KPP, KDP**) + **HVCI**

- Blaster 2 demo (terminate and block protected processes)

Kernel security features

- Driver Signed Enforcement (DSE) → Only digitally signed drivers are allowed
- Kernel Patch Protection (KPP) – Patch Guard → Performs periodic integrity checks on the Kernel code.
- Virtualized-based Security (VBS)
  - Kernel Data Protection (KDP) → Protects read-only data in Kernel, making sure that those structures can't be modified.
  - Secure Kernel Patch Guard (SKPG) – Hyper Guard → Periodically performs integrity checks within the Kernel (in VTL1)
  - Hypervisor Code Integrity (HVCI) →
    - Kernel Control Flow Guard (KCFG) →
    - Kernel Control-Flow Enforcement Technology (KCET) →

Kernel security features

- **HVCI** is **enabled by default in clean Windows 11 Enterprise** 23H2 installations
- Protected by Secure Boot (UEFI lock)
- Ensures that **only code with valid trusted signatures** can run in Kernel mode
- HVCI (VTL1) ensures that code in VTL0 can't be modified. Extended Page Tables (EPT) ensures that pages mapped as **read-execute can't be made writable** (as well as read-write not made executable). When **EPT detects a violation, it informs HVCI**, if it's in a critical Kernel memory page then HVCI would trigger a BSOD aka **DSE patching is not possible**.
- Bypass:
  - Use Kernel Return-Oriented Programming **(ROP) gadgets** to perform arbitrary operations instead of patching memory.

Hypervisor Code Integrity (HVCI)

Kernel Control Flow Guard (KCFG)

Kernel Control-Flow Enforcement Technology (KCET)

Kernel security features

- **HVCI** is **enabled by default in clean Windows 11 Enterprise** 23H2 installations
- Protected by Secure Boot (UEFI lock)
- Ensures that **only code with valid trusted signatures** can run in Kernel mode
- HVCI (VTL1) ensures that code in VTL0 can't be modified. Extended Page Tables (EPT) ensures that pages mapped as **read-execute can't be made writable** (as well as read-write not made executable). When **EPT detects a violation, it informs HVCI**, if it's in a critical Kernel memory page then HVCI would trigger a BSOD aka **DSE patching is not possible**.
- Bypass:
  - Use Kernel Return-Oriented Programming **(ROP) gadgets** to perform arbitrary operations instead of patching memory.

- **KCFG** is **enabled by default** since Win10 (req. VBS)
- Performs checks on indirect function calls to **avoid user-mode memory** be used **as Kernel code**

Hypervisor Code Integrity (HVCI)

Kernel Control Flow Guard (KCFG)

Kernel Control-Flow Enforcement Technology (KCET)

Kernel security features

- **HVCI** is **enabled by default in clean Windows 11 Enterprise** 23H2 installations
- Protected by Secure Boot (UEFI lock)
- Ensures that **only code with valid trusted signatures** can run in Kernel mode
- HVCI (VTL1) ensures that code in VTL0 can't be modified. Extended Page Tables (EPT) ensures that pages mapped as **read-execute can't be made writable** (as well as read-write not made executable). When **EPT detects a violation, it informs HVCI**, if it's in a critical Kernel memory page then HVCI would trigger a BSOD aka **DSE patching is not possible**.
- Bypass:
  - Use Kernel Return-Oriented Programming **(ROP) gadgets** to perform arbitrary operations instead of patching memory.

- **KCFG** is **enabled by default** since Win10 (req. VBS)
- Performs checks on indirect function calls to **avoid user-mode memory** be used **as Kernel code**

Hypervisor Code Integrity (HVCI)

Kernel Control Flow Guard (KCFG)

Kernel Control-Flow Enforcement Technology (KCET)

- **KCET** is a hardware-enforced shadow stack to that **detects call/return mismatch** (flow hijack)
- **Disabled by default** (req. Intel 11th-gen +)
- Kernel **ROP mitigation**

HVCI bypasses

LEVERAGING KERNEL ARBITRARY R/W TO MANIPULATE THE SSDT	LEVERAGING KERNEL ARBITRARY R/W TO HIJACK A USER-MODE THREAD
LEVERAGING LARGE PAGES	EMULATED FILESYSTEM “BUG” NTFS

https://github.com/gavz/DriverJack/blob/master/%5BWhitepaper%5D%20DriverJack%20-%20Abusing%20Emulated%20Read-Only%20Filesystems%20and%20NTFS%20Glitches%20for%20Infection%20and%20Persistence.pdf

LEVERAGING KERNEL ARBITRARY R/W TO HIJACK A USER-MODE THREAD

LPE

SaharaDriver
- Bypass Kernel security defenses

Vulnerable signed driver
- Kernel Read
- Kernel Write

(**RTCore64.sys**)

Shellcode via ROP gadgets in Kernel (KernelForge)

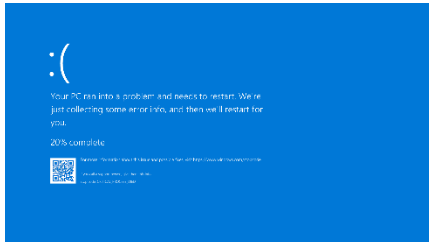
Not present in MS drivers disallowed list (last update)

No BSOD

1. Creating a **dummy thread** that waits for an event.
2. Using Kernel Read/Write primitives **to hijack the thread's execution (**return addr NtWaitForSingleObject)
3. Building and executing a **ROP chain** to call Kernel functions.
4. When the **thread resumes**, it follows the ROP chain instead of returning to its normal flow.

https://github.com/Cr4sh/KernelForge

LEVERAGING KERNEL ARBITRARY R/W TO MANIPULATE THE SSDT:



LPE + **Malk** - Bypass Kernel security defenses + Vulnerable signed driver - Kernel R/W (**procmon.sys, DBUtilDrv2.sys**) + Not present in MS drivers disallowed list (last update) != No BSOD

This method leverages **Kernel arbitrary read/write** to manipulate the **System Service Descriptor Table (SSDT)** by **remapping virtual addresses to physical memory**.

By accessing and manipulating the Page Directory Page Table (PDPT) and bypassing Kernel protections, such as **PatchGuard**, the attacker can **modify SSDT entries to point to malicious kernel functions**. This process allows unauthorized access to Kernel memory, bypassing traditional protections.

https://datafarm-cybersecurity.medium.com/code-execution-against-windows-hvci-f617570e9df0

Driver loader weaponization

	SaharaDriver Loader	Enforced by Windows
Driver Signed Enforcement (DSE)	Implemented	By default
Kernel Patch Protection (KPP) – Patch Guard	Implemented	By default
Virtualized-based Security (VBS)		
Kernel Data Protection (KDP)	Implemented	If VBS enabled
Secure Kernel Patch Guard (SKPG) – Hyper Guard	No bypass known	No
Hypervisor Code Integrity (HVCI)	Implemented	If VBS + enforced
Kernel Control Flow Guard (KCFG)	Implemented	If HVCI enabled
Kernel Control-Flow Enforcement Technology (KCET)	No bypass known	No

https://blog.xpnsec.com/gcioptions-in-a-virtualized-world/

Demo setup:

Windows 11 +  + Windows Defender + **VBS** + **HVCI**

Demo requirements:

Demo (Blaster 2)

02

MS PREINSTALLED DRIVERS N-DAY ★★☆☆☆

Is it even possible N-DAY attacks on MS preinstalled drivers with Windows Update?

Downdate exposes weaknesses in the **Windows Update** process.
The **lack of integrity verification** for older, **signed binaries**, allows the re-introduction of vulnerable versions of system components.

CVE-2024-38202 no patch available right now

Invalid Secure Kernel → VBS not able to tun → HVCI not able to RUN → UEFI Lock Bypass

https://i.blackhat.com/BH-US-24/Presentations/REVISED_US24-Leviev-Windows-Downdate-Downgrade-Attacks-Using-Windows-Updates-Wednesday.pdf

Demo setup:

Windows 11 +  + Windows Defender + **VBS** + **HVCI** + **UEFI Lock**

Demo requirements:

Demo (Downtime)

03

MS PREINSTALLED DRIVERS 0-DAY ★★★☆☆

- Lazarus Group
- CVE-2024-21338 → appid.sys AppLocker driver (IOCTL → PreviousMode)
- Low privileges required (local service)
- **FudModule** rootkit installed
  - Disrupt various Kernel security mechanisms
  - Disable EDRs
  - Disable Protected Process Light (PPL) → PPLKiller, PPLFault (Downtime)
  - Minifilter → intercept file system operations
  - Registry/Process callbacks
  - Network traffic filtering modification
  - Disruption of Event Tracing for Windows (ETW)

LAZARUS

https://decoded.avast.io/janvojtesek/lazarus-and-the-fudmodule-rootkit-beyond-byovd-with-an-admin-to-kernel-zero-day/

Detection ☆☆☆☆☆

- Mitigation
  - Enable VBS + HVCI (UEFI Locked)
  - Enable KCET (if possible)
  - Enable SKPG (if possible)

- Detection (Block)
  - Block drivers with Windows Defender Application Control (WDAC) → Additional Policies (Signed)
  - MS Attack Surface Reduction (ASR) Rule: Block abuse of exploited vulnerable drivers

- Monitoring
  - Driver's load function
  - Disabled HVCI
  - Installation of new services
  - File write drivers on disk

https://techcommunity.microsoft.com/t5/microsoft-security-experts-blog/strategies-to-monitor-and-prevent-vulnerable-driver-attacks/ba-p/4103985

Any questions?

We are **hiring**

- Detection & Monitoring
- Incident Response

Contact us

|| **@spamv** || https://www.swissre.com/careers

Thanks to

- Adam Chester (@_xpn_) → CI.dll patching to bypass KDP (CiValidateImageHeader method)
- T.Roy (Codemachine) → Awesome Kernel security trainings
- Connor McGarr (@33y0re) → Awesome blog posts and research
- Alessandro Magnosi (@klezVirus) --> Awesome paper
- Worawit (@sleepya_) → LEVERAGING KERNEL ARBITRARY R/W TO MANIPULATE THE SSDT method
- BlackSnufkin (@BlackSnufkin42) → SysMon-Killer
- Alon Leviev (@_0xDeku) → Awesome Windows Downdate research and PoC
- Dmytro Oleksiuk (@d_olex)→ KernelForge, awesome library