Project HyperEvade

Invisible Hypervisors: Stealthy Malware Analysis with HyperDbg

Björn Ruytenberg, Sina Karvandi

FOSDEM 2026 - Brussels, Belgium
January 31, 2026

FOSDEM

Who We Are

Björn Ruytenberg
@0Xiphorus@infosec.exchange

- PhD Candidate @ Vrije Universiteit Amsterdam
- Security Researcher, HyperDbg developer
- x86-64 UEFI, hypervisor and PCI Express security
- Previous work: Intel Thunderbolt vulnerability research (thunderspy.io)
- More info: bjornweb.nl

Sina Karvandi
@rayanfam@infosec.exchange

- PhD Candidate @ Vrije Universiteit Amsterdam
- System Programmer, HyperDbg developer
- Windows internals, hypervisor, digital hardware design
- Blog: rayanfam.com

VUSec

HyperDbg at FOSDEM ’26

- **Invisible Hypervisors: Stealthy Malware Analysis with HyperDbg**
  Security track, 13:00, UB5.132 (this talk)

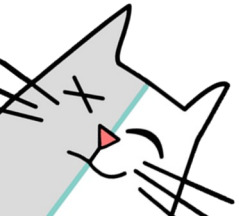
- **MBEC, SLAT, and HyperDbg: Hypervisor-Based Kernel- and User-Mode Debugging**
  Virtualization and Cloud Infrastructure track, 18:30, H.2213

01
Introduction

Introducing hypervisor-assisted debugging and transparency

HyperDbg Debugger

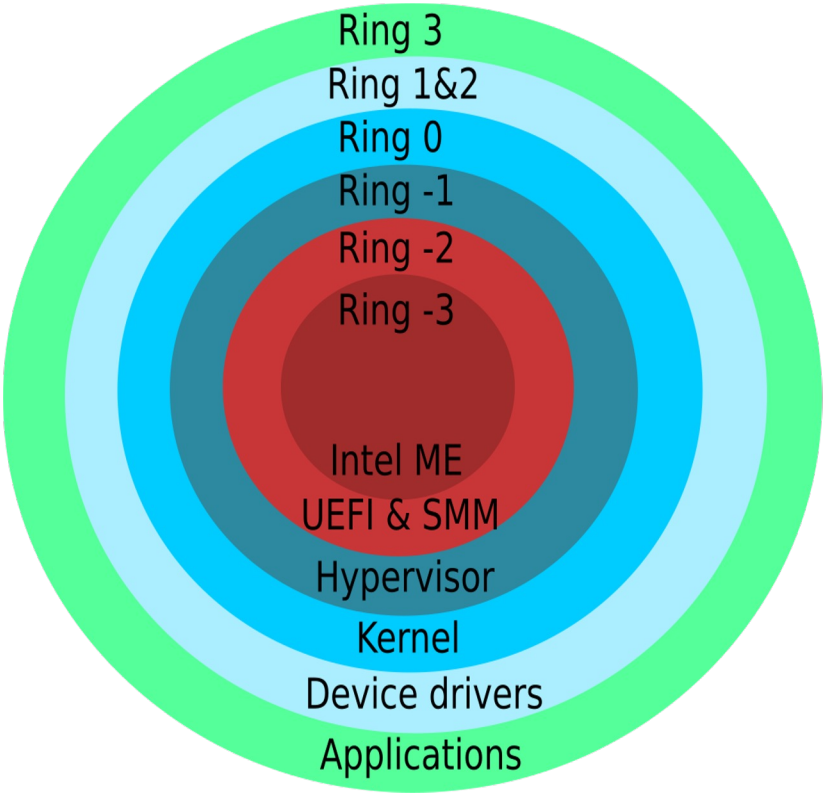
- FOSS (GPLv3) hypervisor-assisted debugger
- Leverages hardware virtualization controls to deliver advanced debugging capabilities (e.g., EPT-based memory monitoring, system call interception, PMIO/MMIO debugging)
- Operates independently of OS-level debugging APIs, providing higher transparency than traditional debuggers
- First released for Windows (2022), actively maintained since
  - UEFI-based, OS-agnostic hypervisor agent scheduled on roadmap



Get the source code:
github.com/HyperDbg/HyperDbg

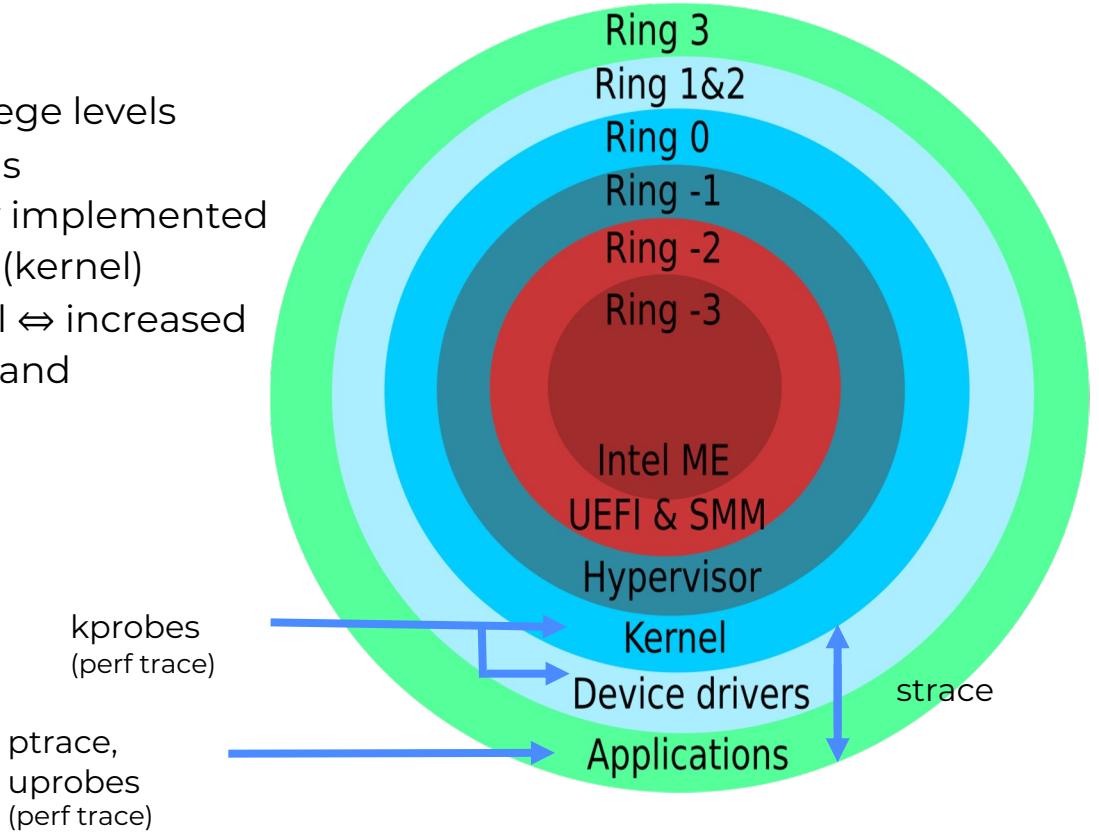
Background

- x86-64 CPUs offer privilege levels through protection rings
- Debuggers are typically implemented in ring 3 (user) or ring 0 (kernel)
- Increased privilege level ⇔ increased debugging capabilities and transparency



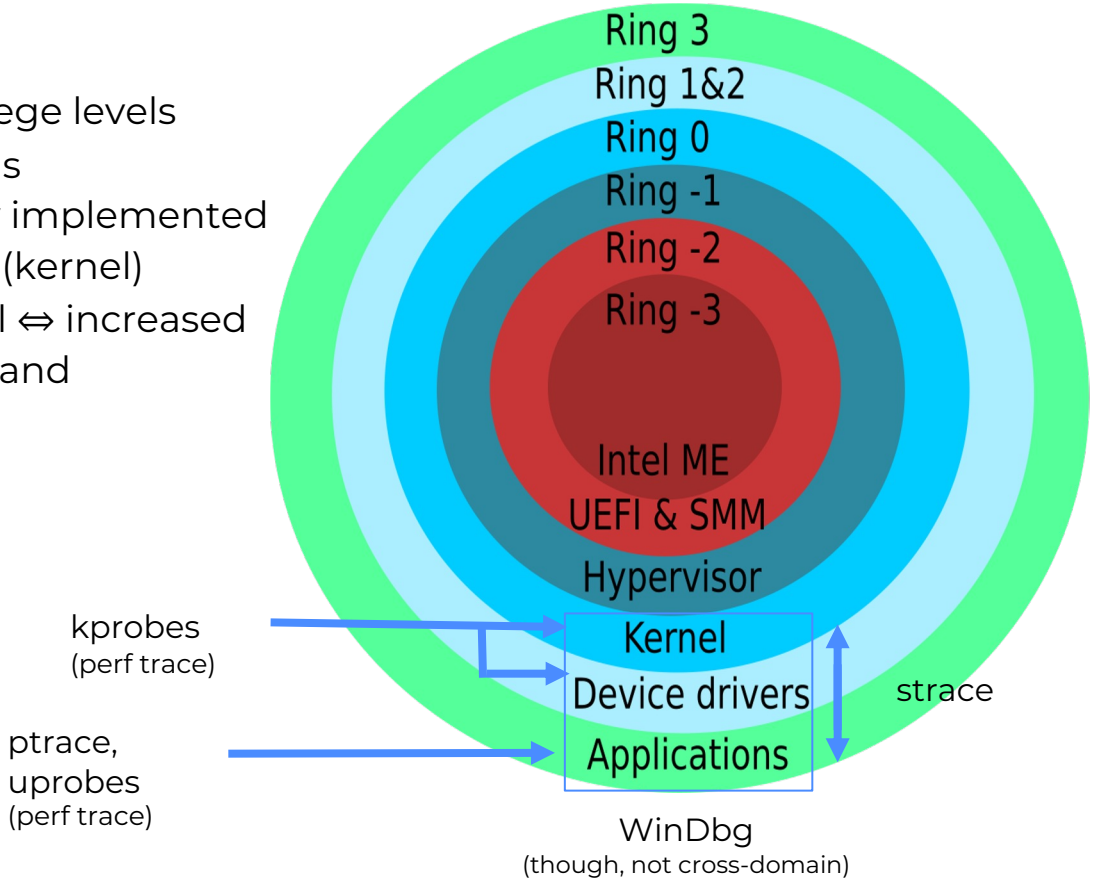
Background

- x86-64 CPUs offer privilege levels through protection rings
- Debuggers are typically implemented in ring 3 (user) or ring 0 (kernel)
- Increased privilege level ⇔ increased debugging capabilities and transparency



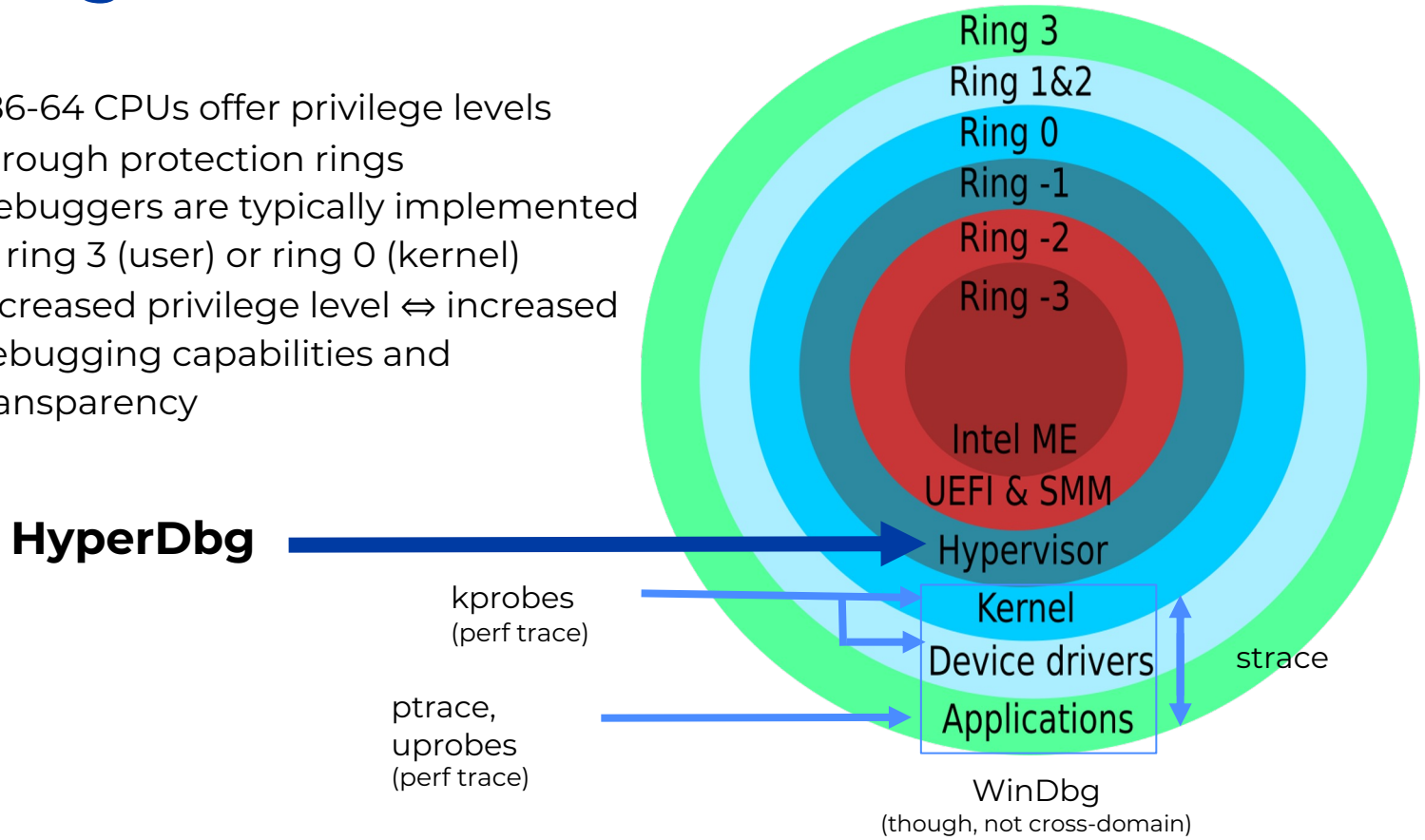
Background

- x86-64 CPUs offer privilege levels through protection rings
- Debuggers are typically implemented in ring 3 (user) or ring 0 (kernel)
- Increased privilege level ⇔ increased debugging capabilities and transparency



Background

- x86-64 CPUs offer privilege levels through protection rings
- Debuggers are typically implemented in ring 3 (user) or ring 0 (kernel)
- Increased privilege level ⇔ increased debugging capabilities and transparency



Debugging and Analyzing Malware

Anti-Debugging Techniques

Malware typically implements numerous anti-debugging and anti-hypervisor techniques

Deviating Dynamic Behavior

If malware detects the presence of a debugger, sandbox, or hypervisor, it typically conceals its internal behavior

Need for Mitigations

Bypassing these protections allows a debugger to analyze and reverse engineer the malware

02

Approach

HyperEvade’s anti-hypervisor and anti-debugging techniques

Hypervisor-Based Transparency

Roadmap (1/2)

2022

- No OS debugging APIs are used
- User mode and OS unaware about debugging environment

Present (WIP)

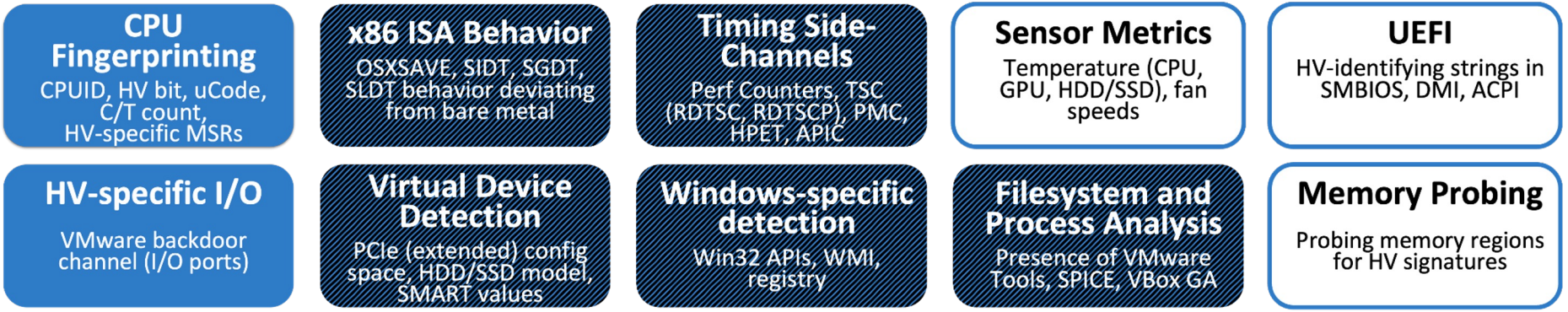
- Minimize HyperDbg artifacts (e.g. user mode process and modules, kernel mode drivers, file handles)
- Hardening hypervisor against tampering attacks (e.g. host IDT)

2027

- Reduce top-level hypervisor footprint, e.g. via HV-specific PCIe devices and drivers, processes (“guest tools”), file system and registry
- Address subset of architectural side-channels (e.g. timing, MSRs, PMCs, XSETBV, SIDT, SGDT)

Hypervisor-Based Transparency

Roadmap (2/2)



- Implemented
- Mostly finished
- To be scheduled

Hypervisor-Based Transparency

Implementation showcase: virtual PCIe devices

Virtual Device Detection

PCIe (extended) config space, HDD/SSD model, SMART values

```
HyperDbg> !pcitree
DBDF         | VID:DID   | Vendor Name       | Device Name
-------------------------------------------------------------------
0000:00:00:0 | 8086:a71b | Intel Corporation | N/A
0000:00:02:0 | 8086:a7ad | Intel Corporation | Raptor Lake-U [Intel Graphics]
0000:00:04:0 | 8086:a71d | Intel Corporation | Raptor Lake Dynamic Platform and Thermal F
0000:00:06:0 | 8086:a74d | Intel Corporation | Raptor Lake PCIe 4.0 Graphics Port
0000:00:08:0 | 8086:a74f | Intel Corporation | GNA Scoring Accelerator module
0000:00:0d:0 | 8086:a71e | Intel Corporation | Raptor Lake-P Thunderbolt 4 USB Controlle
0000:00:14:0 | 8086:51ed | Intel Corporation | Alder Lake PCH USB 3.2 xHCI Host Controll
0000:00:14:2 | 8086:51ef | Intel Corporation | Alder Lake PCH Shared SRAM
0000:00:15:0 | 8086:51e8 | Intel Corporation | Alder Lake PCH Serial IO I2C Controller #0
0000:00:15:1 | 8086:51e9 | Intel Corporation | Alder Lake PCH Serial IO I2C Controller #1
0000:00:16:0 | 8086:51e0 | Intel Corporation | Alder Lake PCH HECI Controller
0000:00:1c:0 | 8086:51bf | Intel Corporation | Alder Lake PCH-P PCI Express Root Port #9
0000:00:1f:0 | 8086:519d | Intel Corporation | Raptor Lake LPC/eSPI Controller
0000:00:1f:3 | 8086:51ca | Intel Corporation | Raptor Lake-P/U/H cAVS
0000:00:1f:4 | 8086:51a3 | Intel Corporation | Alder Lake PCH-P SMBus Host Controller
0000:00:1f:5 | 8086:51a4 | Intel Corporation | Alder Lake-P PCH SPI Controller
0000:01:00:0 | 1e0f:000c | KIOXIA Corporation | NVMe SSD Controller BG5 (DRAM-less)
0000:02:00:0 | 10ec:b852 | Realtek Semiconductor Co., Ltd. | RTL8852BE PCIe 802.11ax Wire
```

Hypervisor-Based Transparency

Implementation showcase: virtual PCIe devices

Virtual Device Detection

PCIe (extended) config space, HDD/SSD model, SMART values

```
0: kHyperDbg> !pcitree
DBDF         | VID:DID   | Vendor Name       | Device Name
-------------------------------------------------------------------------
0000:00:00:0 | 8086:7190 | Intel Corporation | 440BX/ZX/DX - 82443BX/ZX/DX Host bridge
0000:00:01:0 | 8086:7191 | Intel Corporation | 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge
0000:00:07:0 | 8086:7110 | Intel Corporation | 82371AB/EB/MB PIIX4 ISA
0000:00:07:1 | 8086:7111 | Intel Corporation | 82371AB/EB/MB PIIX4 IDE
0000:00:07:3 | 8086:7113 | Intel Corporation | 82371AB/EB/MB PIIX4 ACPI
0000:00:07:7 | 15ad:0740 | VMware            | Virtual Machine Communication Interface
0000:00:0f:0 | 15ad:0405 | VMware            | SVGA II Adapter
0000:00:11:0 | 15ad:0790 | VMware            | PCI bridge
0000:00:15:1 | 15ad:07a0 | VMware            | PCI Express Root Port
...
0000:00:18:7 | 15ad:07a0 | VMware            | PCI Express Root Port
0000:02:00:0 | 15ad:0774 | VMware            | USB1.1 UHCI Controller
0000:02:01:0 | 15ad:1977 | VMware            | HD Audio Controller
0000:02:02:0 | 15ad:0770 | VMware            | USB2 EHCI Controller
0000:02:03:0 | 15ad:07e0 | VMware            | SATA AHCI controller
0000:03:00:0 | 8086:10d3 | Intel Corporation | 82574L Gigabit Network Connection
0000:0b:00:0 | 15ad:077a | VMware            | N/A
0000:13:00:0 | 15ad:07f0 | VMware            | NVMe SSD Controller
```

Hypervisor-Based Transparency

Implementation showcase: virtual PCIe devices

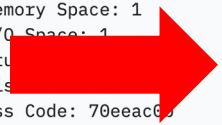

```
6: kHyperDbg> !pcicam 3 0 0
PCI configuration space (CAM) for device 0000:03:00:0

Common Header:
VID:DID: 8086:10d3
Vendor Name: Intel Corporation
Device Name: 82574L Gigabit Network Connection
Command: 0007
  Memory Space: 1
  I/O Space: 1
Status: 0010
Revision ID: 00
Class Code: 70eeac0b
CacheLineSize: 10
PrimaryLatencyTimer: 00
HeaderType: Endpoint (00)
  Multi-function Device: False
Bist: 00

Device Header:
BAR0
 BAR Type: MMIO
 BAR: fea00000
 BAR (actual): fea00000
 Prefetchable: False
 Addressable range: 0-00000000
BAR1
```

Hypervisor-Based Transparency

Implementation showcase: virtual PCIe devices

Virtual Device Detection

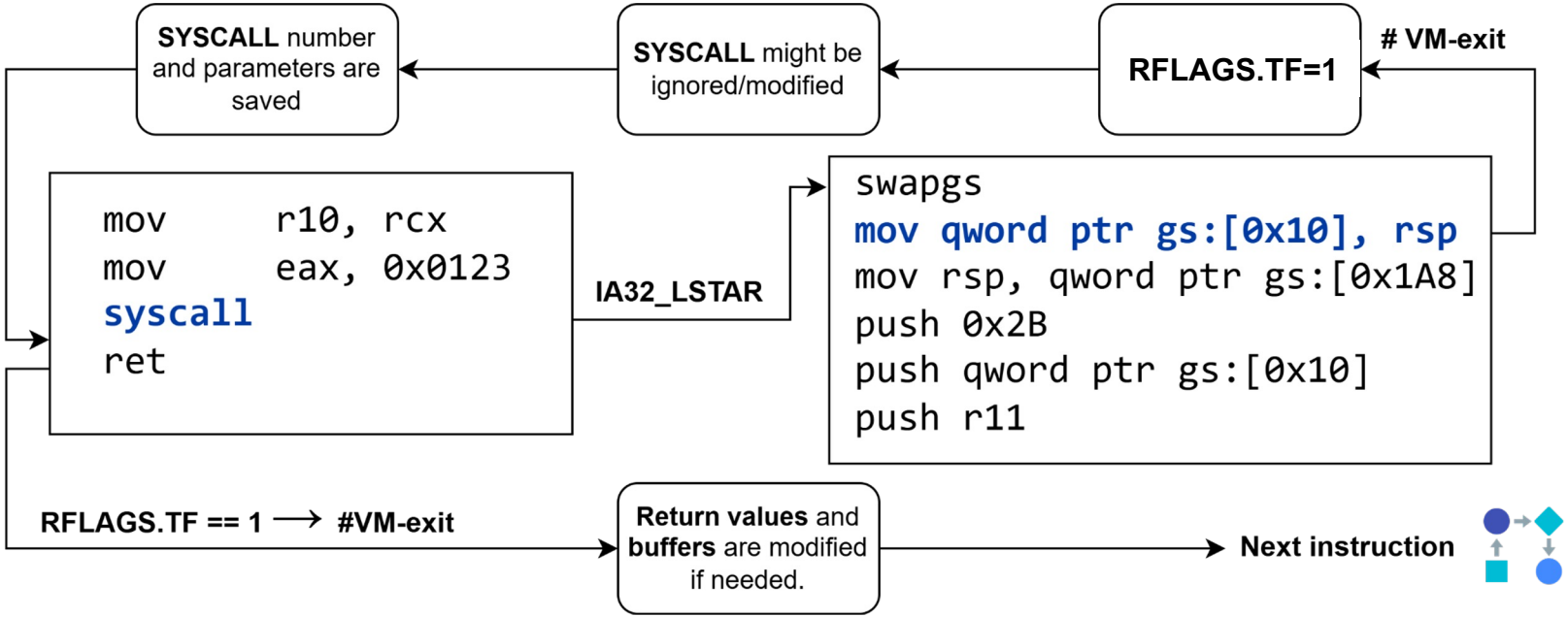
PCIe (extended) config space, HDD/SSD model, SMART values



```
6: kHyperDbg> !pcicam 3 0 0
PCI configuration space (CAM) for device 0000:03:00:0

Common Header:
VID:DID: 8086:1521
Vendor Name: Intel Corporation
Device Name: Ethernet Server Adapter I350-T2V2
Command: 0007
  Memory Space: 1
  I/O Space: 1
Status: 0010
Revision ID: 00
Class Code: 70eeac0b
CacheLineSize: 10
PrimaryLatencyTimer: 00
HeaderType: Endpoint (00)
  Multi-function Device: False
Bist: 00

Device Header:
BAR0
 BAR Type: MMIO
 BAR: fea00000
 BAR (actual): fea00000
 Prefetchable: False
 Addressable range: 0-00000000
BAR1
```

Hypervisor-Based Transparency

Implementation showcase: syscall hooking



Hypervisor-Based Transparency

Side track: Windows debugging crash course

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  ...
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  ...
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
   ...
  ULONG                         SessionId;
} PEB, *PPEB;
```

Source: https://learn.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb

Hypervisor-Based Transparency

Side track: Windows debugging crash course

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  ...
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  ...
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
   ...
  ULONG                         SessionId;
} PEB, *PPEB;
```

Signals debugger presence

Source: https://learn.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb

Hypervisor-Based Transparency

Side track: Windows debugging crash course

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  ...
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  ...
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
   ...
  ULONG                         SessionId;
} PEB, *PPEB;
```

Enumerates PE-loaded modules (malware hides injected modules)

Source: https://learn.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb

Hypervisor-Based Transparency

Side track: Windows debugging crash course

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  ...
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  Pro
  PVOID                         Rese
  PVOID                         Atl
  ...
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
   ...
  ULONG                         SessionId;
} PEB, *PPEB;
```

Undocumented NtGlobalFlag (offsets `0x68` , `0xbc`) reveals debugger presence

Source: https://learn.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb

Hypervisor-Based Transparency

Side track: Windows debugging crash course

... and what about **TEB** (**T**hread **E**nvironment **B**lock), and all the other fields?

Hardware Debug Registers are not enough for monitoring them all - x86 limits us to four breakpoint registers

Hypervisor-Based Transparency

Side track: Windows debugging crash course

... and what about **TEB** (**T**hread **E**nvironment **B**lock), and all the other fields?

Hardware Debug Registers are not enough for monitoring them all - x86 limits us to four breakpoint registers

“EPT Monitor Hooks” to the Rescue!

Hypervisor-Based Transparency

Implementation showcase: Win32 API / PE struct monitoring

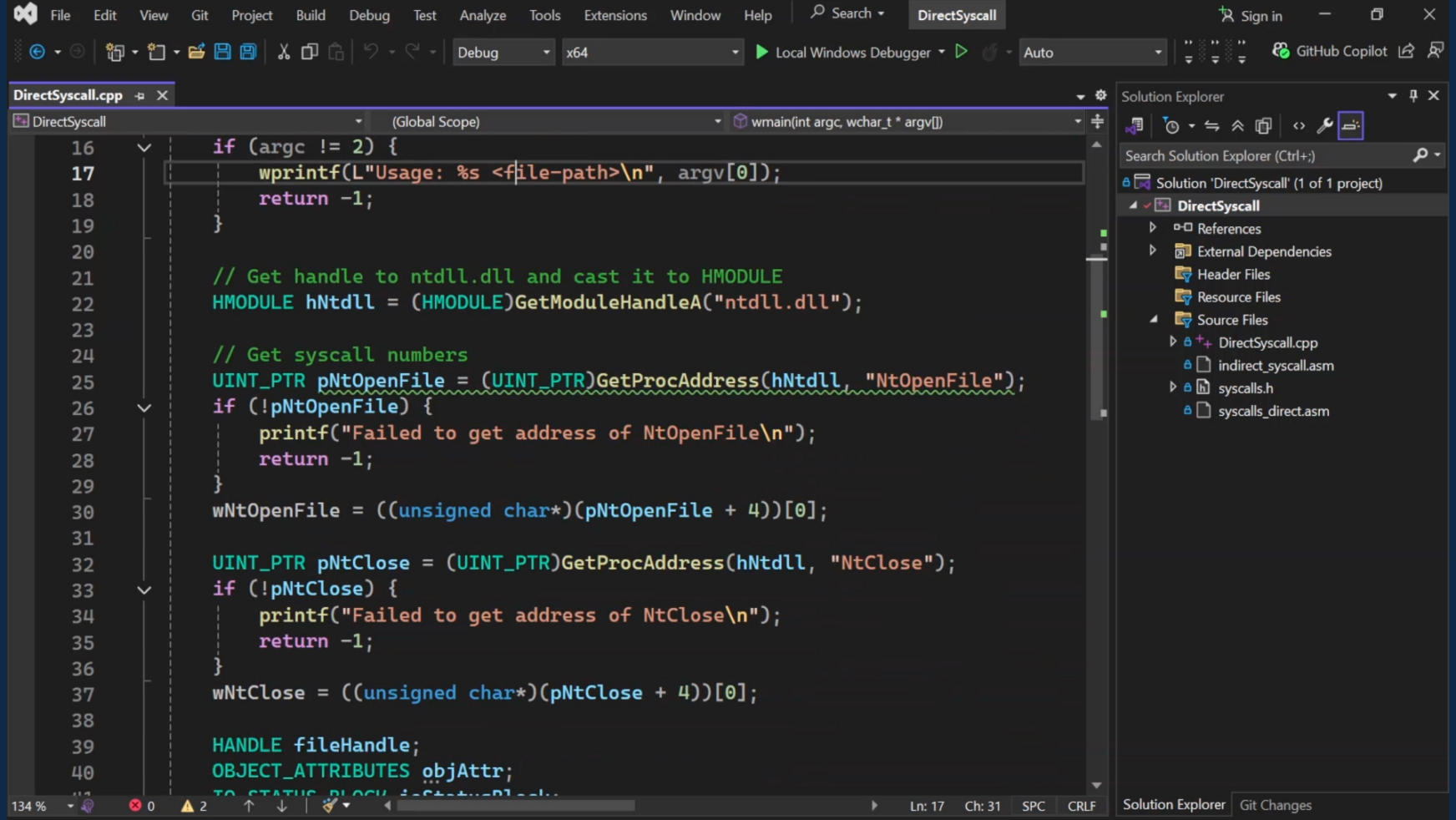
Runtime Field / Structure	Description	Typical Use
`PEB.BeingDebugged`	Flag set if debugger is present	Direct debugger detection
`PEB.NtGlobalFlag`	Contains special flags when debugged	Heap validation flags
`HeapFlags` in ProcessHeap	Indicates debugging heap	Detected via PEB traversal
`IMAGE_DEBUG_DIRECTORY`	Debug info in PE header	Used to detect debug builds
`IMAGE_TLS_DIRECTORY`	TLS callback execution	Pre-main debugger evasion
`NtQueryInformationProcess`	Queries debug port or flags	Kernel-level detection

HyperEvade is capable of intercepting any user and kernel mode attempts to access these fields

03

Demo

Transparent hypervisor-assisted debugging in action


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
DirectSyscall
Sign in
Debug
x64
Local Windows Debugger
Auto
GitHub Copilot
DirectSyscall.cpp
DirectSyscall
(Global Scope)
wmain(int argc, wchar_t * argv[])
16 if (argc != 2) {
17 wprintf(L"Usage: %s <file-path>\n", argv[0]);
18 return -1;
19 }
20
21 // Get handle to ntdll.dll and cast it to HMODULE
22 HMODULE hNtdll = (HMODULE)GetModuleHandleA("ntdll.dll");
23
24 // Get syscall numbers
25 UINT_PTR pNtOpenFile = (UINT_PTR)GetProcAddress(hNtdll, "NtOpenFile");
26 if (!pNtOpenFile) {
27 printf("Failed to get address of NtOpenFile\n");
28 return -1;
29 }
30 wNtOpenFile = ((unsigned char*)(pNtOpenFile + 4))[0];
31
32 UINT_PTR pNtClose = (UINT_PTR)GetProcAddress(hNtdll, "NtClose");
33 if (!pNtClose) {
34 printf("Failed to get address of NtClose\n");
35 return -1;
36 }
37 wNtClose = ((unsigned char*)(pNtClose + 4))[0];
38
39 HANDLE fileHandle;
40 OBJECT_ATTRIBUTES objAttr;
134 %
0
2
Ln: 17
Ch: 31
SPC
CRLF
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'DirectSyscall' (1 of 1 project)
DirectSyscall
References
External Dependencies
Header Files
Resource Files
Source Files
DirectSyscall.cpp
indirect_syscall.asm
syscalls.h
syscalls_direct.asm
Solution Explorer
Git Changes
Ready
0 / 0
41
main
DirectSyscall-Example
Search web &...
ENG
10:46 AM
7/2/2025

Conclusion

- Although 100% transparency is not yet feasible, HyperEvade significantly raises the bar for transparent debugging
- HyperEvade extends HyperDbg to provide system-wide visibility and transparency
- As malware techniques evolve, new countermeasures will be required to address emerging threats
- HyperEvade is FOSS (GPLv3), under active development, and available for the community to contribute to and enhance

Thanks

Björn Ruytenberg

@0Xiphorus@infosec.exchange
https://bjornweb.nl

Mohammad Sina Karvandi

@rayanfam@infosec.exchange
https://rayanfam.com

Get the source code:
github.com/HyperDbg/HyperDbg

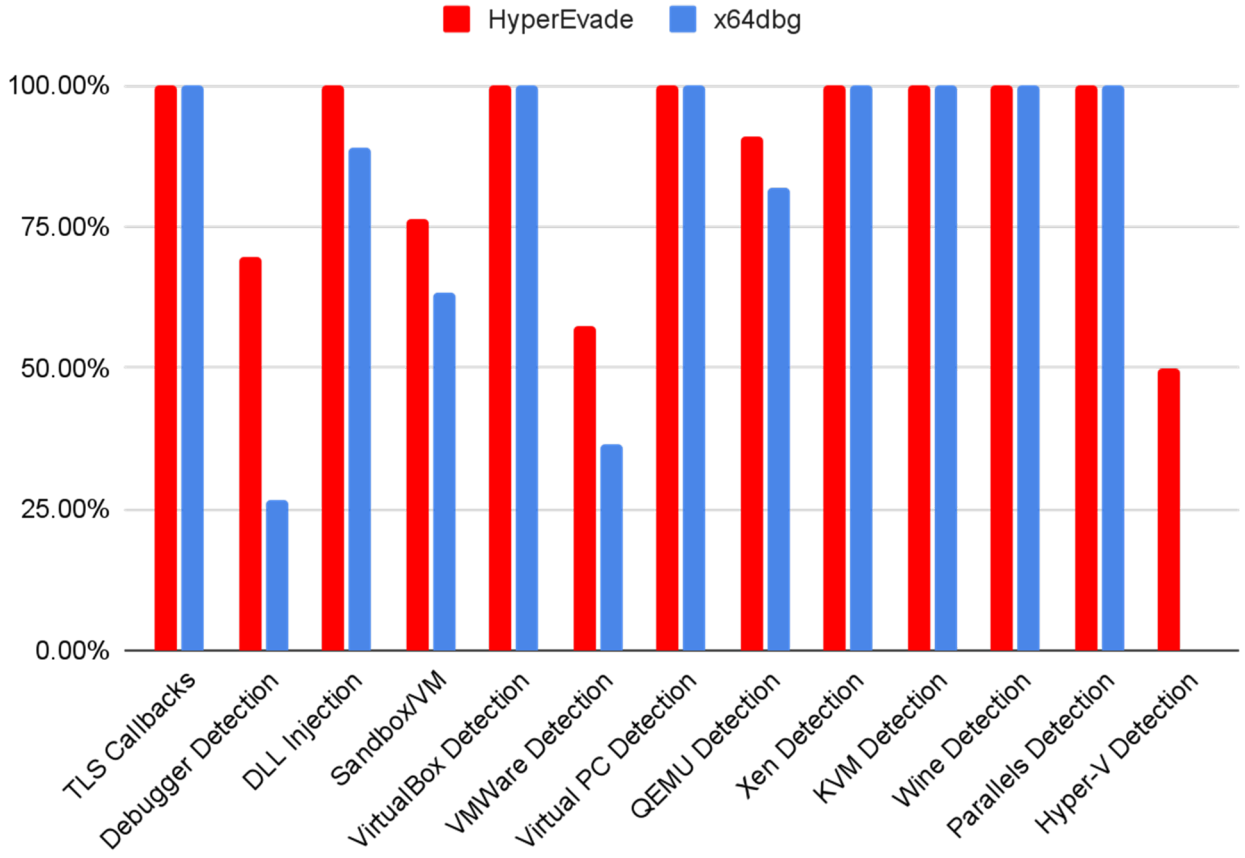
Additional Slides

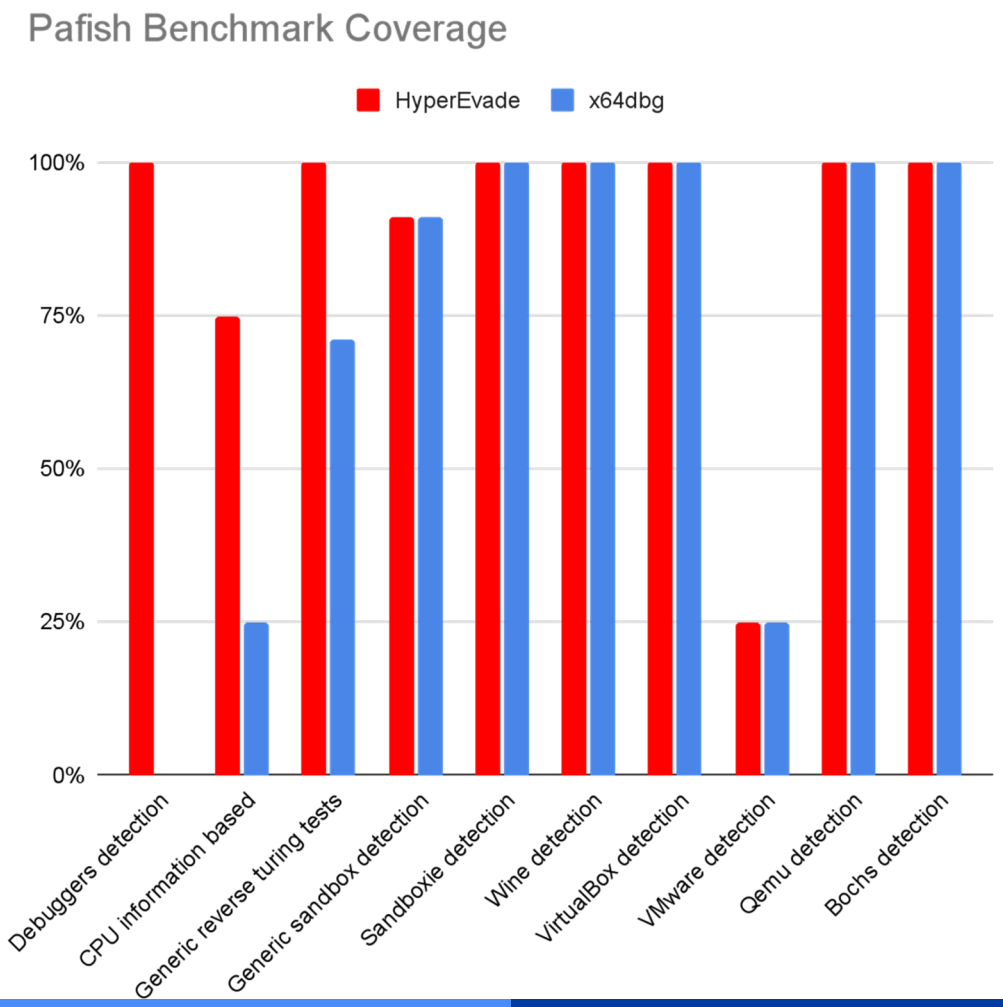
04

Evaluation

Comparing HyperEvade with state of the art

Al-Khaser Benchmark Coverage




Pafish Benchmark Coverage
HyperEvade
x64dbg
100%
75%
50%
25%
0%
Debuggers detection
CPU information based
Generic reverse turing tests
Generic sandbox detection
Sandboxie detection
Wine detection
VirtualBox detection
VMware detection
Qemu detection
Bochs detection

Hypervisor-Based Transparency

Implementation showcase: Kernel struct monitoring

Structure / Field	Description	Check
`EPROCESS->DebugPort`	Non-null when a debugger is attached	Detect debugger on any process
`KdDebuggerEnabled` / `KdDebuggerNotPresent`	Global kernel flags	Detect kernel debugging
IDT Table	Hooks to interrupts	Look for handlers outside kernel
DR7 (Debug Register)	HW breakpoints	Check if debugger set one
CR4	VMX/Debug trap flag	Detect hypervisor presence
PsLoadedModuleList	Loaded drivers	Detect debugger-related modules
`DbgPrint` Hook	Output redirection	Check if hooked by tools

Challenges in Malware Analysis