Project HyperEvade

Countering Anti-Debugging Techniques: Enhancing Transparency in Nested Virtualization using HyperDbg

Björn Ruytenberg, Mohammad Sina Karvandi

DEBT 2025 – Third Workshop on Future Debugging Techniques
July 4, 2025

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), sandbox escapes (major web browsers, Microsoft Office, Adobe)
- More info: bjornweb.nl

Mohammad Sina Karvandi
@rayanfam@infosec.exchange

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

VUSec

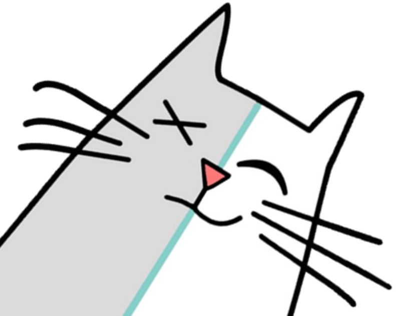
01

Introduction

Introducing hypervisor-assisted debugging and transparency

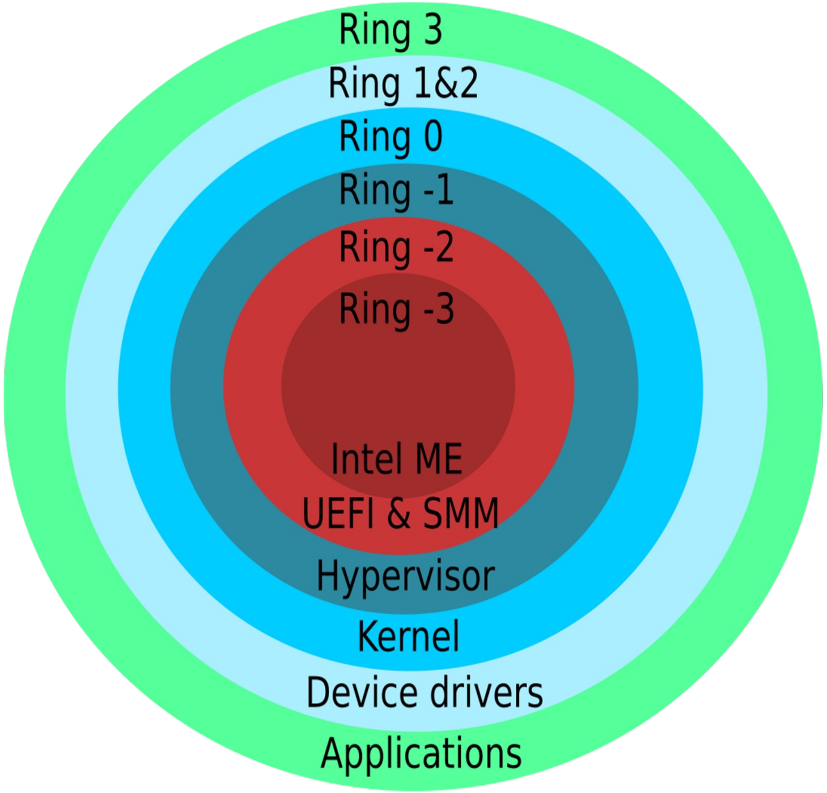
HyperDbg Debugger

- Open source (GPLv3) hypervisor-assisted debugger
- Uses hypervisor controls to provide advanced debugging features (e.g., EPT and memory monitoring hooks, system call hooks, PMIO and MMIO debugging, etc.)
- Does not rely on OS-level APIs for debugging, hence offers greater transparency than traditional debuggers
- Launched and actively maintained since 2022 (first release)



Background

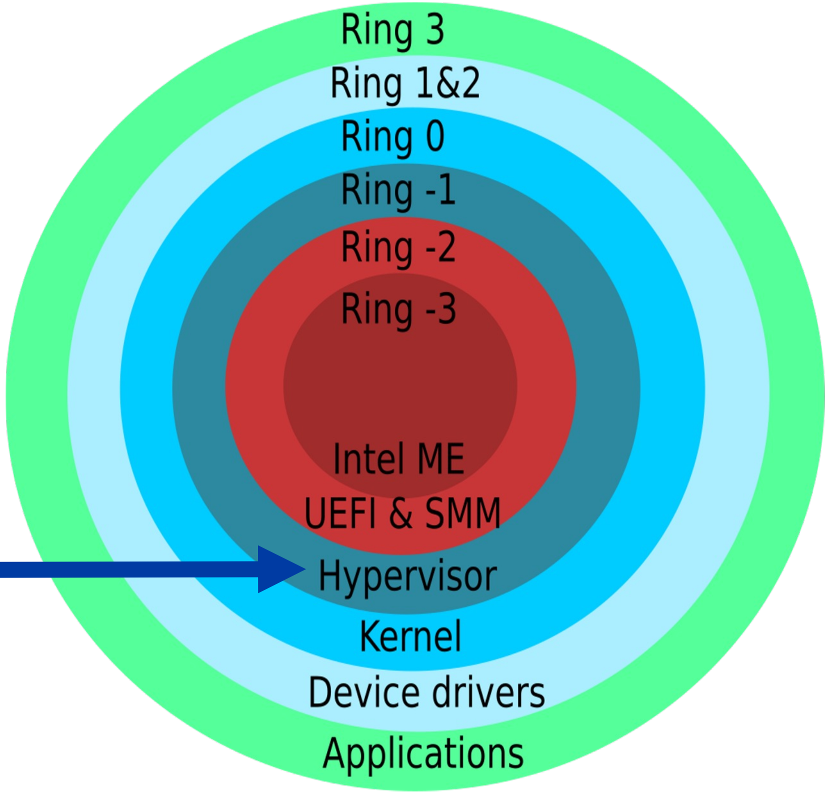
- Intel processors offer different protection rings.
- Debuggers are typically implement in ring 3 (user debuggers) or ring 0 (kernel debuggers).
- The more privileged you become, the more you are able to be transparent.



Background

- Intel processors offer different protection rings.
- Debuggers are typically implement in ring 3 (user debuggers) or ring 0 (kernel debuggers).
- The more privileged you become, the more you are able to be transparent.

HyperDbg



Debugging and Analyzing Malware

Anti-Debugging Techniques

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

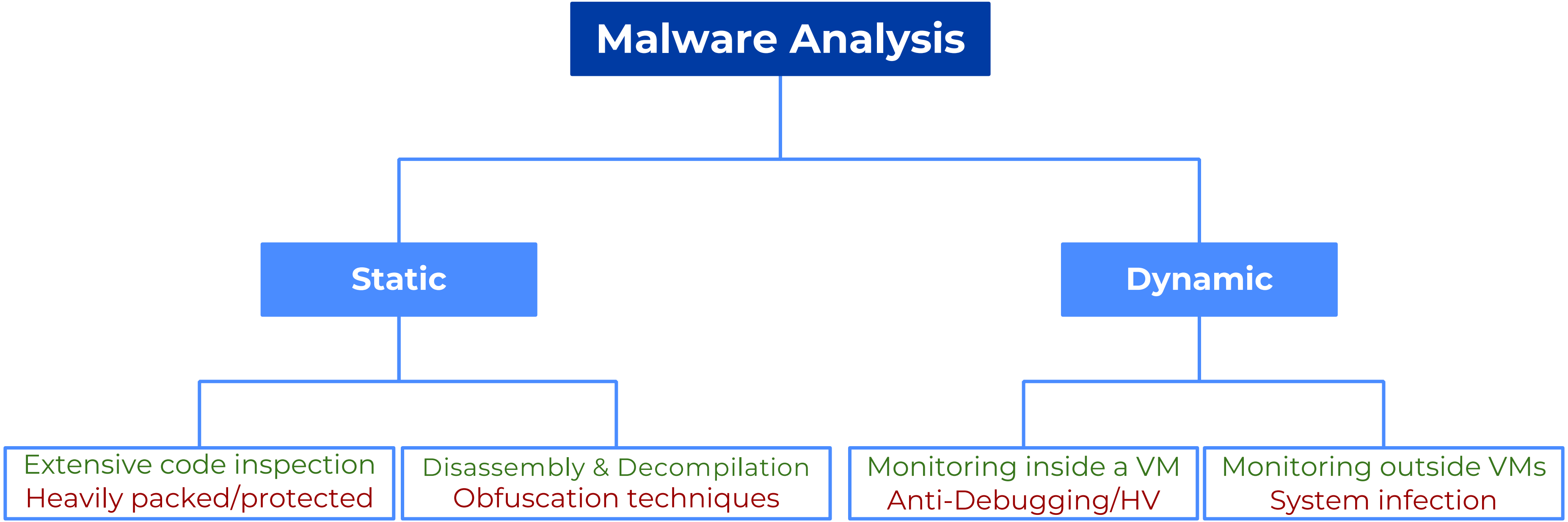
Deviating Dynamic Behavior

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

Challenges in Malware Analysis



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)

2026

- 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)

CPU Fingerprinting
CPUID, HV bit, uCode, C/T count, HV-specific MSRs

x86 ISA Behavior
OSXSAVE, SIDT, SGDT, SLDT behavior deviating from bare metal

Timing Side-Channels
Perf Counters, TSC (RDTSC, RDTSCP), PMC, HPET, APIC

Sensor Metrics
Temperature (CPU, GPU, HDD/SSD), fan speeds

UEFI
HV-identifying strings in SMBIOS, DMI, ACPI

HV-specific I/O
VMware backdoor channel (I/O ports)

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

Windows-specific detection
Win32 APIs, WMI, registry

Filesystem and Process Analysis
Presence of VMware Tools, SPICE, VBox GA

Memory Probing
Probing memory regions for HV signatures

- Implemented
- Mostly done
- 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 Controller
0000:00:14:0 | 8086:51ed | Intel Corporation | Alder Lake PCH USB 3.2 xHCI Host Controlle
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

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: 8086:10d3
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

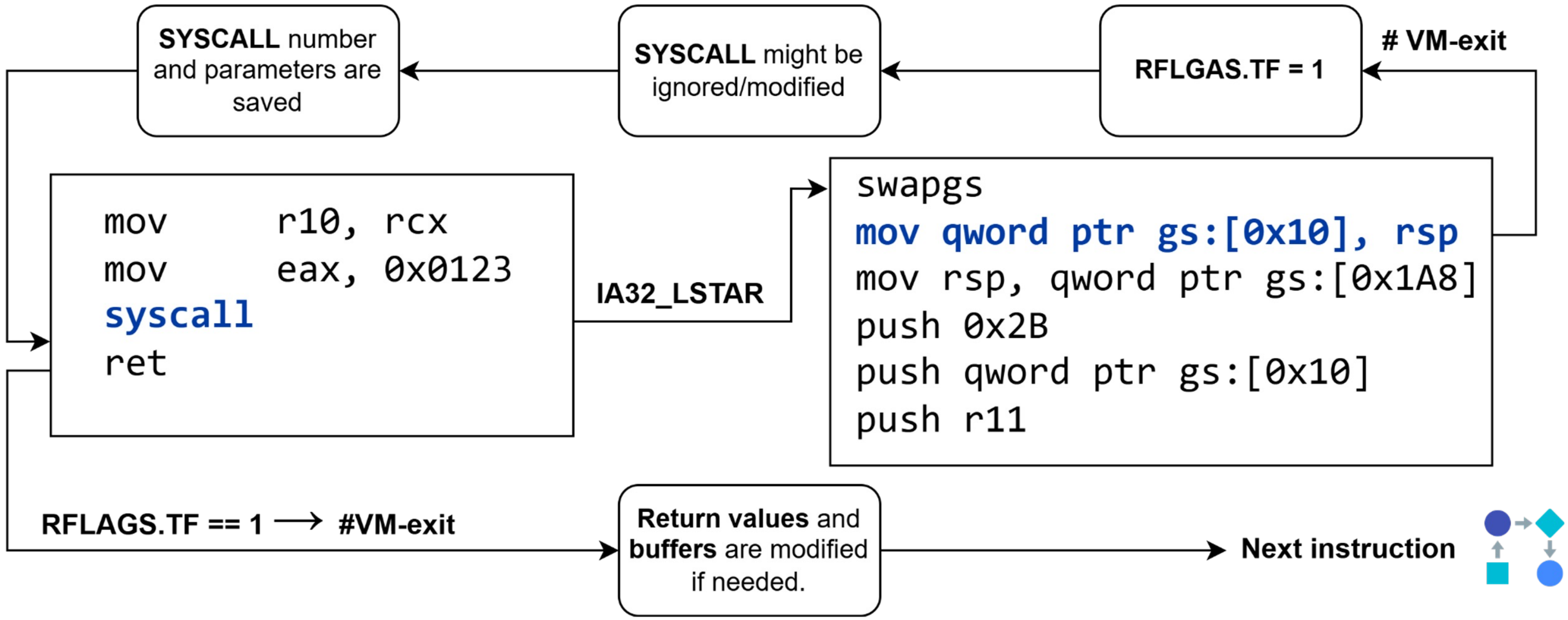
PCIe (extended) config space, HDD/SSD model, SMART values

```
6: kHyperDbg> !pcicam 3 0 0
PCI configuration space (CA

Common Header:
VID:DID: 8086:10d3
Vendor Name: Intel Corporat
Device Name: 82574L Gigabit
Command: 0007
  Memory Space: 1
  I/O Space: 1
Statu
Revis
Class Code: 70eeac0
CacheLineSize: 10
PrimaryLatencyTimer: 00
HeaderType: Endpoint (00)
  Multi-function Device: Fa
Bist: 00

Device Header:
BAR0
 BAR Type: MMIO
 BAR: fea00000
 BAR (actual): fea00000
 Prefetchable: False
 Addressable range: 0-00000
BAR1
```

```
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;
```

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;
```

Checking the presence of the debugger

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;
```

Enumerating PE loaded modules (malware hide injected modules)

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  Proc
  PVOID                         Rese
  PVOID                         AtlT
  ...
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
   ...
  ULONG                         SessionId;
} PEB, *PPEB;
```

Undocumented NtGlobalFlag at offset 0x68 or 0xbc shows the presence of debugger

Hypervisor-Based Transparency

Side track: Windows debugging crash course

… and there is also a **TEB** (**T**hread **E**nvironment **B**lock), and even more fields!

Hardware Debug Registers are not enough for monitoring them all, there are only **four** of them on each CPU.

Hypervisor-Based Transparency

Side track: Windows debugging crash course

… and there is also a **TEB** (**T**hread **E**nvironment **B**lock), and even more fields!

Hardware Debug Registers are not enough for monitoring them all, there are only **four** of them on each CPU.

EPT Monitoring 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

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

03

Demo

Transparent hypervisor-assisted debugging in action

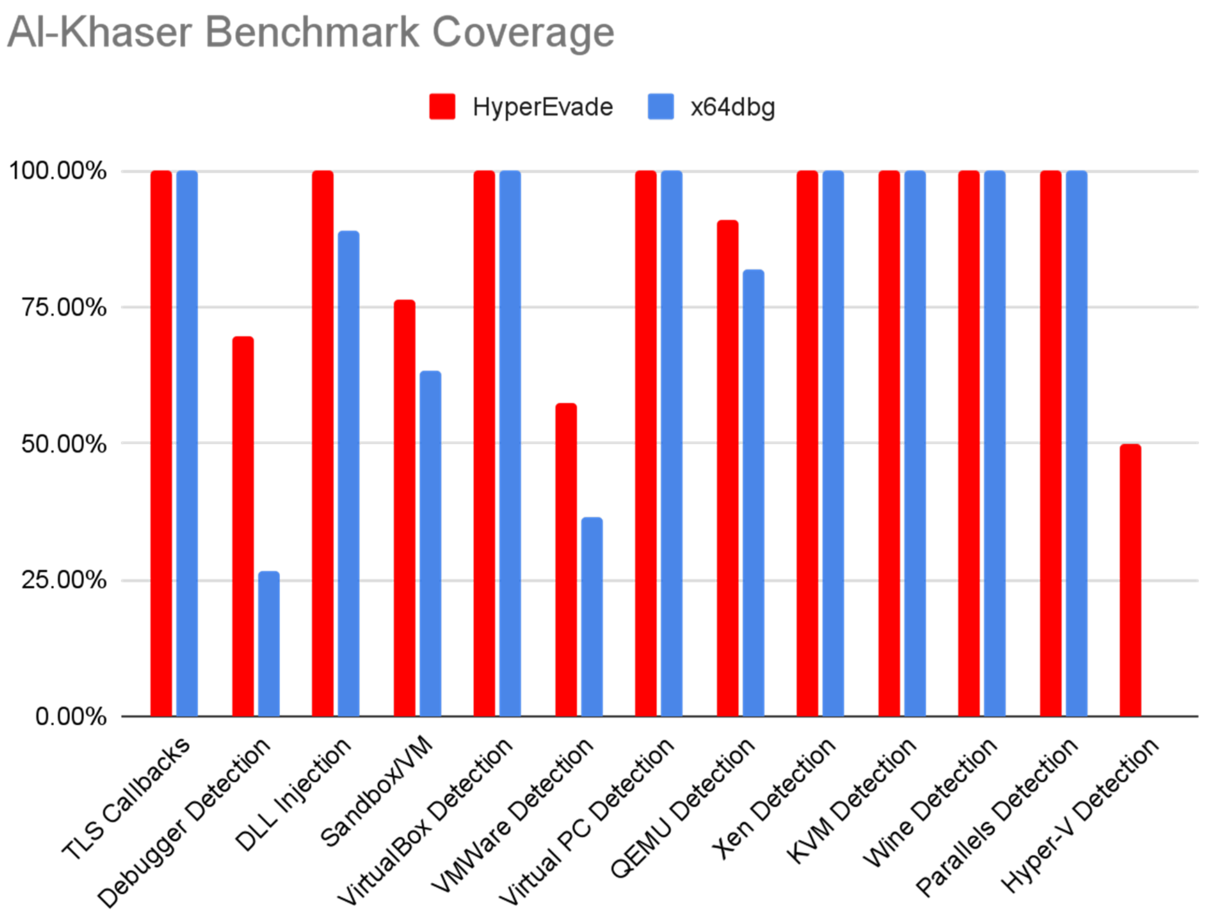
DirectSyscall.cpp

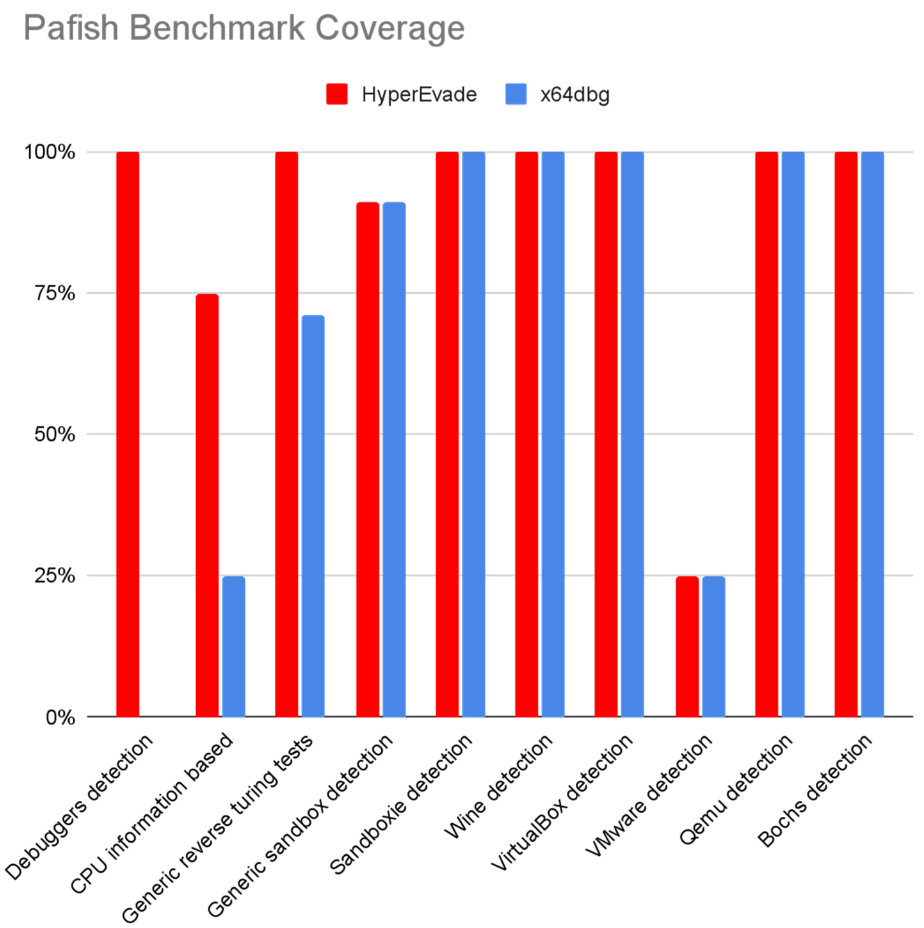
```cpp
#include "syscalls.h"

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

// Define the syscall numbers
DWORD wNtOpenFile;
DWORD wNtClose;

// Declare the RtlInitUnicodeString function
extern "C" void RtlInitUnicodeString(PUNICODE_STRING DestinationString, PCWSTR

int wmain(int argc, wchar_t* argv[]) {
    if (argc != 2) {
        wprintf(L"Usage: %s <file-path>\n", argv[0]);
        return -1;
    }

    // Get handle to ntdll.dll and cast it to HMODULE
    HMODULE hNtdll = (HMODULE)GetModuleHandleA("ntdll.dll");

    // Get syscall numbers
    UINT_PTR pNtOpenFile = (UINT_PTR)GetProcAddress(hNtdll, "NtOpenFile");
    if (!pNtOpenFile) {
        printf("Failed to get address of NtOpenFile\n");
        return -1;
```

04

Evaluation

Comparing HyperEvade with state of the art


Al-Khaser Benchmark Coverage
HyperEvade
x64dbg
100.00%
75.00%
50.00%
25.00%
0.00%
TLS Callbacks
Debugger Detection
DLL Injection
Sandbox/VM
VirtualBox Detection
VMWare Detection
Virtual PC Detection
QEMU Detection
Xen Detection
KVM Detection
Wine Detection
Parallels Detection
Hyper-V Detection


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

Conclusion

- Although 100% transparency guarantee is not yet feasible, **HyperEvade** significantly raises the bar for transparent debugging
- With the HyperEvade extension, HyperDbg provides an ideal platform for countering anti-debugging techniques due to its system-wide visibility
- HyperEvade is **open source**, under active development, and available for the community to contribute to and enhance
- As malware techniques evolve, new countermeasures will be required to address emerging threats

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