TLP:CLEAR

Evading in Plain Sight: How Adversaries Beat User-Mode Protection Engines

Omri Misgav (Independent, Israel)

About Me

Omri Misgav

- Independent Security Researcher
- Previously Head of FortiGuard Research IL @ Fortinet
- Reverse engineering, OS internals and malware research
- Past speaker at DEFCON, AVAR, BSidesLV and others

P.S. – know a cool place for bungy jumping? Feel free to share :)

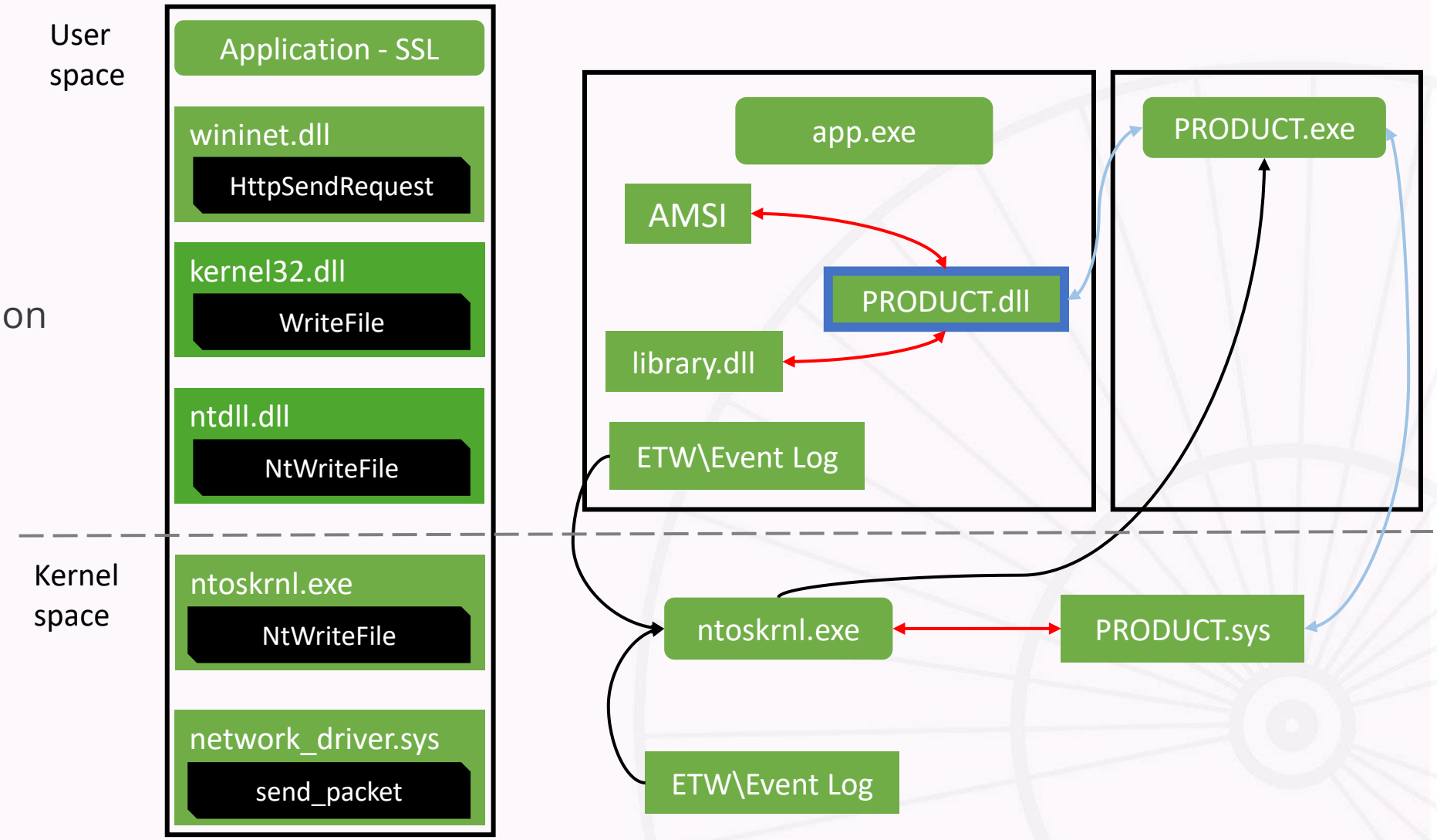
in/omri-misgav

- Introduction
- Hook Evasion tactic
- Argument Forgery tactic
- Engine Disarming tactic
- Conclusions

* No AI was used while preparing this session ☺

Intro

- User-mode monitoring
  - Instrumentation
  - Hooking
- Why?
  - Simple, stable
  - Lack of Patch Protection
  - Full context



Intro

Inline hooks

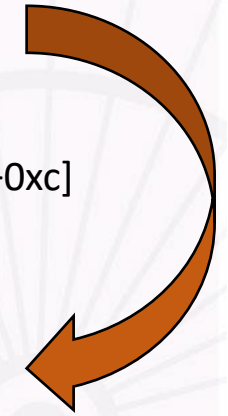
- Modify the code of the target function in-memory
- Usually with control flow instructions
  - jmp \ call \ push + ret

- Most common hooking method in security products
  - Average of 34 ntdll.dll functions by 11 endpoint vendors*
  - Cuckoo\CAPE
  - Frida

```
function_A:
0x801000:  55              push ebp
0x801001:  89 e5           mov  ebp, esp
0x801003:  83 ec 40        sub  esp, 0x40
0x801006:  50              push eax
0x801007:  8b 44 24 0c     mov  eax, [esp+0xc]
0x80100a:  …
```

```
function_A:
0x801000:  e9 95 09 00 00  jmp  hook_A
0x801005:  90              nop
0x801006:  50              push eax
0x801007:  8b 44 24 0c     mov  eax, [esp+0xc]
0x80100a:  …

hook_A:
0x802000:  55              push ebp
0x802001:  89 e5           mov  ebp, esp
0x802003:  83 ec 40        sub  esp, 0x40
0x802006:  e9 fc ef ff ff  jmp  function_A+0x6
```



Intro

- Bypasses and evasions exist for a very long time
- Increasing number of reports since 2020 (both malware and red teamers)
- MITRE ATT&CK T1562.001 (Impair Defenses: Disable or Modify Tools) mentions unhooking (i.e. “Binary Restoration” or “Engine Disarming”) only from v10 (October 2021)

- ✓ Introduction
- ❑ Hook Evasion tactic
- ❑ Argument Forgery tactic
- ❑ Engine Disarming tactic
- ❑ Conclusions

Hook Evasion Tactic

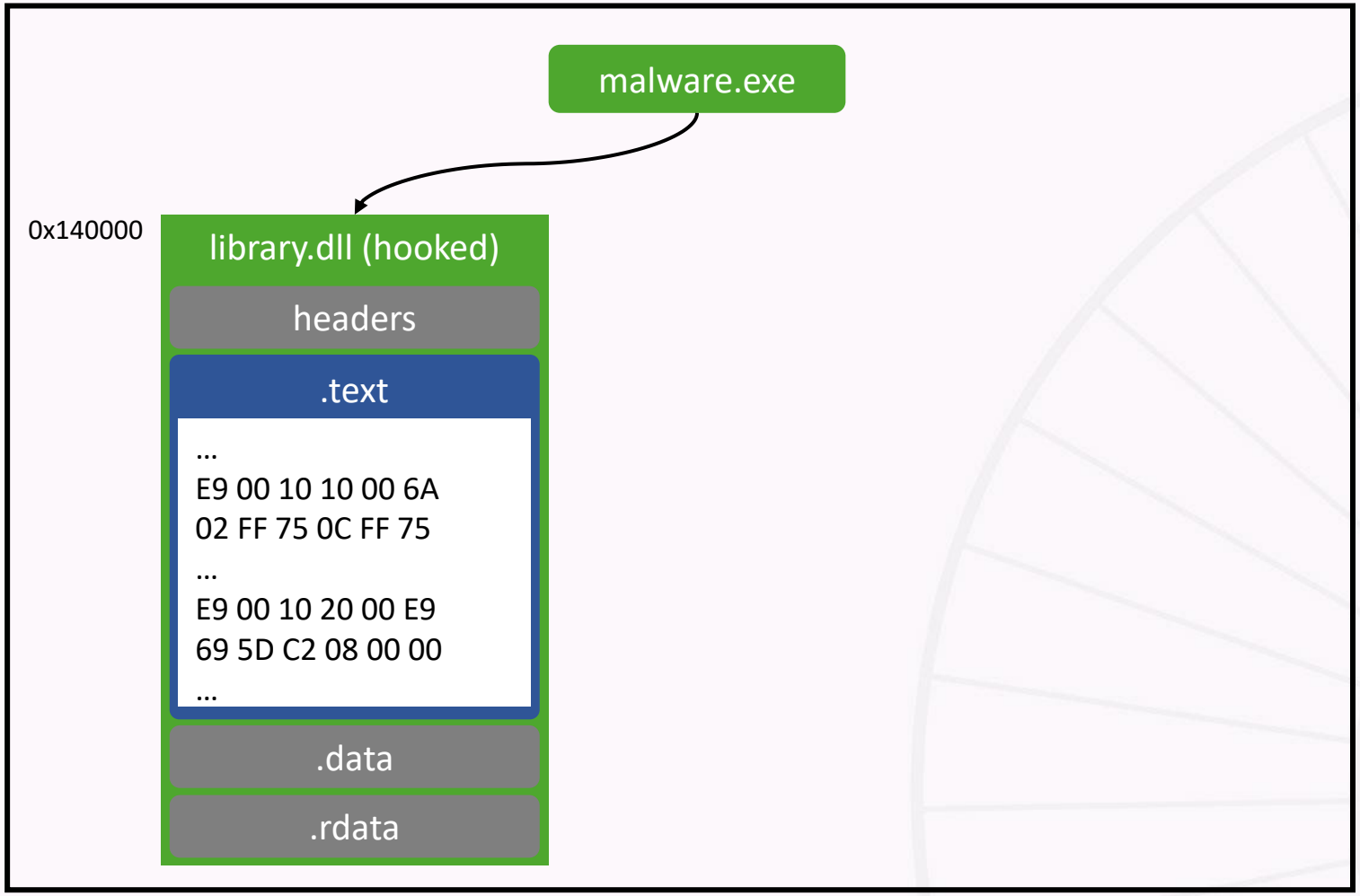
Overview

- How can it be accomplished?
  a) Execute the original instructions
  b) Execute the rest of the function

- Classes of techniques
  1. Secondary DLL mapping
  2. Binary restoration
  3. Direct system call invocation
  4. Code splicing

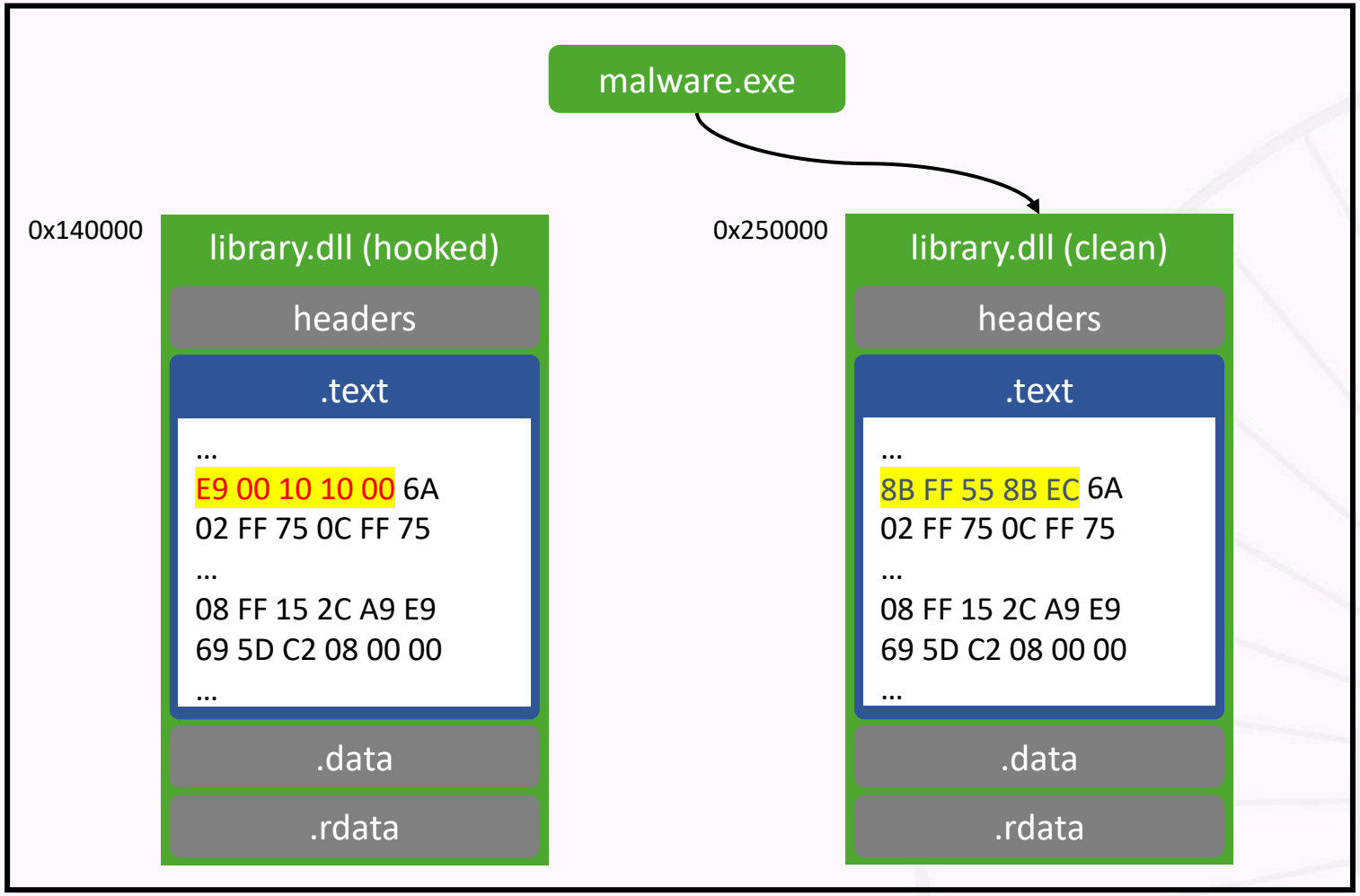
Secondary DLL Mapping

Process Memory



Secondary DLL Mapping

Process Memory



Secondary DLL Mapping [1,2]

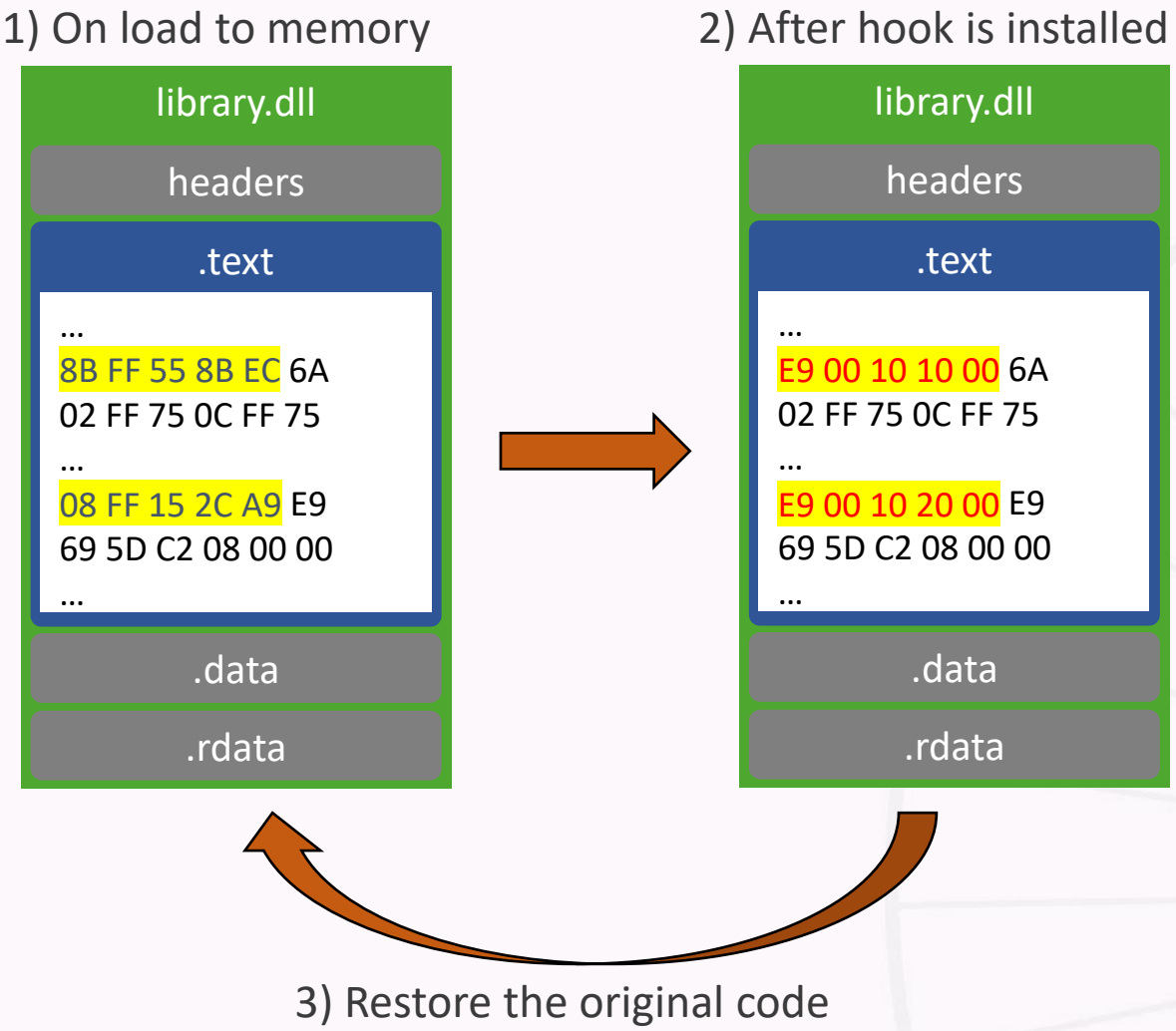
1. Manually load DLL from disk
   a) ReadFile()
   b) Reflective loading
2. Clone DLL
   a) CopyFile(<old_path>, <new_path>)
   b) LoadLibrary(<new_path>)
3. Section remapping
   a) CreateFile() + NtCreateSection(..., SEC_IMAGE, ...) \ NtOpenSection (“KnownDlls\...”)
   b) NtMapViewOfSection()
   - Does not handle relocations and initializations (imports\dependencies, data, ...)

Secondary DLL Mapping

Detection and trade-offs

Technique	Runtime Indicators	Forensic Artifacts	Drawbacks
Reflective Loading	Call stacks missing relevant DLLs	Floating PE copy in memory	Significantly different from standard operation
Clone DLL	Call stacks with unexpected DLLs identical to other DLLs	Identical PEs in memory	Internal\lower-level dependencies can be hooked
		Changes to file system	
Section Remapping	Call stacks with DLLs at different base address	Multiple mappings of same PE	Can’t be used for complex code

Binary Restoration \ Unhooking



Binary Restoration

1. Temporary Copy

a) Secondary mapping provides the original code

1. Manually Load DLL From Disk (Reflective Loading) [3,4]
   - Handle relocations according to the base address of the target DLL
2. Clone DLL [5]
   - Apply relocations after LoadLibrary according to the base address of the target DLL
3. Section Remapping [6]
   - No need to handle relocations (they are already there due to OS operation)

b) Restore code

- VirtualProtect() + memcpy()

Binary Restoration

2. Peer Ripping

a) Get a process handle
   1. Suspended child process [7]
      - CreateProcess(..., CREATE_SUSPENDED, ...)
      - Applicable only for ntdll.dll
   2. Debugged child process
      - CreateProcess(..., DEBUG_PROCESS, ...) + WaitForDebuggerEvent()
      - Set hardware breakpoint (SetThreadContext) at the loader functions [8] or use LOAD_DLL_DEBUG_EVENT*
   3. Existing process [9]
      - Not all processes are monitored
      - OpenProcess() [+ RtlCreateProcessReflection() \ PssCaptureSnapshot()]

b) NtReadVirtualMemory()

c) VirtualProtect() + memcpy()

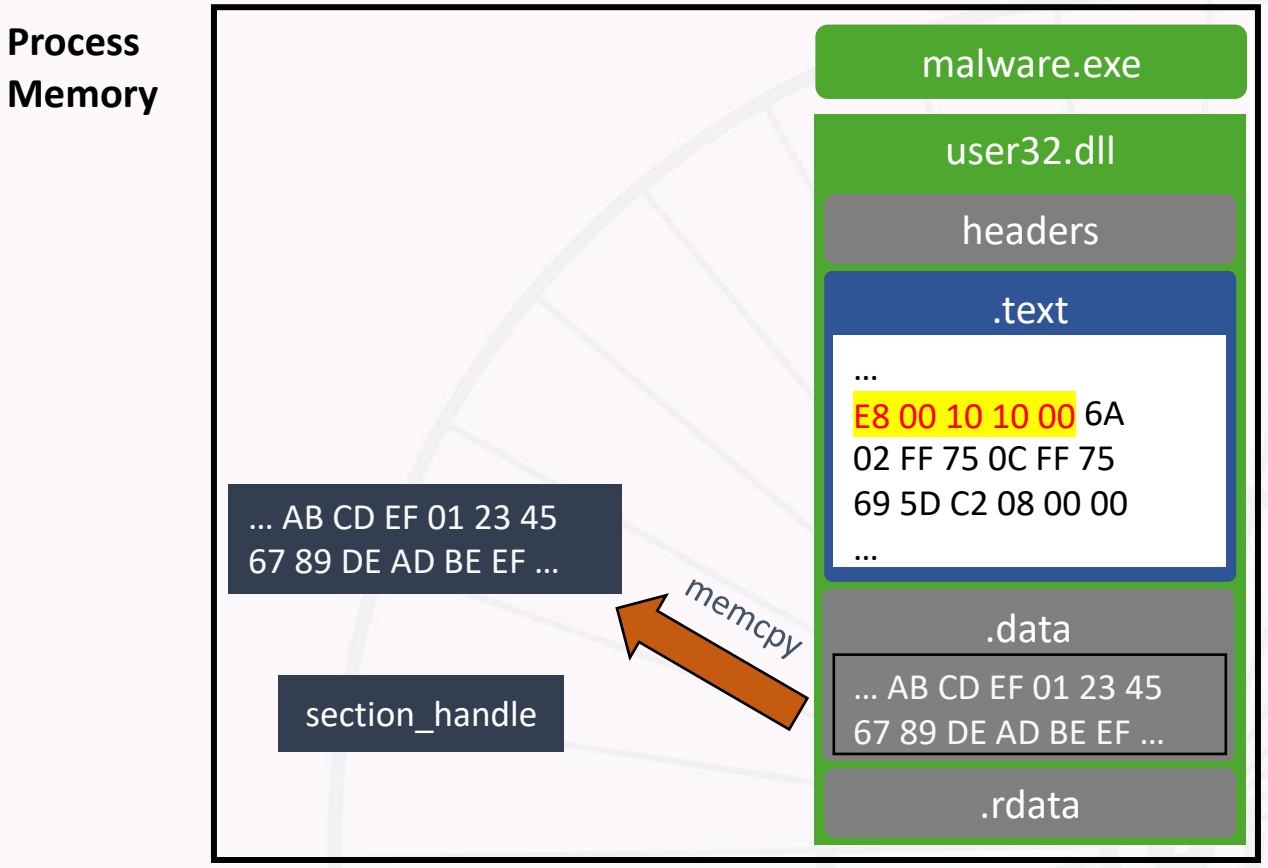
* https://learn.microsoft.com/en-us/windows/win32/api/minwinbase/ns-minwinbase-load_dll_debug_info

Binary Restoration

3. Section Refresh [2]

a) Capture
   - Writable, non-shared PE sections
   - Current memory protections

b) Refresh
   1) NtCreateSection(..., SEC_IMAGE, ...)

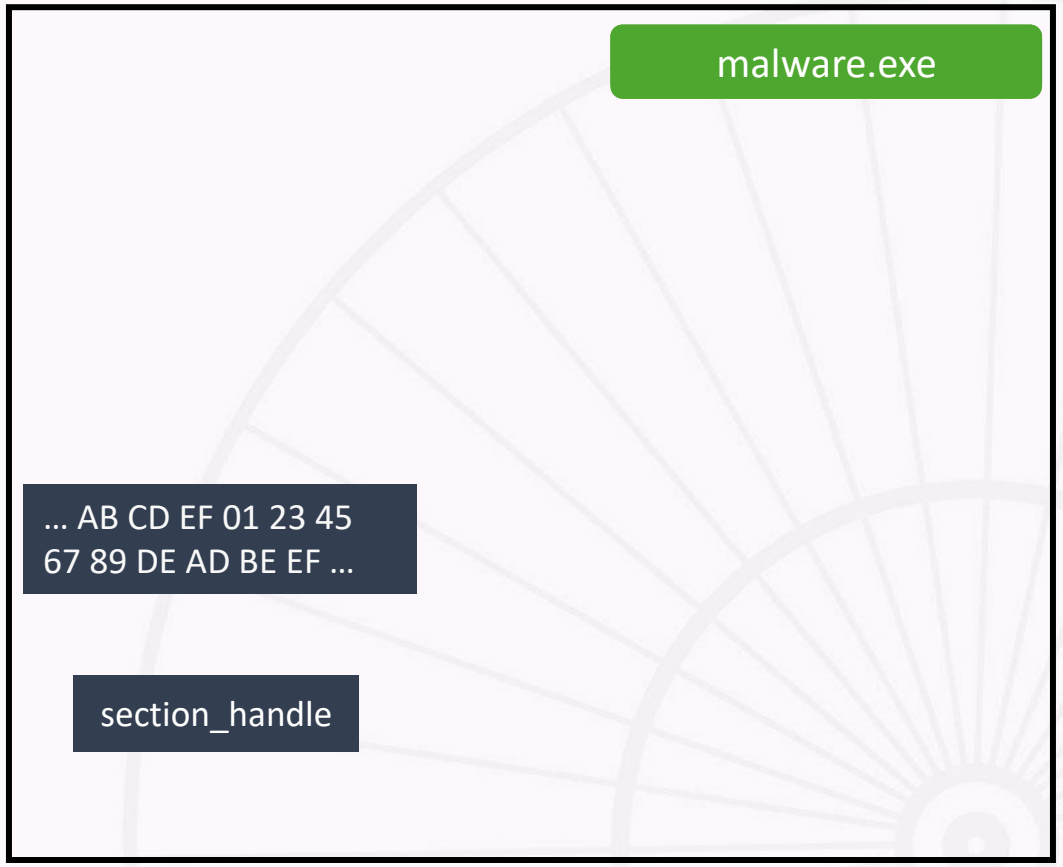


Binary Restoration

3. Section Refresh [2]

a) Capture
   - Writable, non-shared PE sections
   - Current memory protections

b) Refresh
   1) NtCreateSection(..., SEC_IMAGE, ...)
   2) NtUnmapViewOfSection(..., module_base, ...)

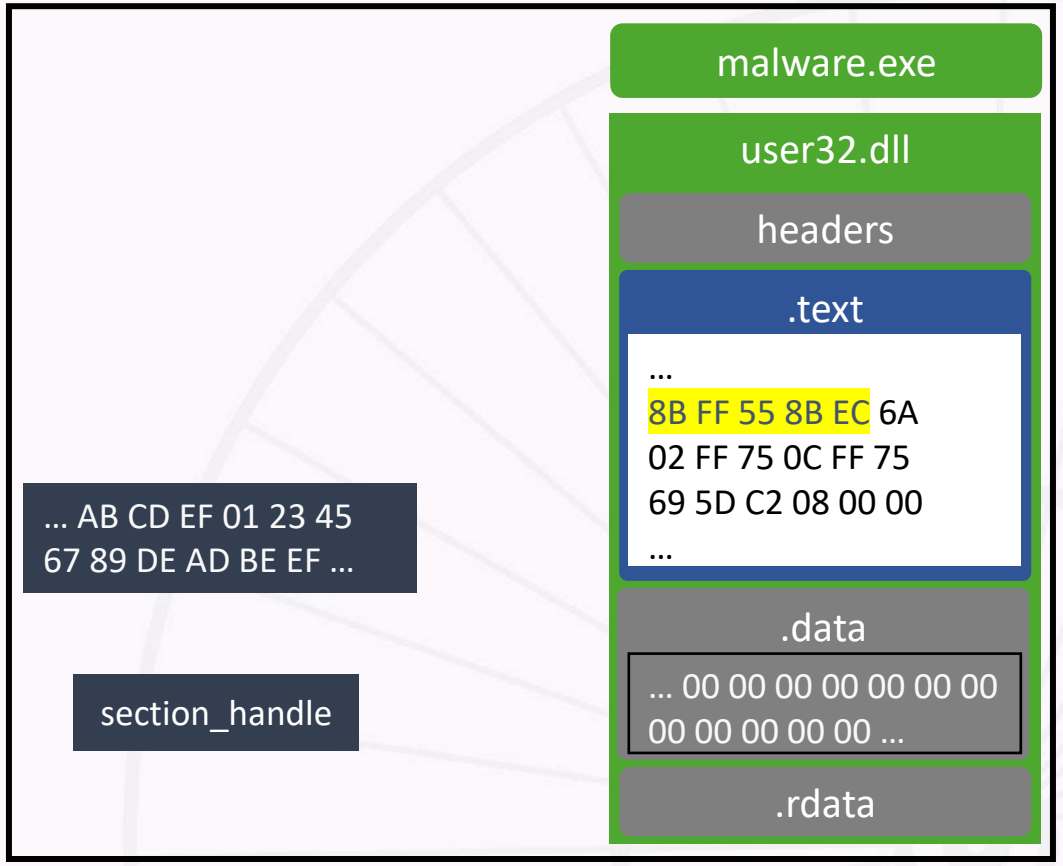


Binary Restoration

3. Section Refresh [2]

a) Capture
   - Writable, non-shared PE sections
   - Current memory protections

b) Refresh
   1) NtCreateSection(..., SEC_IMAGE, ...)
   2) NtUnmapViewOfSection(..., module_base, ...)
   3) NtMapViewOfSection(..., module_base)



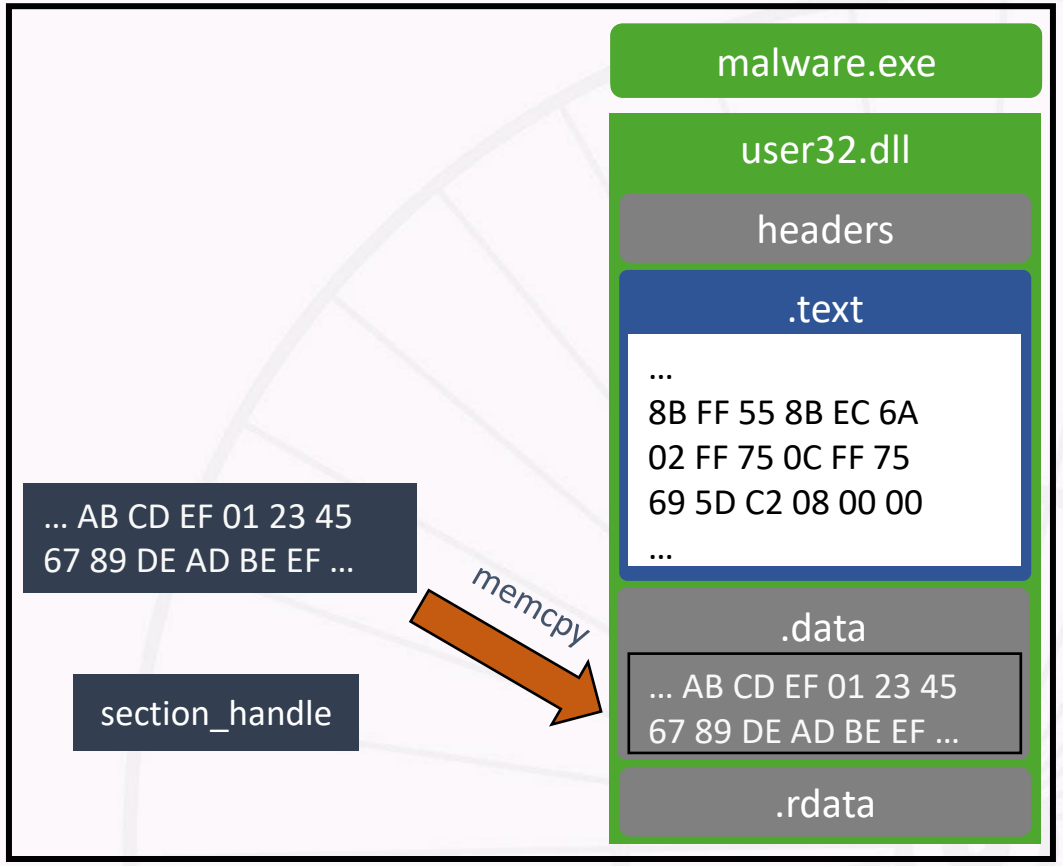
Binary Restoration

3. Section Refresh [2]

a) Capture
   - Writable, non-shared PE sections
   - Current memory protections

b) Refresh
   1) NtCreateSection(..., SEC_IMAGE, ...)
   2) NtUnmapViewOfSection(..., module_base, ...)
   3) NtMapViewOfSection(..., module_base)

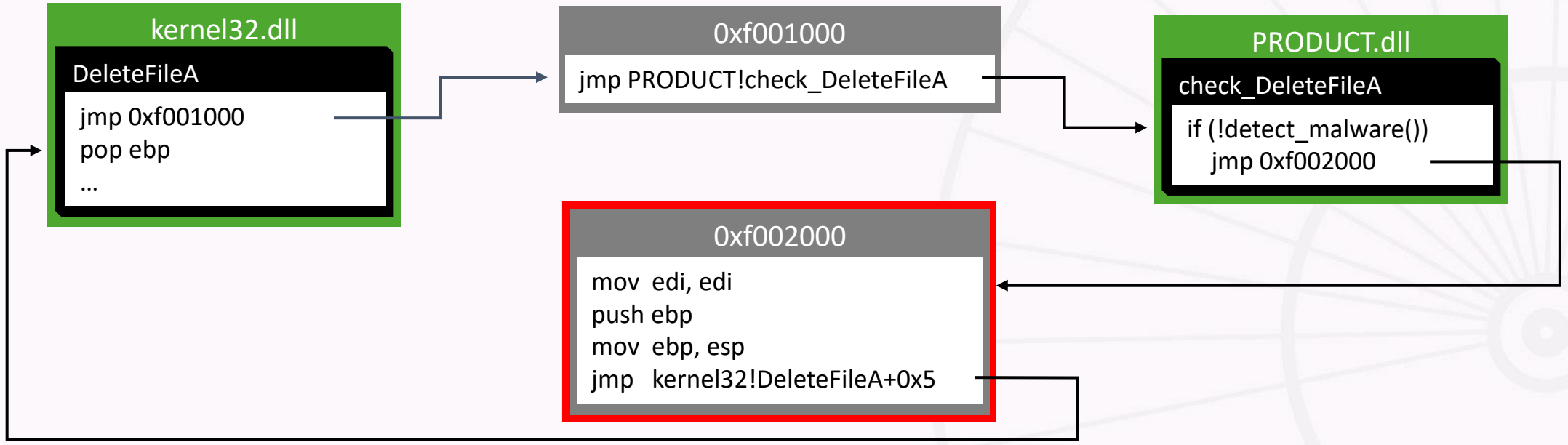
c) Restore
   - IAT, forwarder exports, CFG
   - Captured state



Binary Restoration

4. Short-circuiting

a) Find the original instructions copy in memory
   - MEM_COMMIT | MEM_PRIVATE | PAGE_EXECUTE
   - The control flow instruction back to the target function

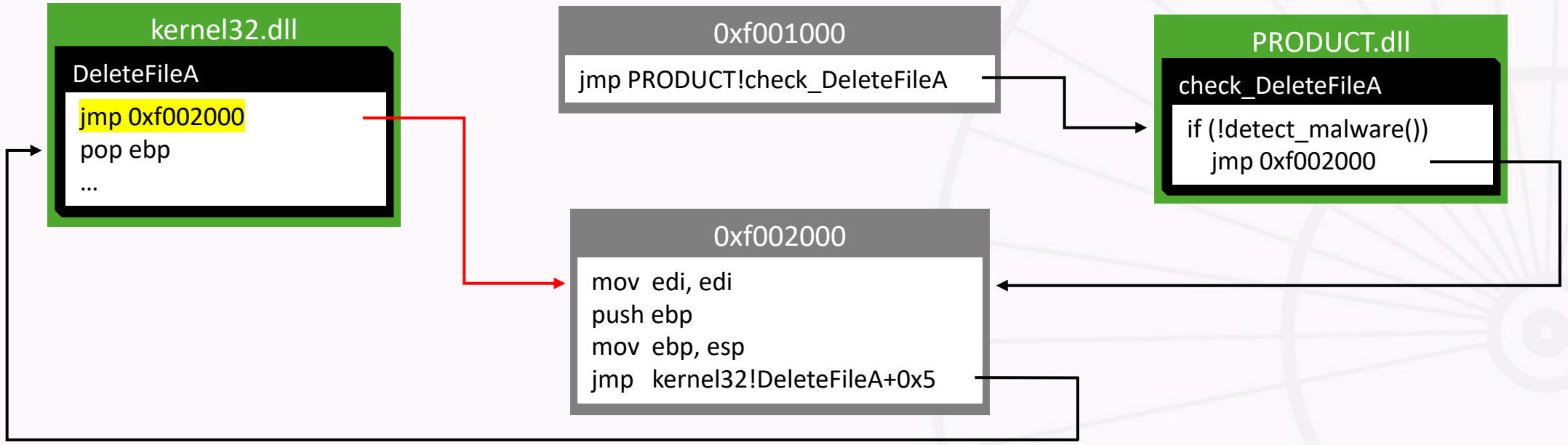


Binary Restoration

4. Short-circuiting

a) Find the original instructions copy in memory
   - MEM_COMMIT | MEM_PRIVATE | PAGE_EXECUTE
   - The control flow instruction back to the target function

b) Redirect
   1. Hook [10]

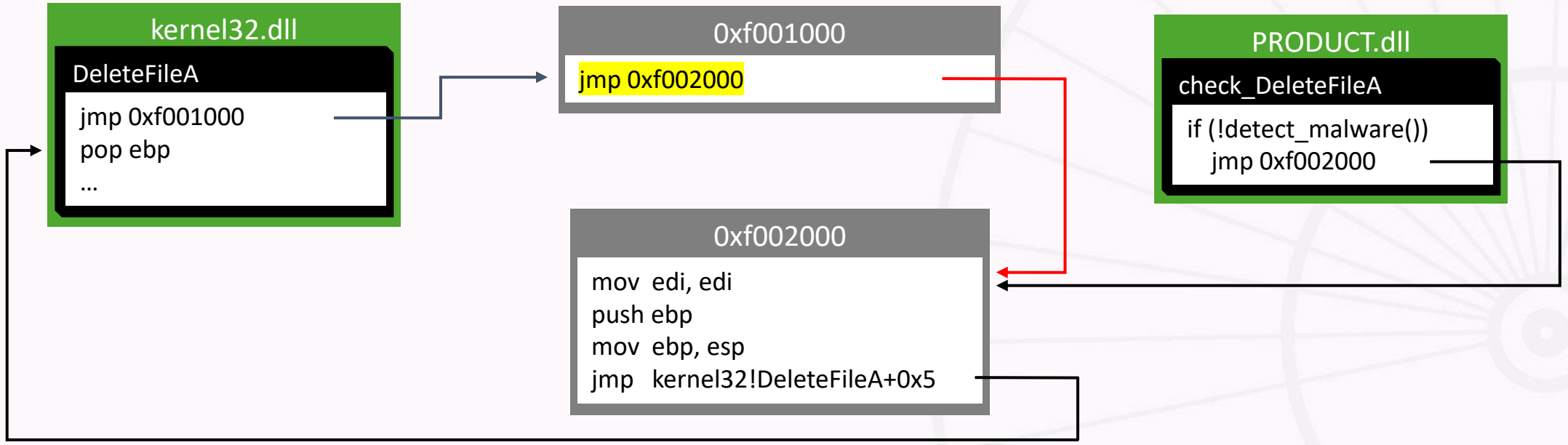


Binary Restoration

4. Short-circuiting

a) Find the original instructions copy in memory
   - MEM_COMMIT | MEM_PRIVATE | PAGE_EXECUTE
   - The control flow instruction back to the target function

b) Redirect
   1. Hook [10]
   2. Trampoline [11]

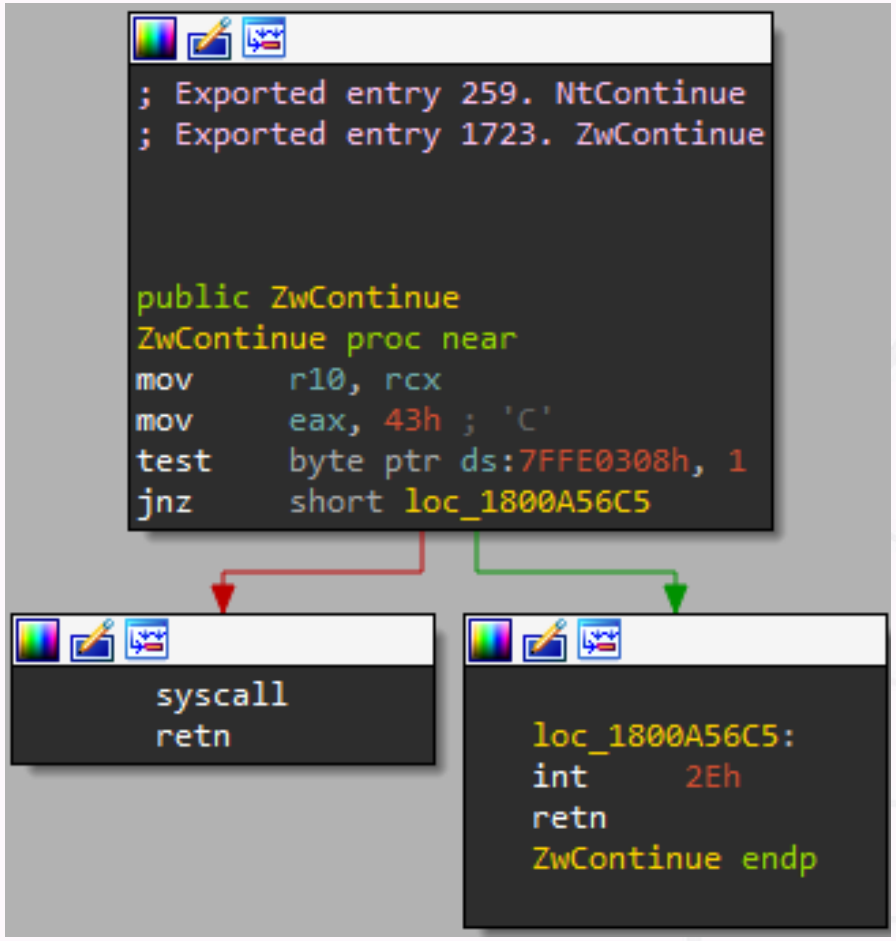


Binary Restoration

Detection and trade-offs

Technique		Runtime Indicators	Forensic Artifacts	Drawbacks
Temporary Copy	Reflective Loading		Hooks removed [32]	
	Clone DLL	Multiple mappings of identical PEs		
	Section Remapping	Multiple mappings of same PE		
Peer Ripping	Suspended Child			
	Debugged Child	Debugged child process		
	Existing Process			
Section Refresh		Reoccurring mappings of same PE		Can’t be used on ntdll.dll
Short-circuiting	Trampoline		Hooks modified	
	Hook			

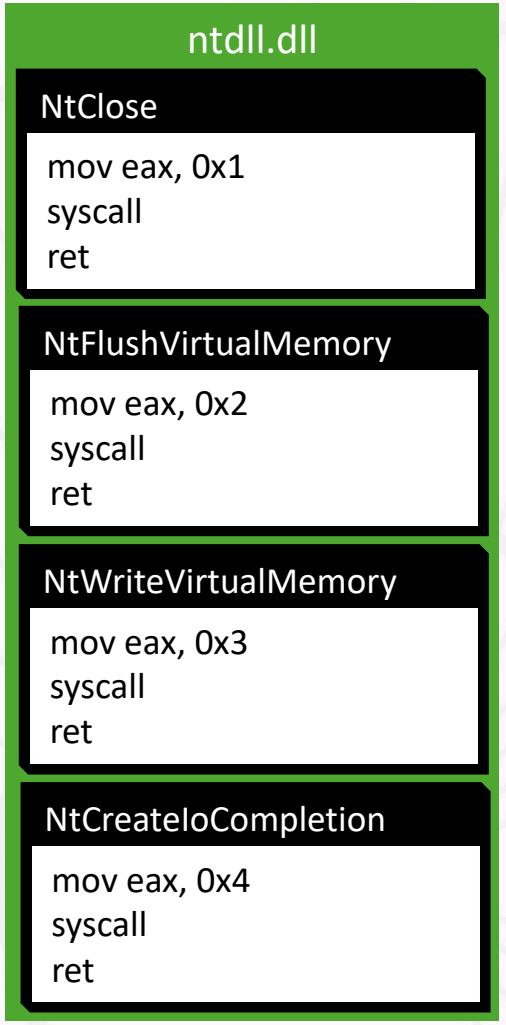
Direct System Call Invocation



Direct System Call Invocation

1. ntdll.dll Parsing

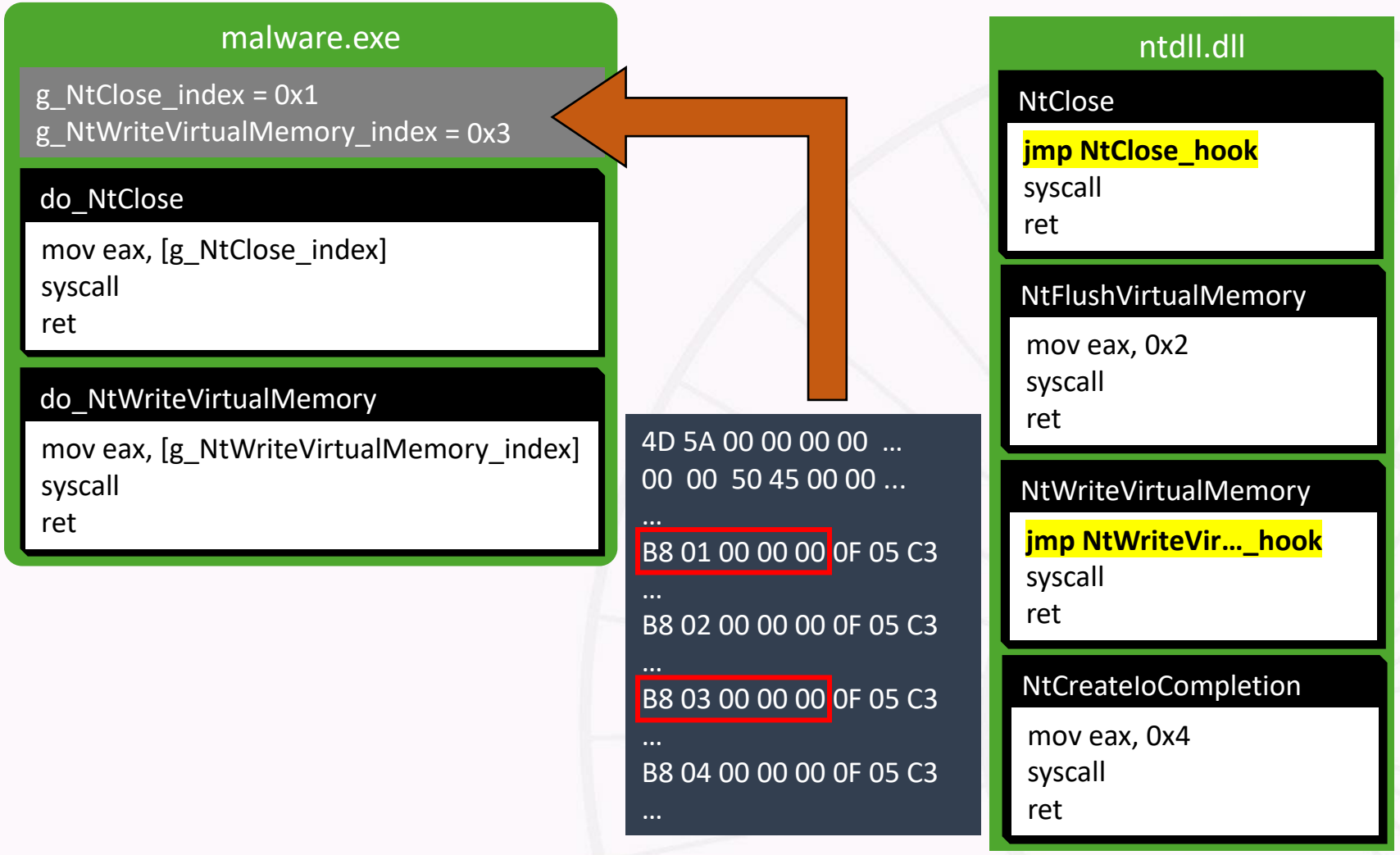
1. Instructions



Direct System Call Invocation

1. ntdll.dll Parsing

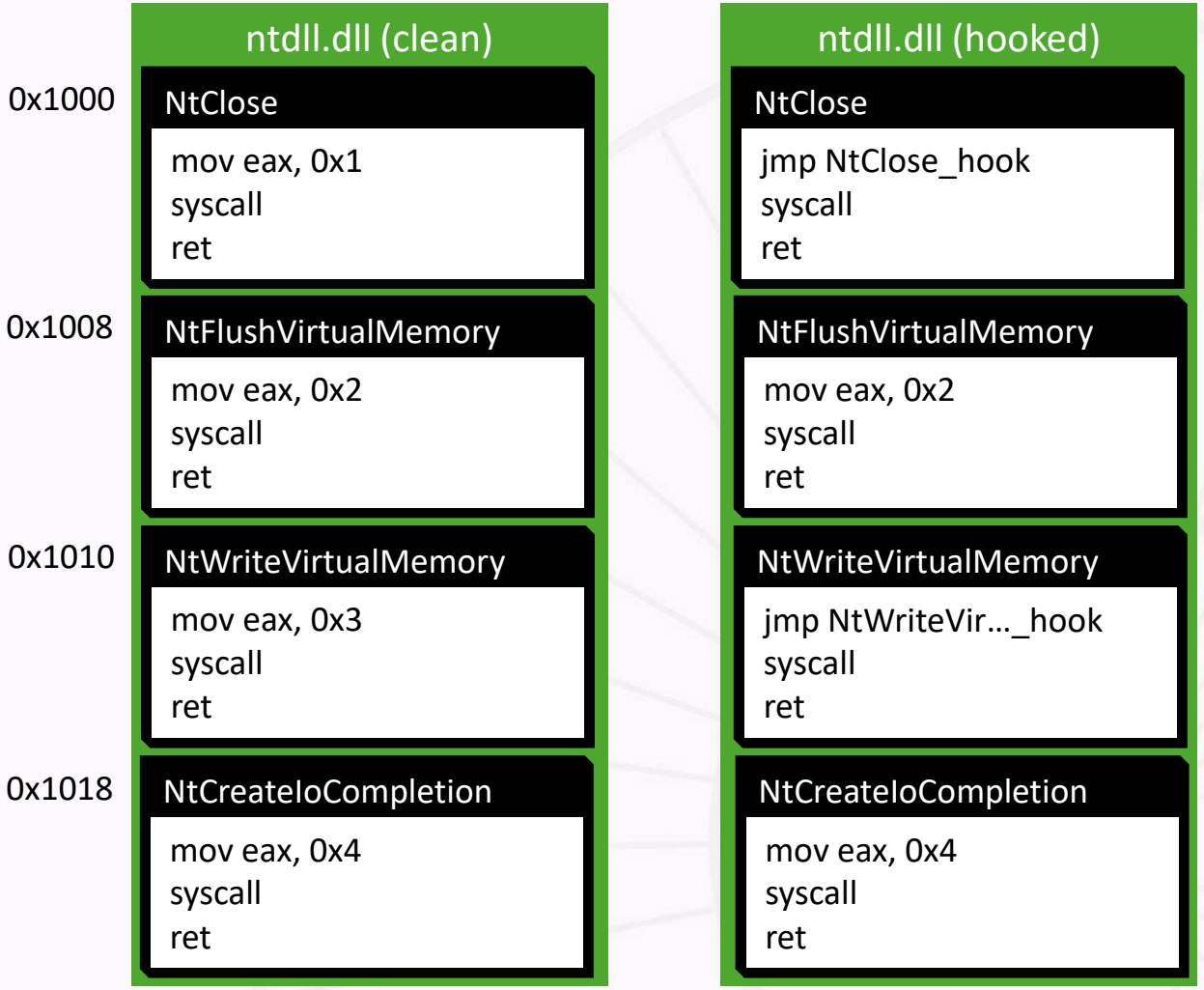
1. Instructions
    1. From disk [1,2,12]
    2. From memory [13]



Direct System Call Invocation

1. ntdll.dll Parsing

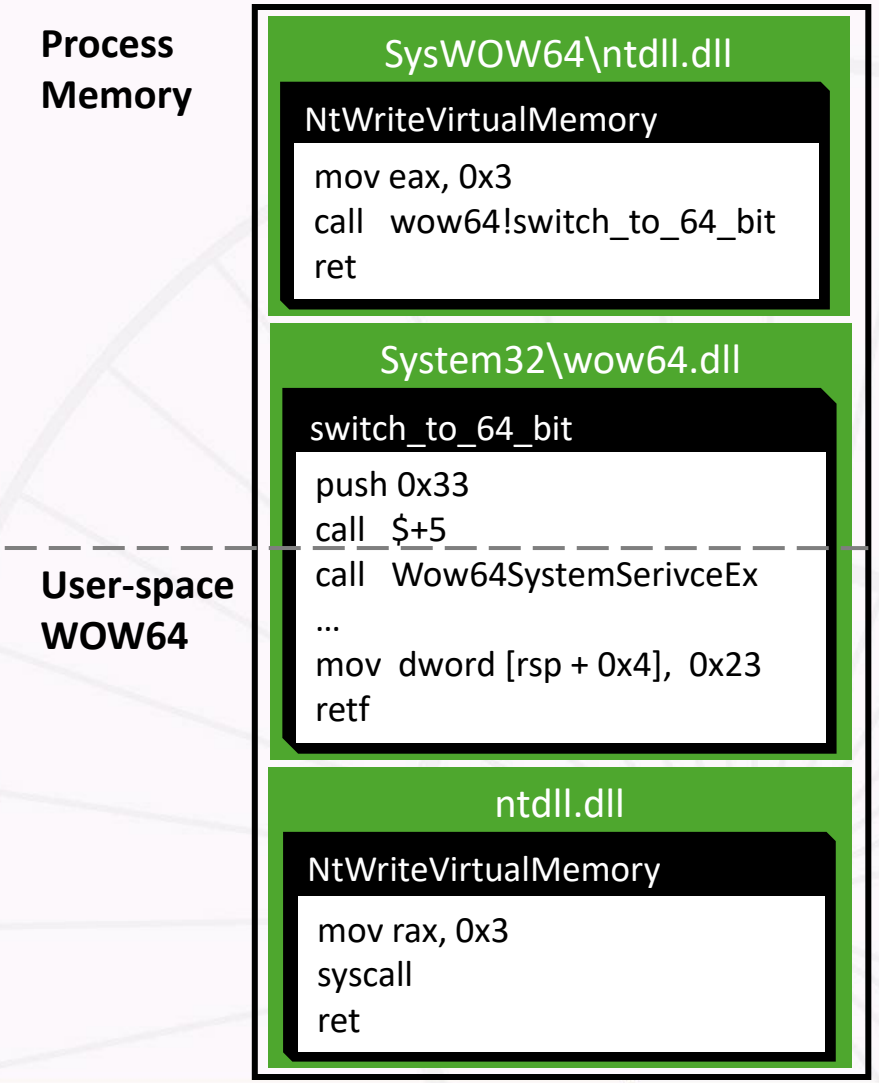
1. Instructions
   1. From disk [1,2,12]
   2. From memory [13]
2. Count [14,15]
3. Proximity check [16,17,18]



Direct System Call Invocation

2. Heaven's Gate

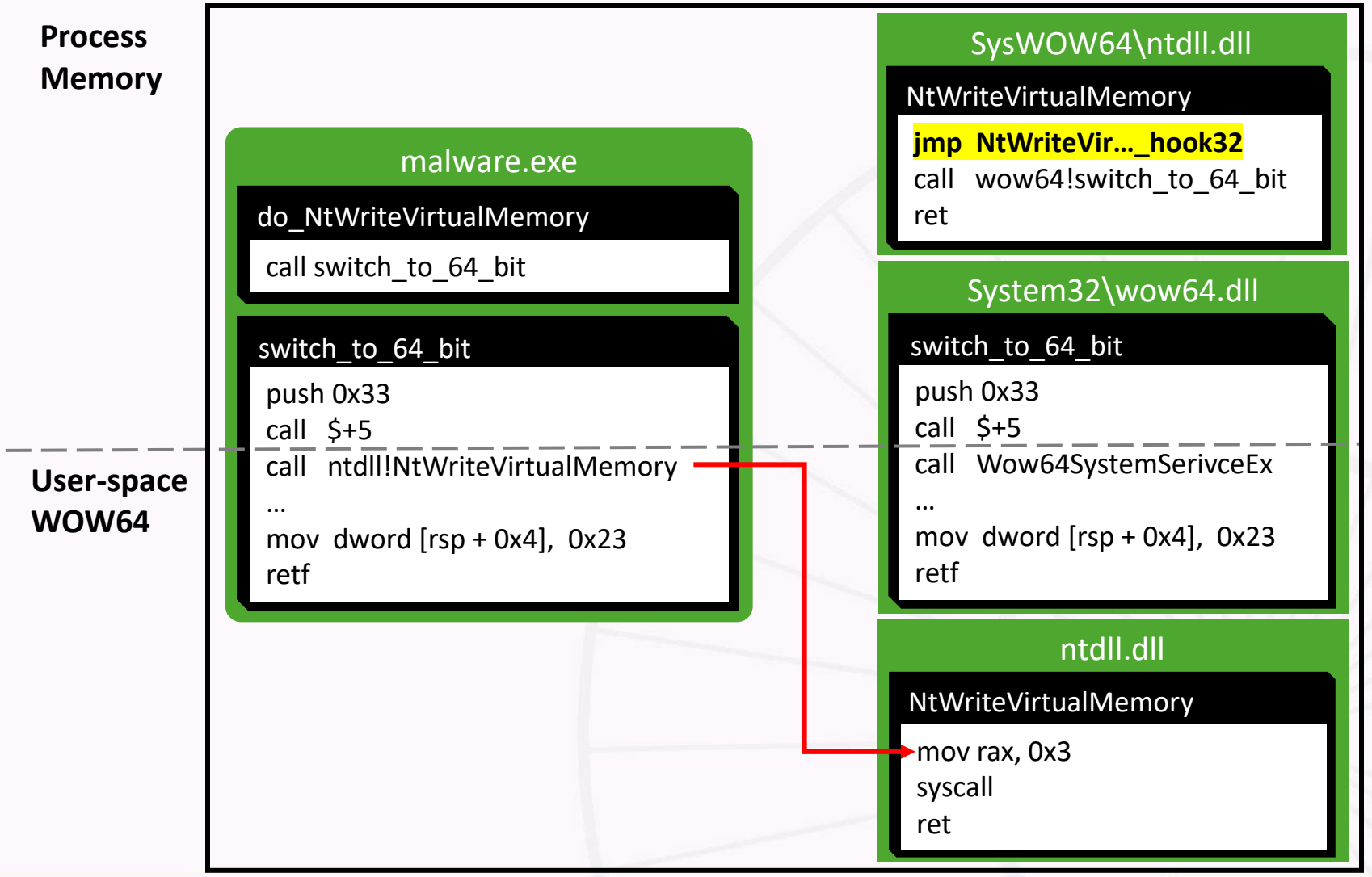
- Make system calls from within WOW64 emulation layer
  - Only 32-bit application on 64-bit Windows



Direct System Call Invocation

2. Heaven's Gate

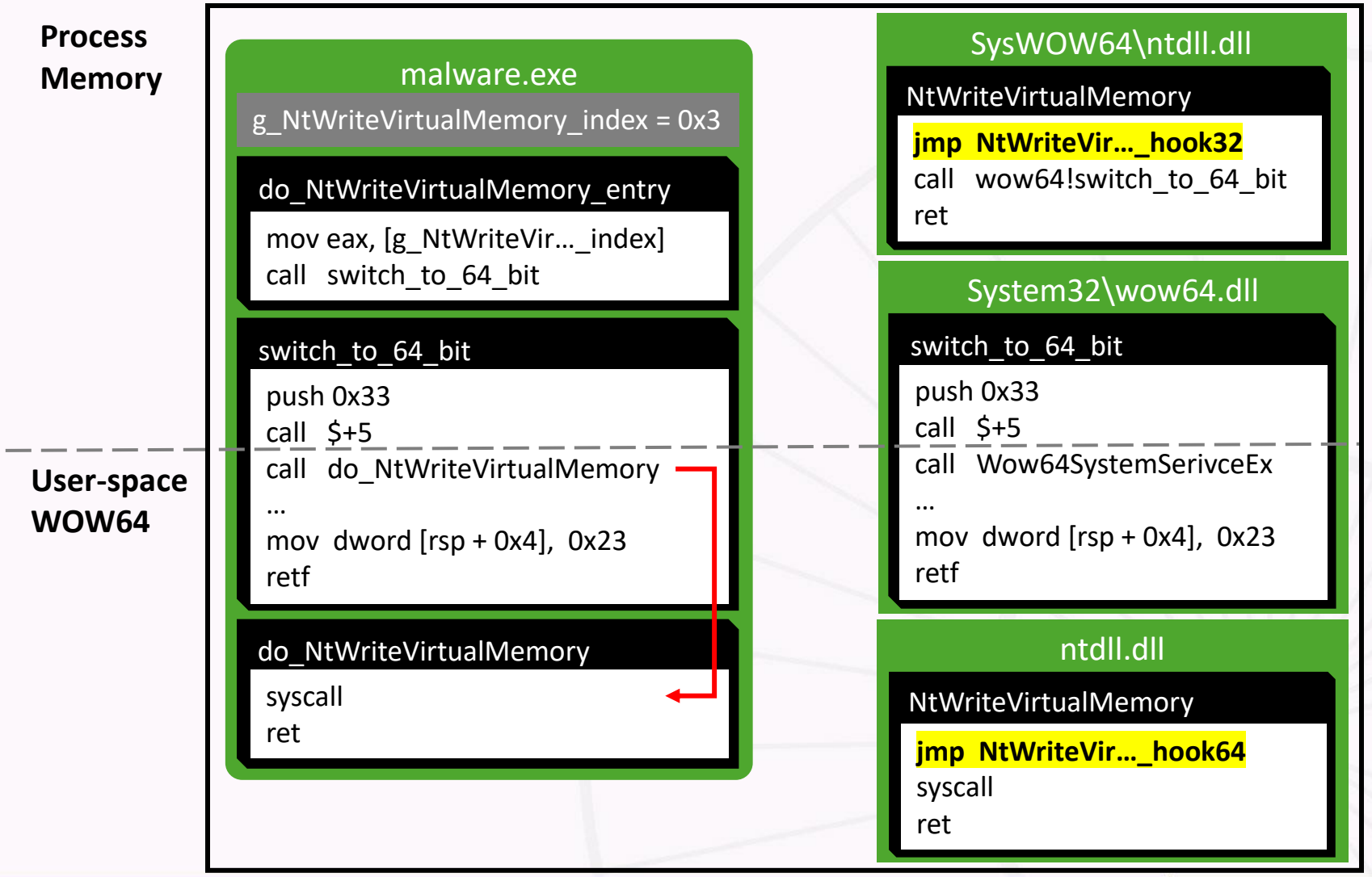
1. Skip WOW64 layer [1,2]



Direct System Call Invocation

2. Heaven's Gate

1. Skip WOW64 layer [1,2]
2. Completely native [1,2]



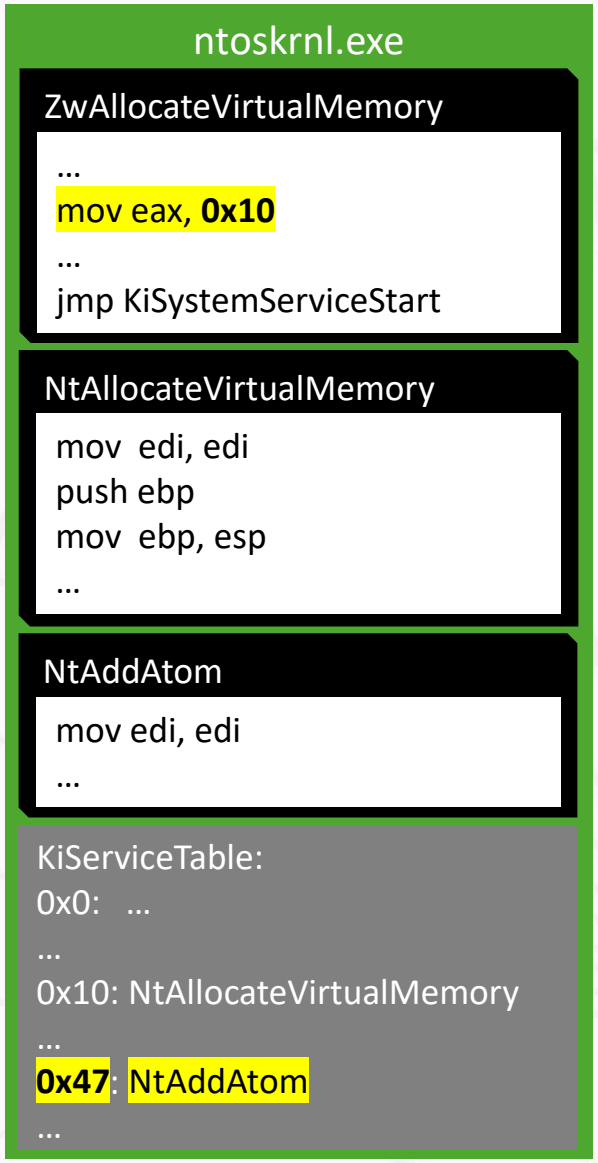
Direct System Call Invocation

3. ntoskrnl.exe Parsing [2]

a) Zw* exported
   1. GetProcAddress(ntoskrnl, “Zw…”)
   2. Look for “mov eax, <imm>”

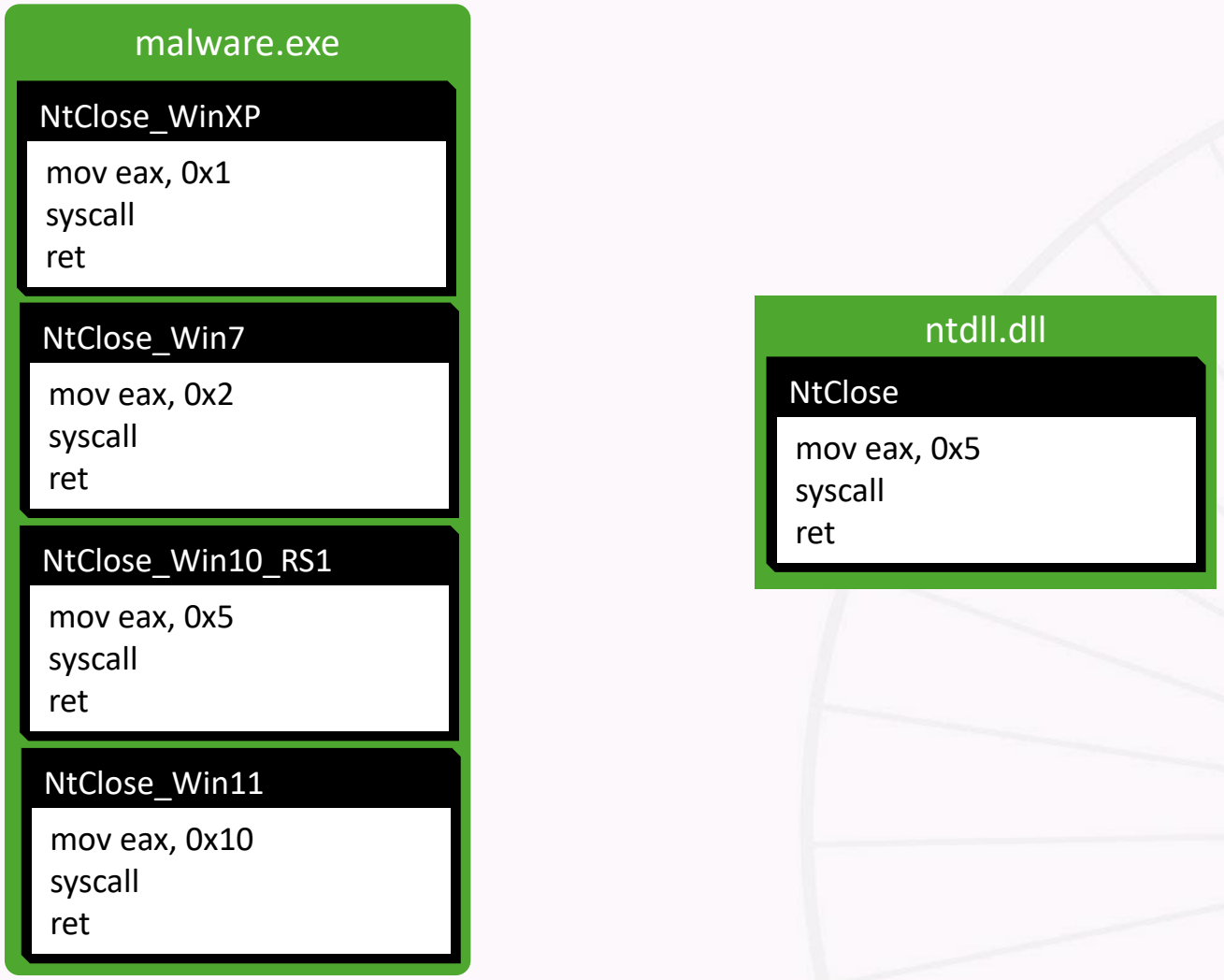
b) Some functions only have the Nt version exported
   1. GetProcAddress(ntoskrnl, “Nt…”)
   2. Locate ntoskrnl!KiServiceTable
   3. Traverse the table to find reference to the target

- A partial set of system calls but still quite comprehensive
  - ZwCreate*, ZwQuery\Set*
  - ZwReadFile, ZwWriteFile
  - ZwMap\UnmapViewOfSection
  - ZwAllocate\Protect\FreeVirtualMemory
  - ZwLoad\UnloadDriver
  - ZwYieldExecution (Sleep)
  - Registry API
  - Transactions API



Direct System Call Invocation

4. Bring Your Own Index (BYOI) [2,19]



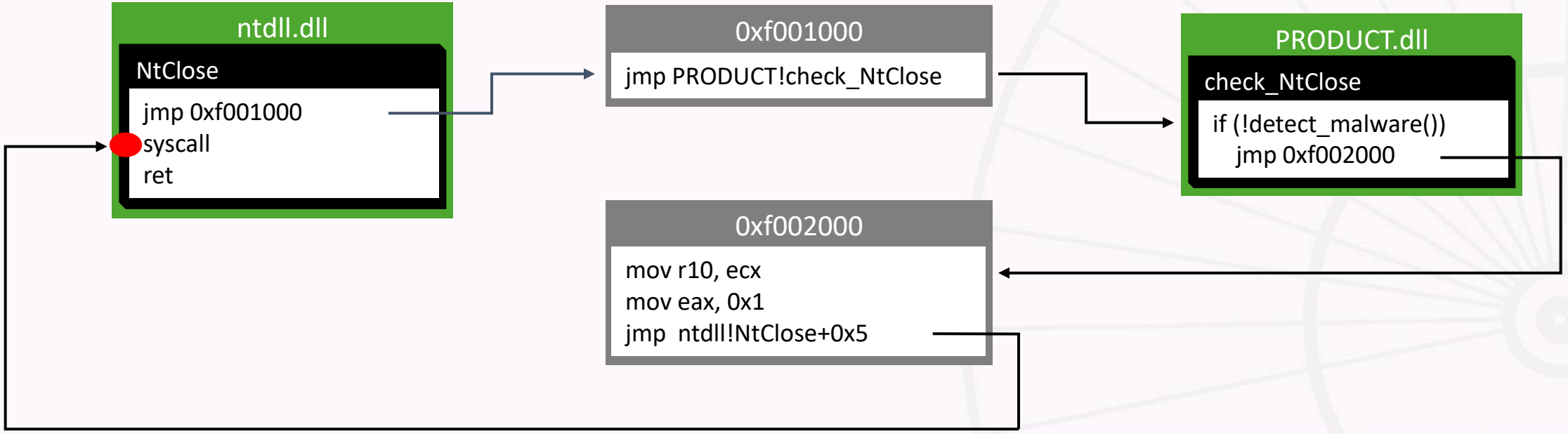
Direct System Call Invocation

5. Dynamic Resolution

a) SetUnhandledExceptionFilter [20] \ RtlAddVectoredExceptionHandler

b) Place a breakpoint on the syscall instruction

c) Call the function



Direct System Call Invocation

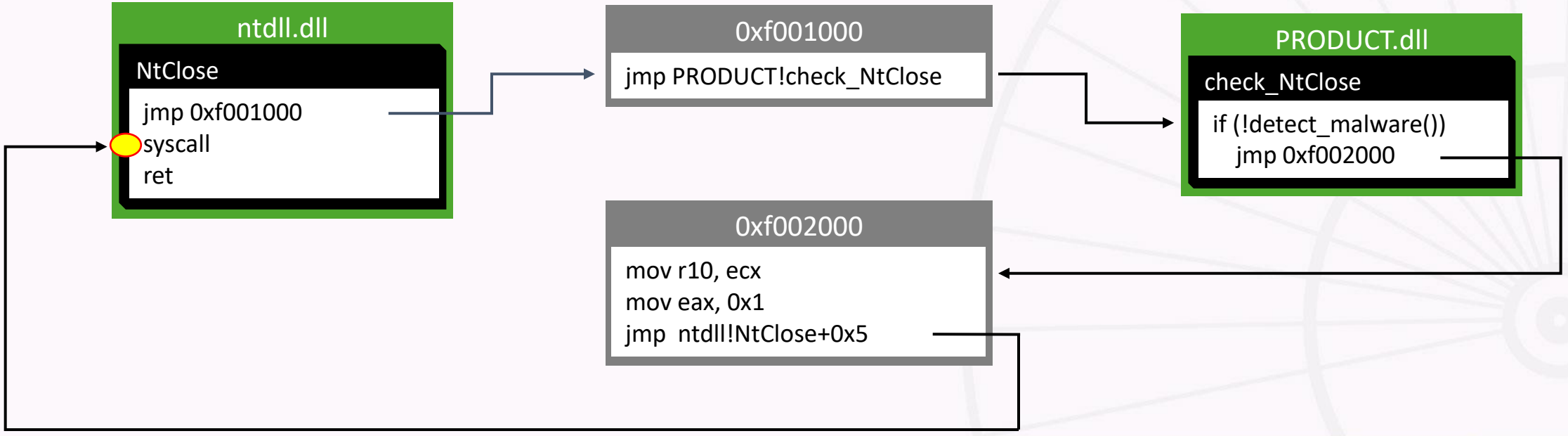
5. Dynamic Resolution

a) SetUnhandledExceptionFilter [20] \ RtlAddVectoredExceptionHandler

b) Place a breakpoint on the syscall instruction

c) Call the function

d) Get eax's value from the CONTEXT structure on the exception handler



Direct System Call Invocation

Detection and trade-offs

<table>
<tr><th colspan="2">Technique</th><th>Runtime Indicators</th><th>Forensic Artifacts [32]</th><th>Drawbacks</th></tr>
<tr><td colspan="2">ntdll.dll Parsing</td><td rowspan="4">Call stacks missing ntdll.dll</td><td rowspan="5"></td><td rowspan="5">Can’t be used generically</td></tr>
<tr><td colspan="2">ntoskrnl.exe Parsing</td></tr>
<tr><td colspan="2">Bring Your Own Index (BYOI)</td></tr>
<tr><td colspan="2">Dynamic Resolution</td></tr>
<tr><td rowspan="2">Heaven’s Gate</td><td>Skip WOW64</td><td>Call stacks missing WOW64 system DLLs</td></tr>
<tr><td>Native</td><td>Call stacks missing WOW64 system DLLs and ntdll.dll</td></tr>
</table>

- Additional runtime indicator on Windows with VBS enabled – the “syscall” instruction is used instead of “int 2E”

Code Splicing \ Byte Stealing

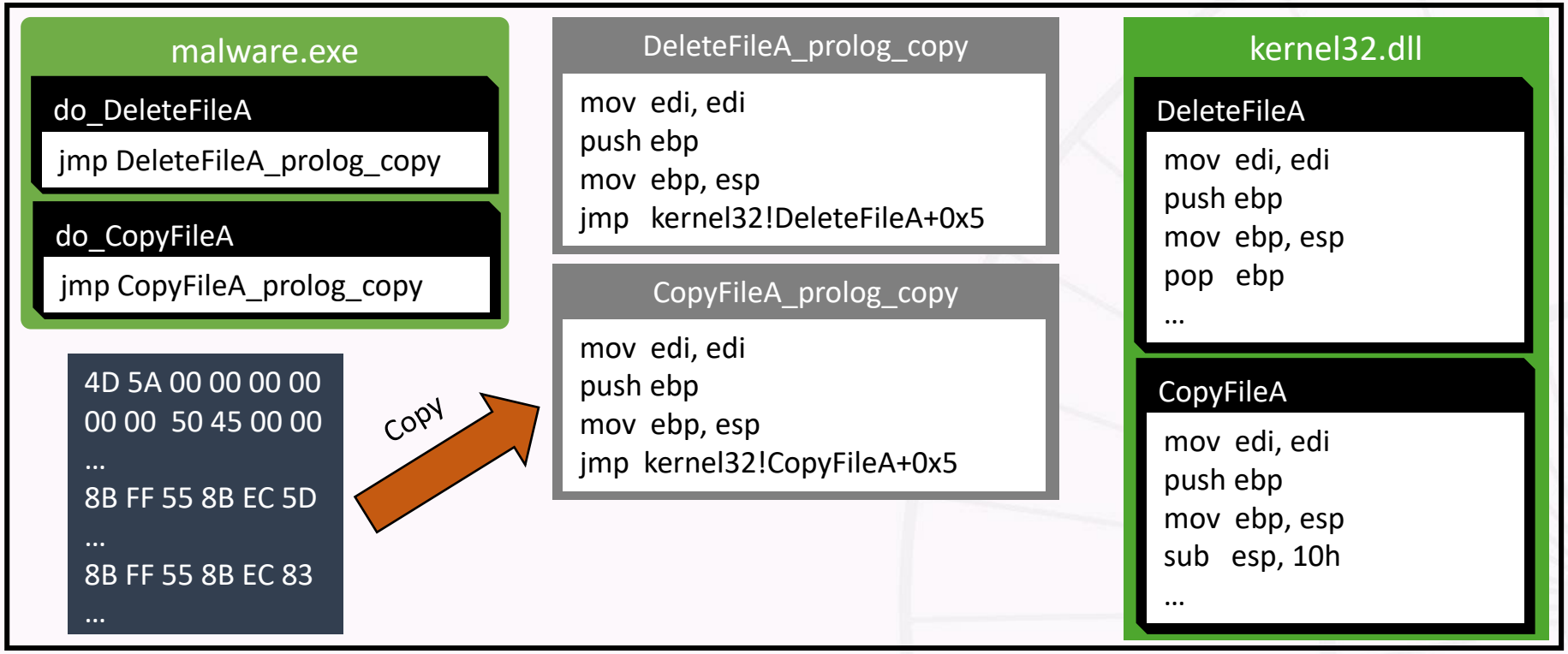
- Rebuild function stubs elsewhere
- Can be considered as “Indirect System Call Invocation” with the target function a system service

Code Splicing

1. From Disk [1,2]

a) Use “Manually Loading DLL From Disk” (Reflective Loading)

b) Extract the target function’s instructions

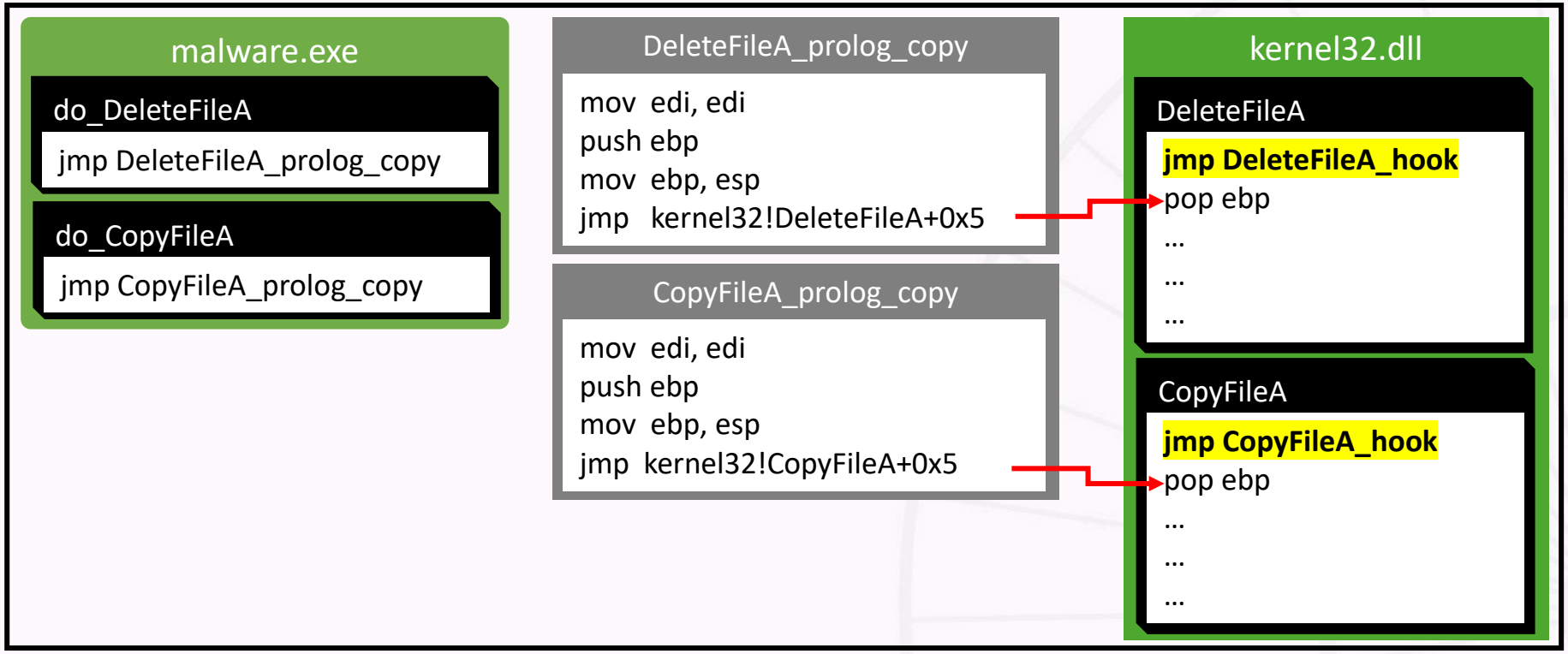


Code Splicing

1. From Disk [1,2]

a) Use “Manually Loading DLL From Disk” (Reflective Loading)

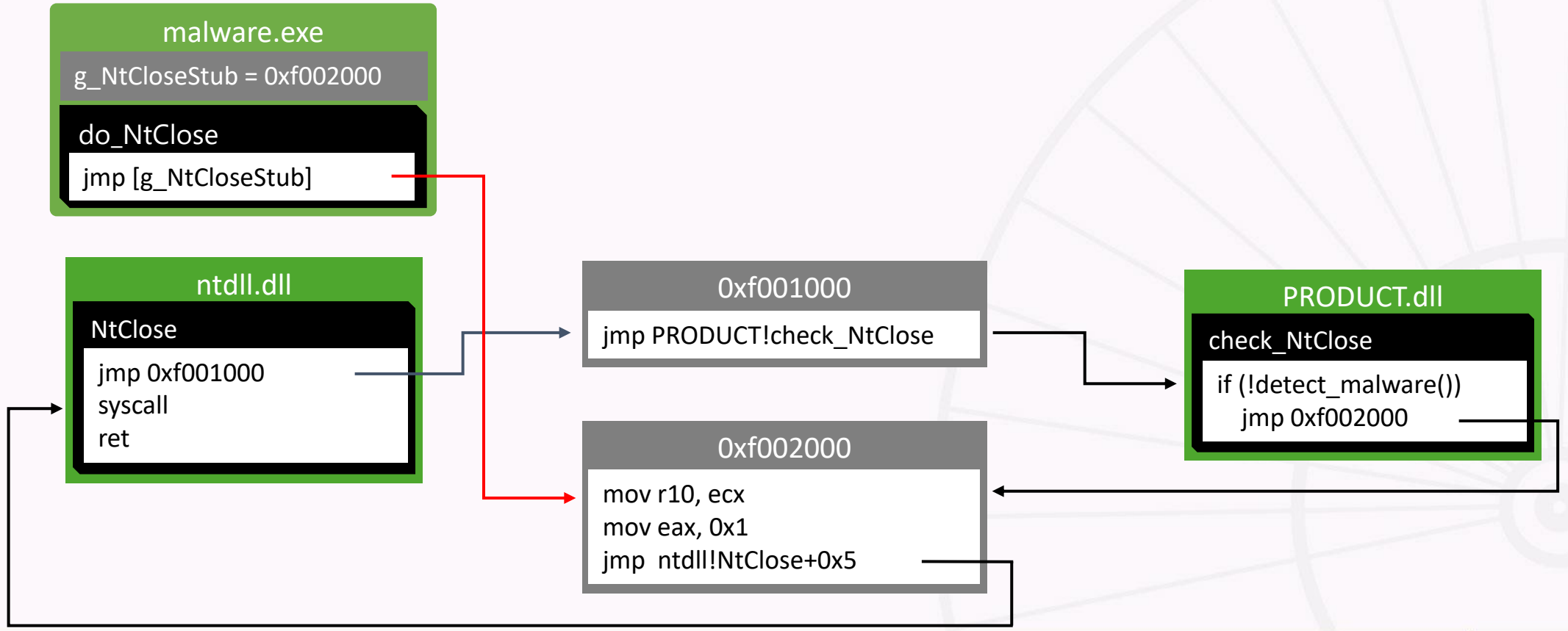
b) Extract the target function’s instructions



Code Splicing

2. Stub Reuse [21]

- The stubs are already in memory and it's possible to find them (remember Short-circuiting?)
  - (MEM_COMMIT | MEM_PRIVATE | PAGE_EXECUTE) + control flow instruction back to the target function



Code Splicing

Detection and trade-offs

Technique	Runtime Indicators	Forensic Artifacts	Drawbacks
From Disk			Internal\lower-level dependencies can be hooked
Stub Reuse			

- ✓ Introduction
- ✓ Hook Evasion tactic
- ❑ Argument Forgery tactic
- ❑ Engine Disarming tactic
- ❑ Conclusions

Argument Forgery Tactic

Overview

- Swap the arguments after they were inspected and before they are used by the target function
  - Time-Of-Check to Time-Of-Use (TOCTOU)
- Halt execution without manipulating memory
  1. Hardware breakpoints [22] (like Dynamic Resolution)
  2. Argument that causes the protection engine to trigger an exception [23]
     - Vendor-specific so not a reliable option

Argument Forgery Tactic

Implementation

a) SetUnhandledExceptionFilter \ RtlAddVectoredExceptionHandler

b) Place a breakpoint on the instruction the protection engine returns to in the target function

```
0:000> u wininet!InternetOpenUrlA L4
WININET!InternetOpenUrlA:
00007ffe`d41d7f30 e9e19365f0      jmp     product!InternetOpenUrlA_hook
00007ffe`d41d7f35 58              pop     rax
00007ffe`d41d7f36 084889          or      byte ptr [rax-77h],cl
00007ffe`d41d7f39 6810488970      push    70894810h
```

```
0:000> u wininet!InternetOpenUrlA L4
WININET!InternetOpenUrlA:
00007ffe`d41d7f30 488bc4          mov     rax,rsp
00007ffe`d41d7f33 48895808        mov     qword ptr [rax+8],rbx
00007ffe`d41d7f37 48896810        mov     qword ptr [rax+10h],rbp
00007ffe`d41d7f3b 48897018        mov     qword ptr [rax+18h],rsi
```

```
0:000> u wininet!InternetOpenUrlA+0x7 L2
WININET!InternetOpenUrlA+0x7:
00007ffe`d41d7f37 48896810        mov     qword ptr [rax+10h],rbp
00007ffe`d41d7f3b 48897018        mov     qword ptr [rax+18h],rsi
```

Argument Forgery Tactic

Implementation

a) SetUnhandledExceptionFilter \ RtlAddVectoredExceptionHandler

b) Place a breakpoint on the instruction the protection engine returns to in the target function

c) Call the function with fake values

```
0:000> u malware!main+0xe5
malware!main+0xe5:
00007ff6`d7a91c15 ff151df60000    call    qword ptr [malware!_imp_InternetOpenUrlA]
0:000> da rdx
00007ff6`b056ad10  "https://benign-domain.com"
```

Argument Forgery Tactic

Implementation

a) SetUnhandledExceptionFilter \ RtlAddVectoredExceptionHandler

b) Place a breakpoint on the instruction the protection engine returns to in the target function

c) Call the function with fake values

d) Pass through the protection engine logic (TOC)

```
0:000> u product!InternetOpenUrlA_hook+0x8a
product!InternetOpenUrlA_hook+0x8a:
00007ffe`c4831b3a ff15a8110200    call    qword ptr [product!orig_InternetOpenUrl]
0:000> da rdx
00007ff6`b056ad10  "https://benign-domain.com"
0:000> dq product!orig_InternetOpenUrl L1
00007ffe`c4852ce8  00007ffe`d4090000
0:000> u 0x00007ffed4090000 L3
00007ffe`d4090000 488bc4          mov     rax,rsp
00007ffe`d4090003 48895808        mov     qword ptr [rax+8],rbx
00007ffe`d4090007 e92b7f1400      jmp     WININET!InternetOpenUrlA+0x7
```

Argument Forgery Tactic

Implementation

a) SetUnhandledExceptionFilter \ RtlAddVectoredExceptionHandler

b) Place a breakpoint on the instruction the protection engine returns to in the target function

c) Call the function with fake values

d) Pass through the protection engine logic (TOC)

e) The exception handler swaps the arguments
   - Registers in the CONTEXT structure
   - On stack

```
0:000> u wininet!InternetOpenUrlA+0x7 L2
WININET!InternetOpenUrlA+0x7:
00007ffe`d41d7f37 48896810        mov     qword ptr [rax+10h],rbp
00007ffe`d41d7f3b 48897018        mov     qword ptr [rax+18h],rsi
0:000> da rdx
00007ff6`d7a9ace0  "https://malicious-domain.com"
```

Argument Forgery Tactic

Implementation

a) SetUnhandledExceptionFilter \ RtlAddVectoredExceptionHandler

b) Place a breakpoint on the instruction the protection engine returns to in the target function

c) Call the function with fake values

d) Pass through the protection engine logic (TOC)

e) The exception handler swaps the arguments
   - Registers in the CONTEXT structure
   - On stack

f) Execution continue (TOU)

```
0:000> u wininet!InternetOpenUrlA+0x7 L2
WININET!InternetOpenUrlA+0x7:
00007ffe`d41d7f37 48896810        mov     qword ptr [rax+10h],rbp
00007ffe`d41d7f3b 48897018        mov     qword ptr [rax+18h],rsi
0:000> da rdx
00007ff6`d7a9ace0  "https://malicious-domain.com"
```

Argument Forgery Tactic

Detection and trade-offs

Runtime Indicators	Forensic Artifacts	Drawbacks
Hardware breakpoints set without a debugger attached		The protection engine is still invoked

- ✓ Introduction
- ✓ Hook Evasion tactic
- ✓ Argument Forgery tactic
- ❑ Engine Disarming tactic
- ❑ Conclusions

Engine Disarming Tactic

1. FreeLibrary [24] or trigger the engine's unload function (vendor-specific)
2. Unmap all DLLs [25] (like "Section Refresh" just without the "Refresh")
3. Allow to load only MS-signed binaries (using process mitigation policies) [26,27]
4. Preloading in new child process
    1. Debugged process [28] – break on image load and switch DLL entrypoint to do nothing on load
    2. Inject the process [29,30] – use AppVerifier and Shim Engine callbacks to run before the protection engine and prevent its DLLs from loading into it
5. Other vendor-specific implementation issues (e.g. set hook disabled flag [31])

- All the generic methods (1-4) can be detected in runtime as the protection engine isn't present

- ✓ Introduction
- ✓ Hook Evasion tactic
- ✓ Argument Forgery tactic
- ✓ Engine Disarming tactic
- ❑ Conclusions

Conclusions

Limitations of purely user-mode endpoint security solutions

1. Boot-time protection and early enough start (ELAM)
2. Self-protection (hardening)
3. Access to OS and critical processes
   - Monitor all parts of the OS (RPC, privilege escalation, etc.)
4. Real-time or rapid enough prevention

- P.S. – it's still prone to stability issues (especially with "Critical Process")*

* https://devblogs.microsoft.com/oldnewthing/20180216-00/?p=98035

Conclusions

Closing remarks

- Trivial to implement, simple to use (most have source code available)
  - Many techniques\variants are built on similar approaches and share certain primitives
- Most prolific post-exploitation technique, even more than code injection
  - Hook evasion count went up from 12 to 20 in just 5 years
  - Generic engine disarming count grew from 2 to 5 (and additional other non-generic) in the same time
- Prerequisite for other EDR evasion (as code execution is first required)
  - ETW and AMSI bypasses
  - Removing EDRs' kernel callbacks
- Additional layers of evasions can be chained to thwart detection (like call stack spoofing)
  - Leverage CET shadow stack [33] or transition-based code tracing [34] when controlling the hardware
- Using user-mode hooks for security is insufficient
  - Underlying flaw – the reliance on the same execution environment that is intended to be protected

References

1. https://www.first.org/resources/papers/telaviv2019/Ensilo-Omri-Misgav-Udi-Yavo-Analyzing-Malware-Evasion-Trend-Bypassing-User-Mode-Hooks.pdf
2. https://www.youtube.com/watch?v=Zk_8nQJeOQg&pp=ygUJYnNpZGVzdGx2
3. https://breakdev.org/defeating-antivirus-real-time-protection-from-the-inside/
4. https://files.speakerdeck.com/presentations/739577f45e014bfa80c0eca0895f1ead/You_re_Off_the_Hook.pdf
5. https://www.cybereason.com/blog/operation-cuckoobees-a-winnti-malware-arsenal-deep-dive
6. https://www.ired.team/offensive-security/defense-evasion/how-to-unhook-a-dll-using-c++
7. https://blog.sektor7.net/#!res/2021/perunsfart.md
8. https://cymulate.com/blog/blindside-a-new-technique-for-edr-evasion-with-hardware-breakpoints
9. https://rp.os3.nl/2020-2021/p68/report.pdf
10. https://signal-labs.com/analysis-of-edr-hooks-bypasses-amp-our-rust-sample/
11. https://www.secforce.com/blog/whisper2shout-unhooking-technique/
12. https://cybercoding.wordpress.com/2012/12/01/union-api/
13. https://github.com/am0nsec/HellsGate
14. https://github.com/crummie5/FreshyCalls
15. https://github.com/jthuraisamy/SysWhispers2
16. https://www.fortinet.com/blog/threat-research/prevalent-threats-targeting-cuckoo-sandbox-detection-and-our-mitigation
17. https://blog.sektor7.net/#!res/2021/halosgate.md

References

18. https://github.com/trickster0/TartarusGate
19. https://github.com/jthuraisamy/SysWhispers
20. https://github.com/rad9800/TamperingSyscalls/blob/stripped/TamperingSyscalls/entry.cpp
21. https://www.mdsec.co.uk/2020/08/firewalker-a-new-approach-to-generically-bypass-user-space-edr-hooking/
22. https://github.com/rad9800/TamperingSyscalls
23. https://malwaretech.com/2023/12/silly-edr-bypasses-and-where-to-find-them.html
24. https://malwarejake.blogspot.com/2013/07/interesting-malware-defense.html
25. https://winternl.com/memfuck/
26. https://blog.xpnsec.com/protecting-your-malware/
27. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10412066
28. https://github.com/CCob/SharpBlock
29. https://malwaretech.com/2024/02/bypassing-edrs-with-edr-preload.html
30. https://www.outflank.nl/blog/2024/10/15/introducing-early-cascade-injection-from-windows-process-creation-to-stealthy-injection/
31. https://www.deepinstinct.com/blog/evading-antivirus-detection-with-inline-hooks
32. https://www.volexity.com/wp-content/uploads/2024/08/Defcon24_EDR_Evasion_Detection_White-Paper_Andrew-Case.pdf
33. https://www.elastic.co/security-labs/finding-truth-in-the-shadows
34. https://www.vmray.com/just-carry-a-ladder-why-your-edr-let-pikabot-jump-through

Thank you!
Questions?

in/omri-misgav

TLP:CLEAR