Evading in Plain Sight: How Adversaries Beat User-Mode Protection Engines for Over A Decade

Who Am I

Omri Misgav

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

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

in/omri-misgav

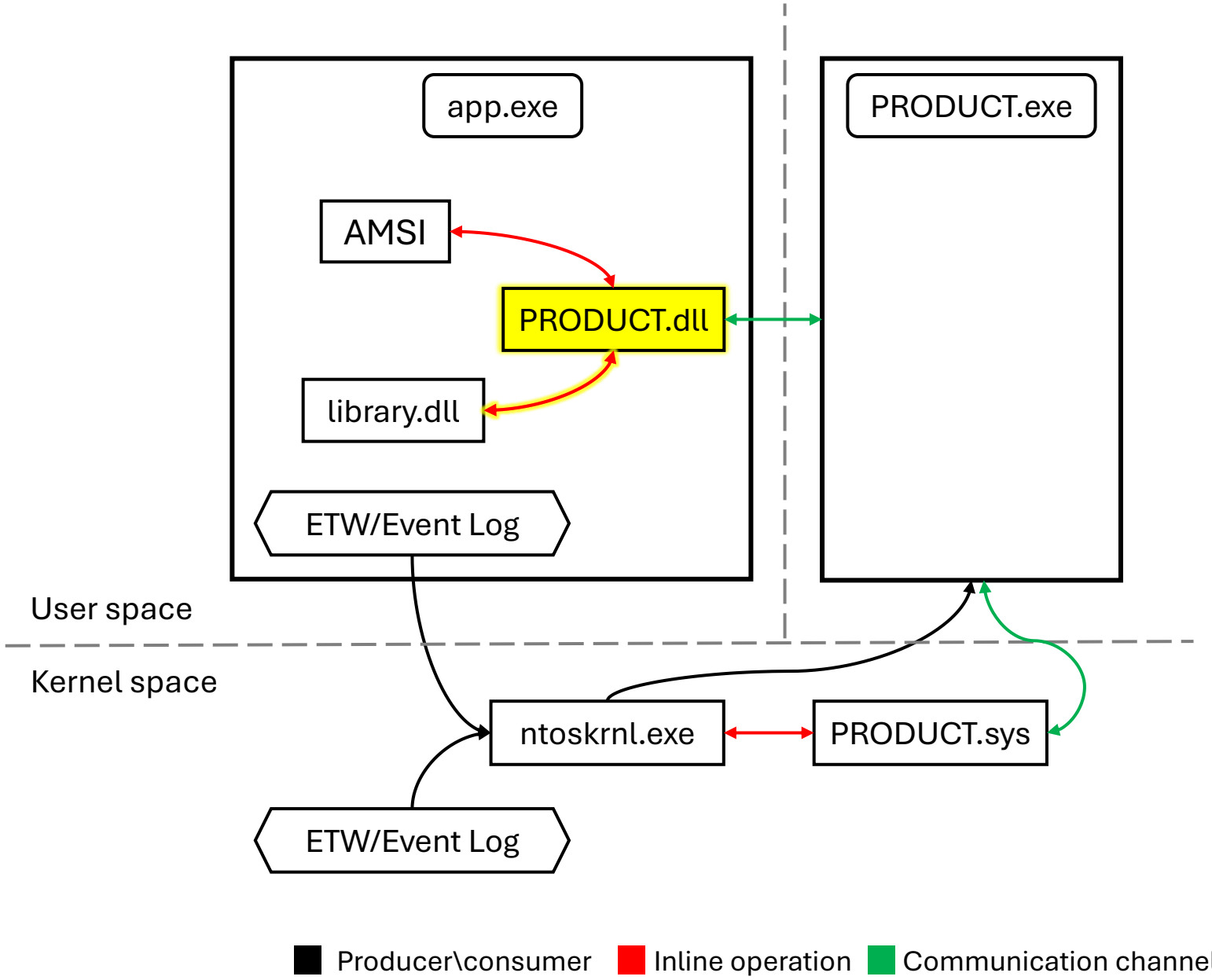
Agenda

- ❑ 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

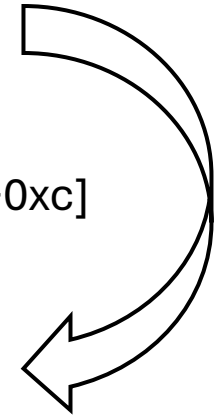
Inline hooks

- 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
```



* https://github.com/Mr-Un1k0d3r/EDRs

Agenda

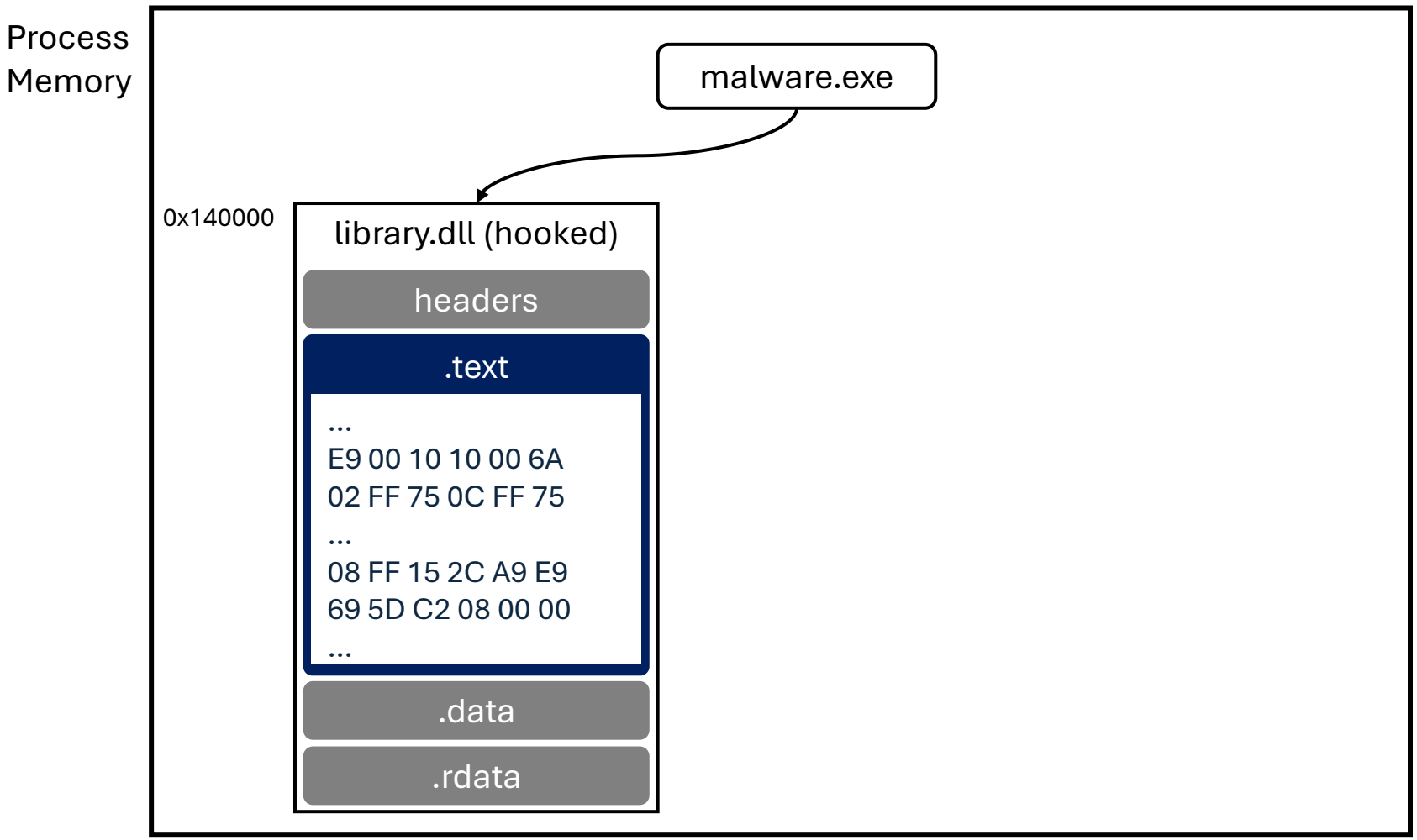
- ✓ Introduction
- ❑ Hook Evasion tactic
- ❑ Argument Forgery tactic
- ❑ Engine Disarming tactic
- ❑ Conclusions

Hook Evasion Tactic

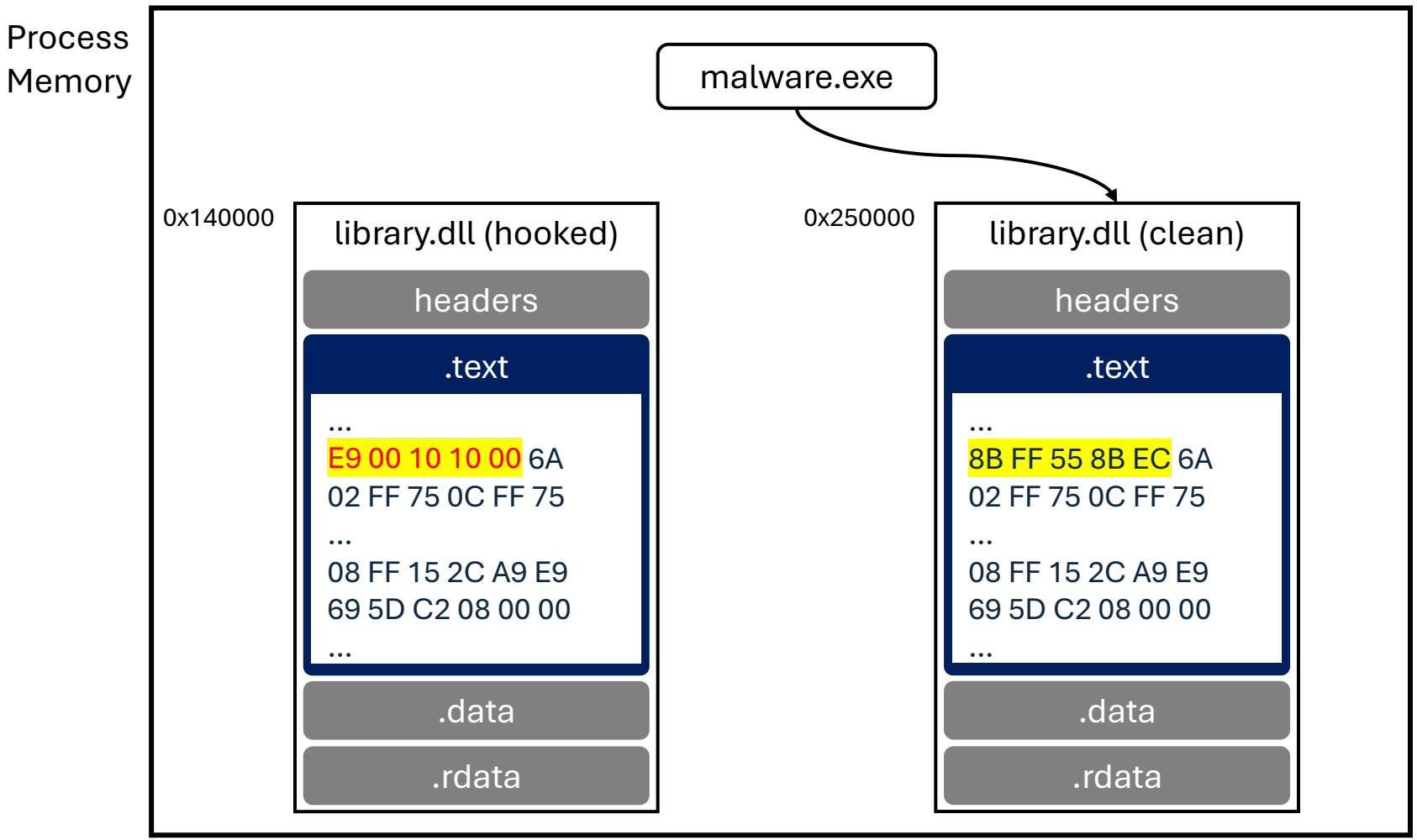
Overview

- 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



Secondary DLL Mapping



Secondary DLL Mapping

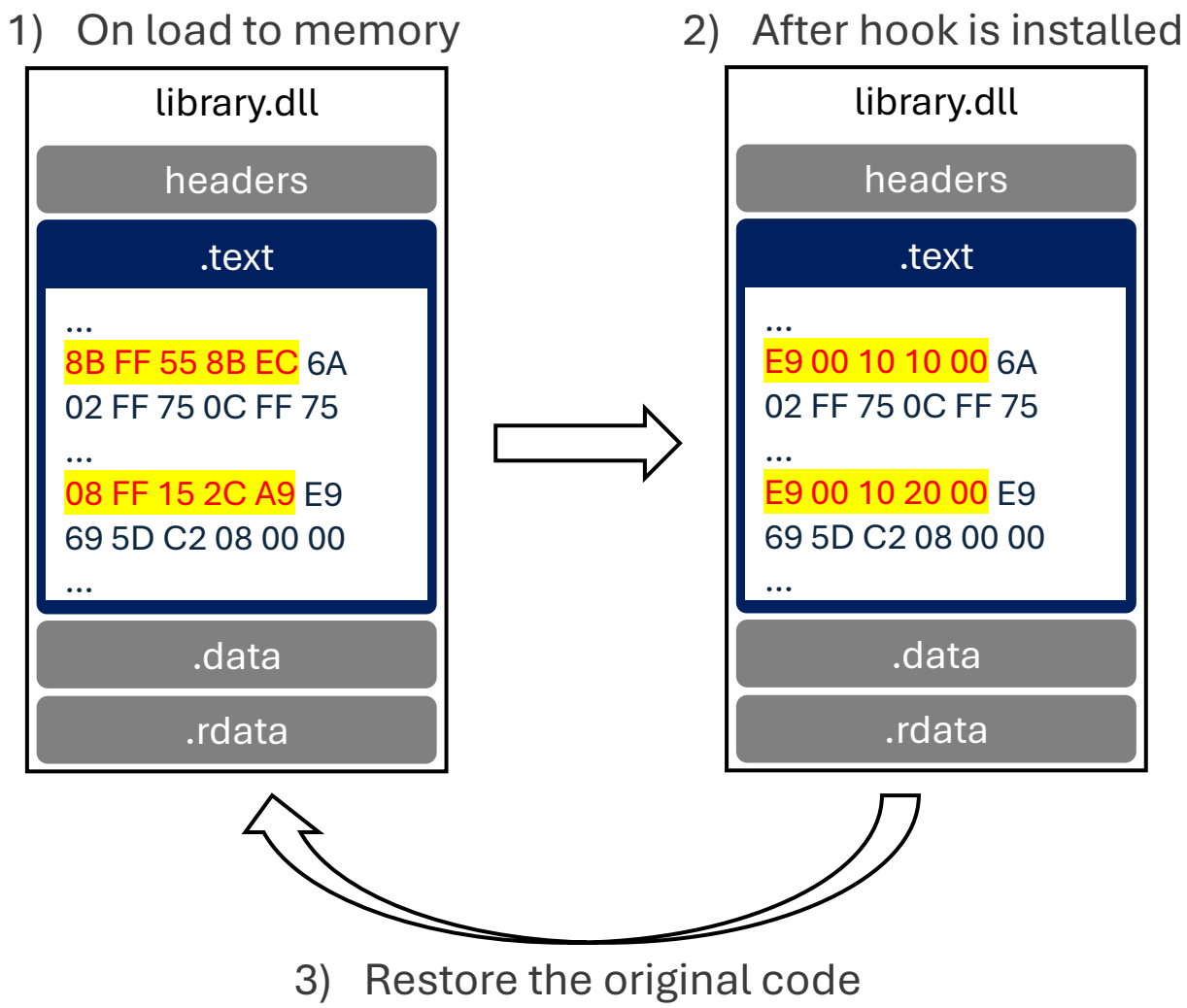
1. Manually load DLL from disk [1]
   a) ReadFile()
   b) Reflective loading
2. Clone DLL [2]
   a) CopyFile(<old_path>, <new_path>)
   b) LoadLibrary(<new_path>)
3. Section remapping [3]
   a) CreateFile() + NtCreateSection(..., SEC_IMAGE, ...) / NtOpenSection (“KnownDlls\...”)
   b) NtMapViewOfSection()
   - 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

(1/4) Temporary Copy

a) Secondary mapping provides the original code

1. Manually Load DLL From Disk (Reflective Loading) [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

(2/4) 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

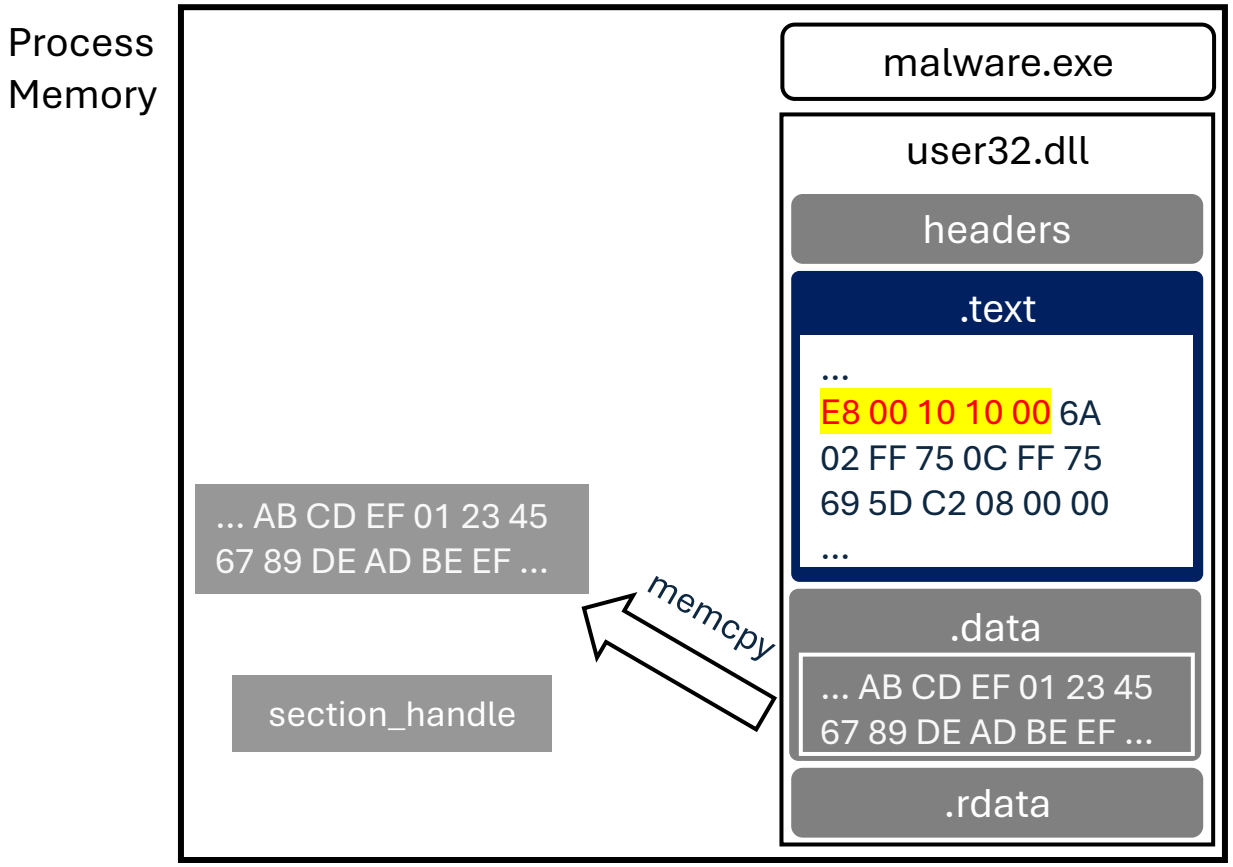
Binary Restoration

(3/4) Section Refresh [10]

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

b) Refresh

1) NtCreateSection(..., SEC_IMAGE, ...)

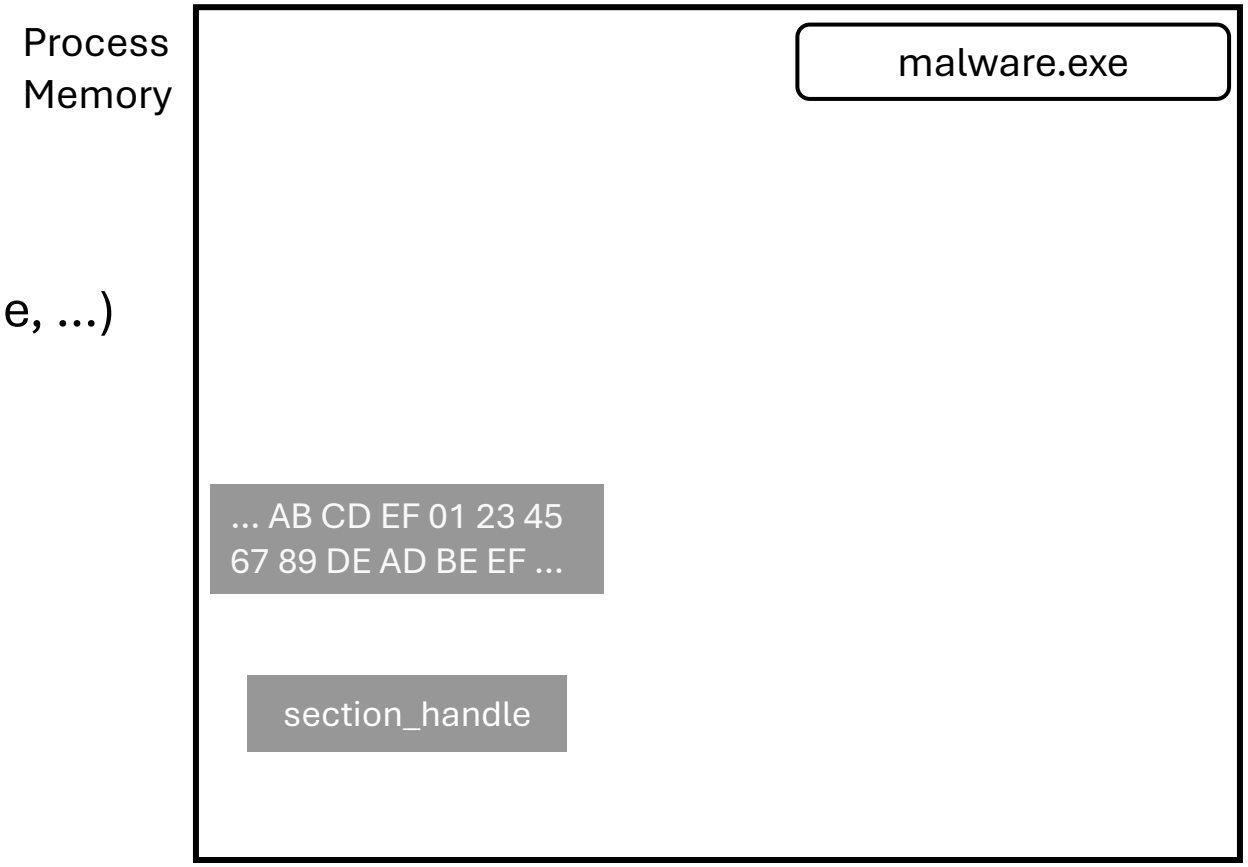


Binary Restoration

(3/4) Section Refresh [10]

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

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

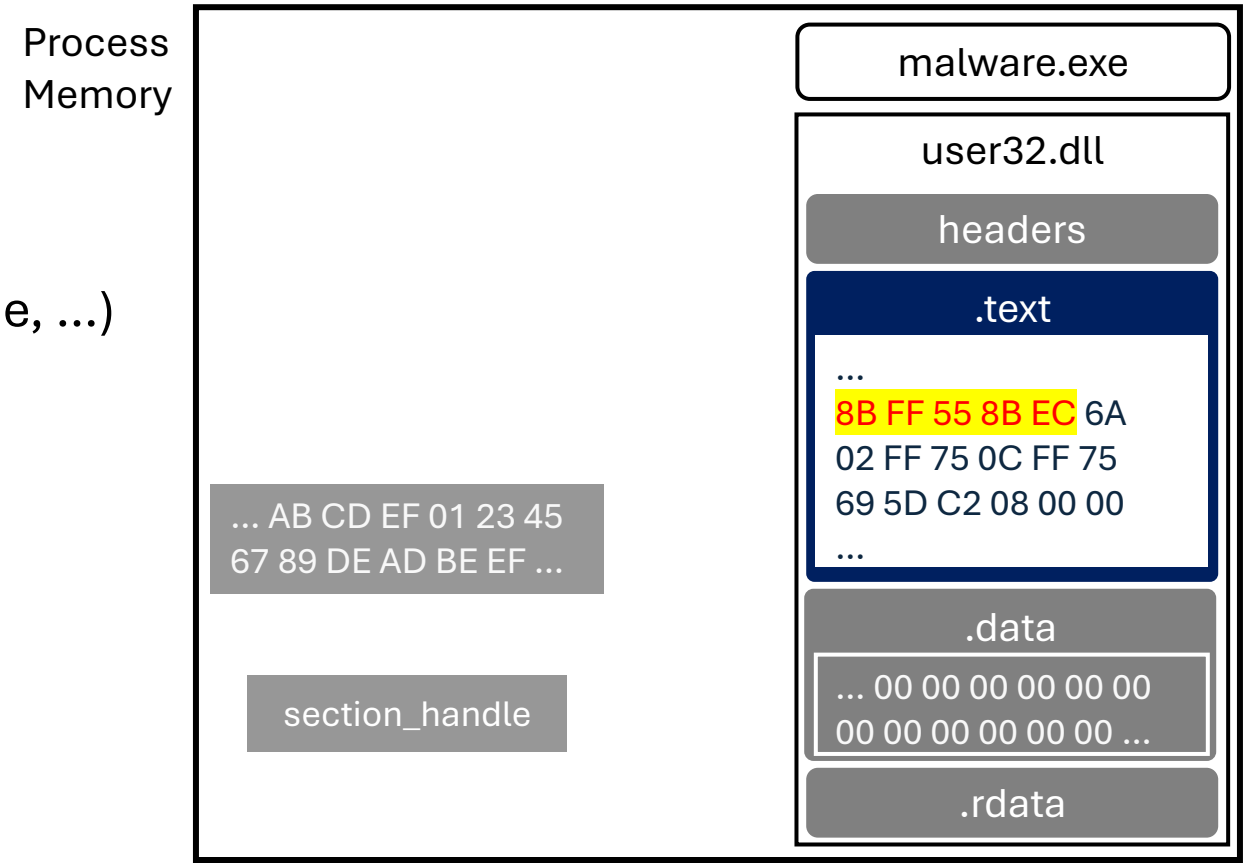


Binary Restoration

(3/4) Section Refresh [10]

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

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



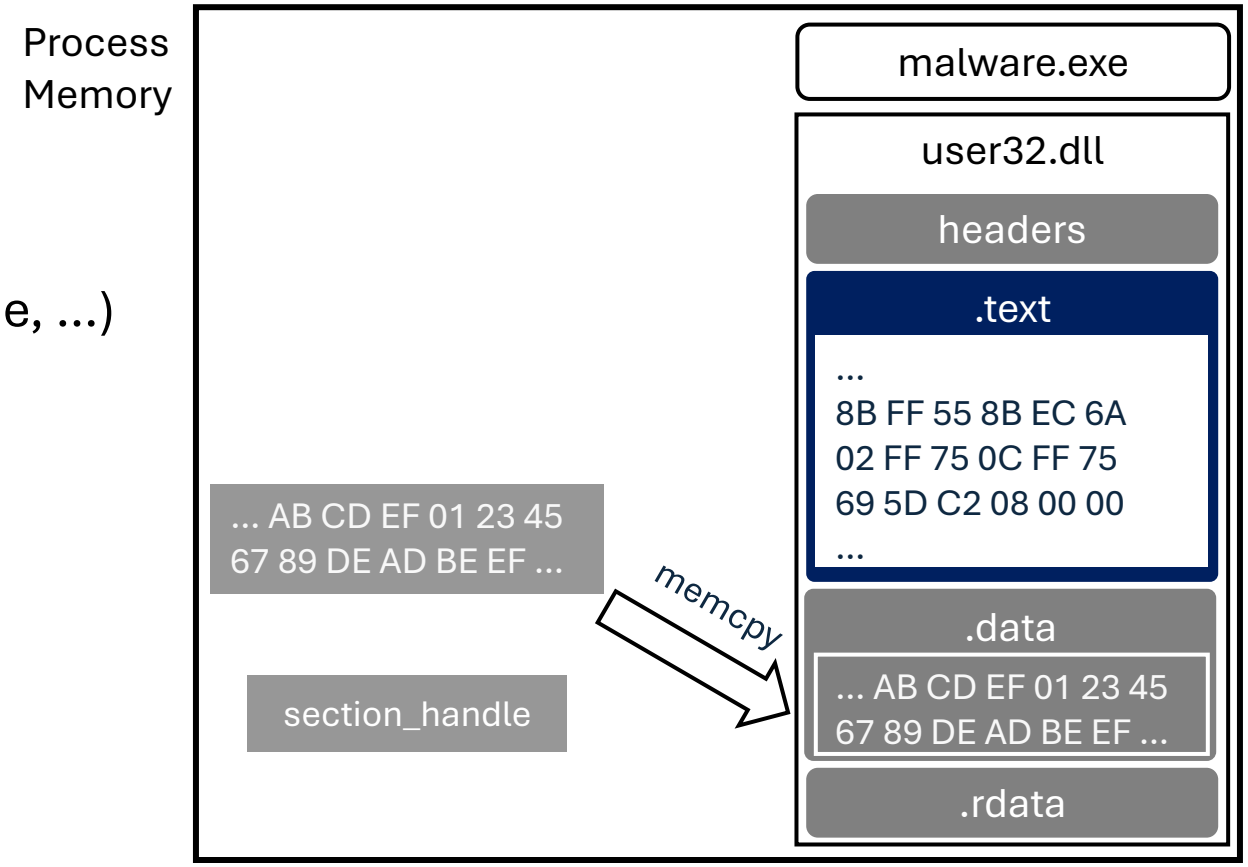
Binary Restoration

(3/4) Section Refresh [10]

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

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

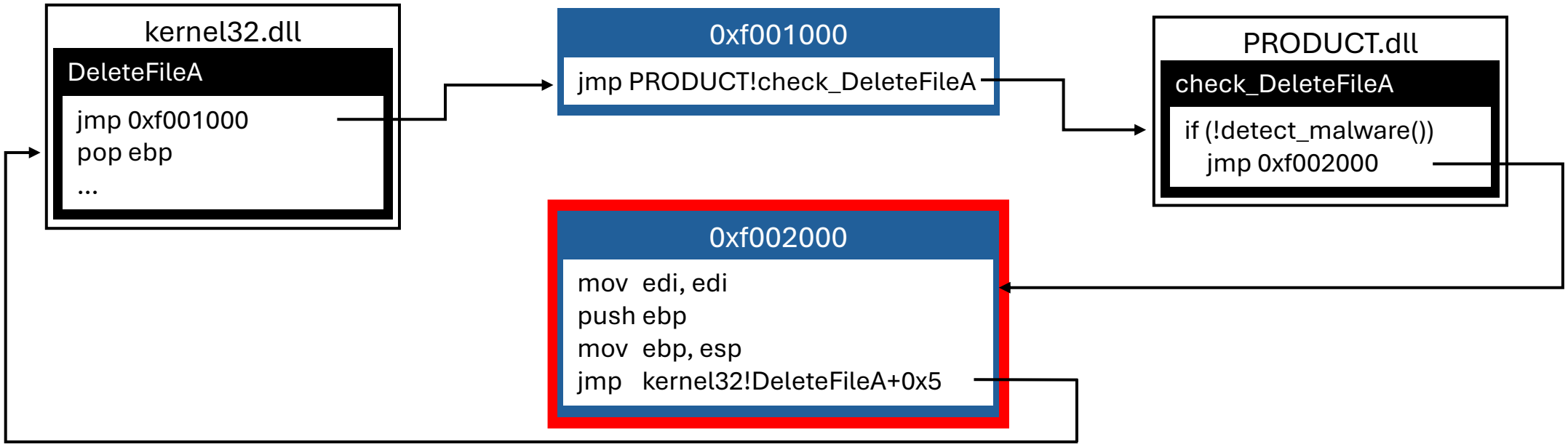
c) Restore
- IAT, forwarder exports, CFG
- Captured state



Binary Restoration

(4/4) 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

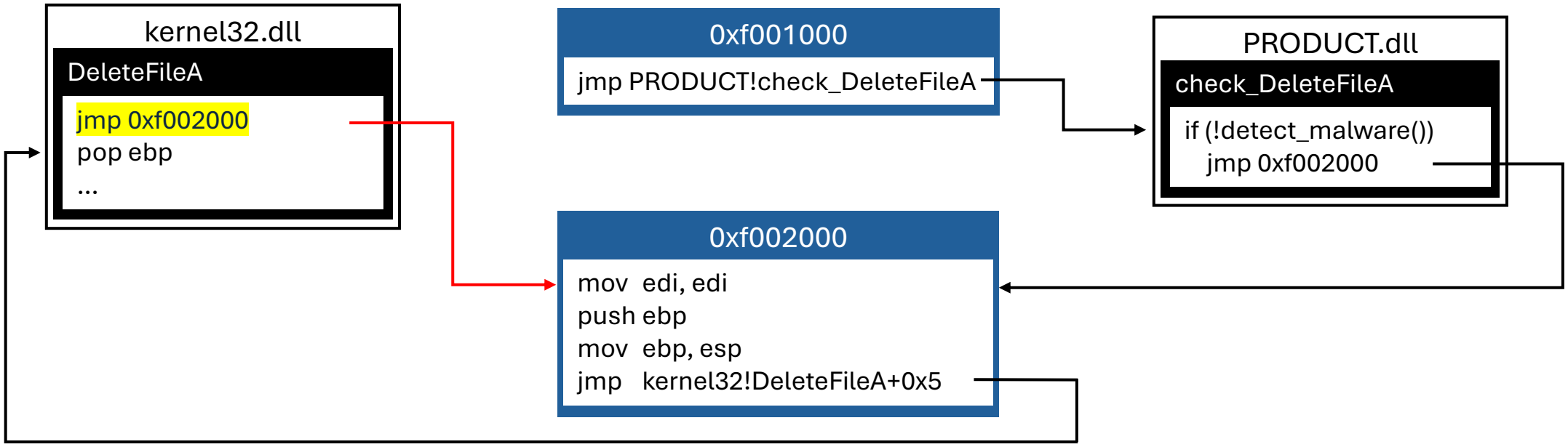


Binary Restoration

(4/4) 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 [11]

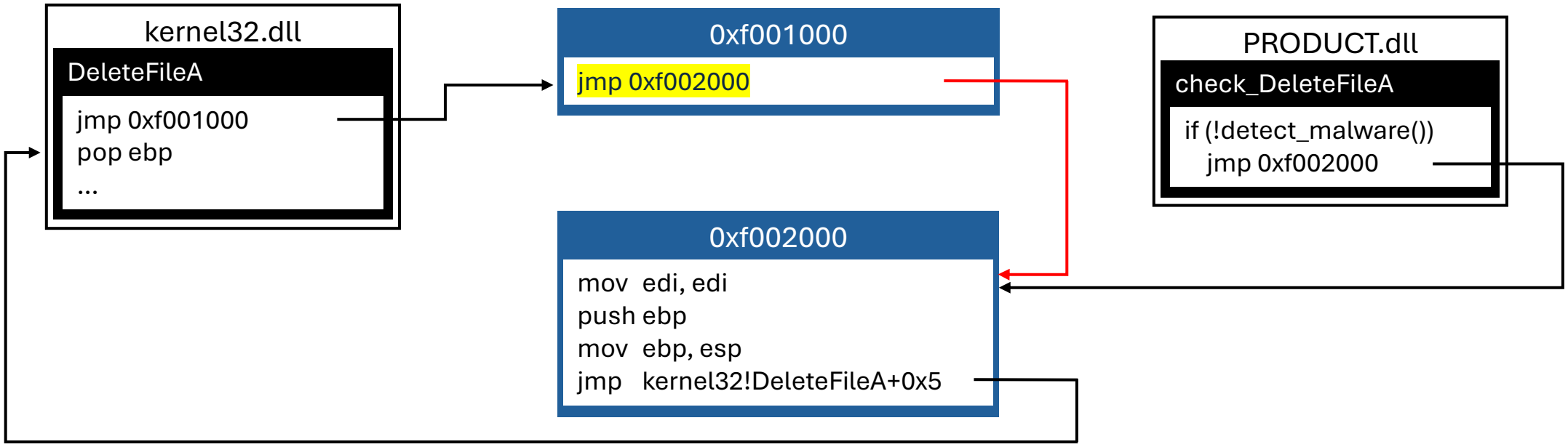


Binary Restoration

(4/4) 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 [11]
   2. Trampoline [12]

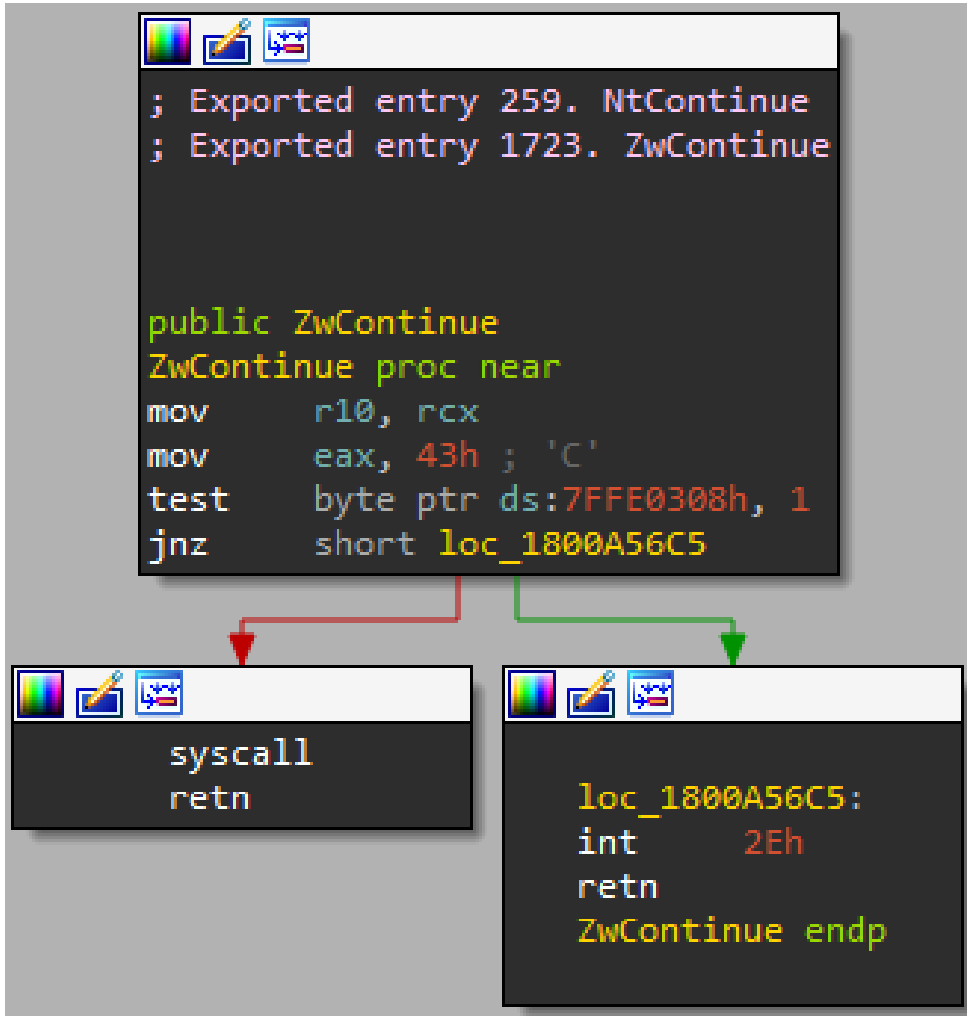


Binary Restoration

Detection and trade-offs

Technique		Runtime Indicators	Forensic Artifacts	Drawbacks
Temporary Copy	Reflective Loading		Hooks removed [13]	
	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	Hook		Hooks modified	
	Trampoline			

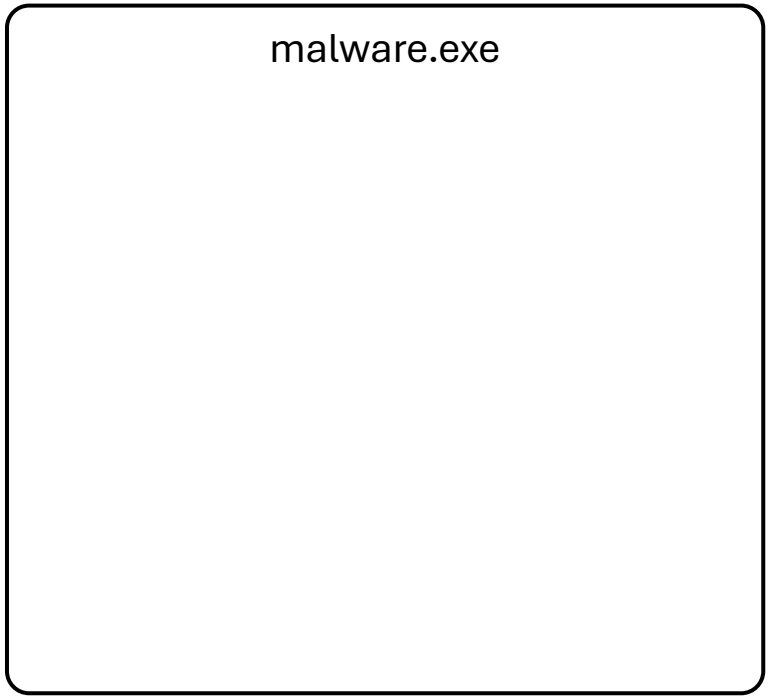
Direct System Call Invocation

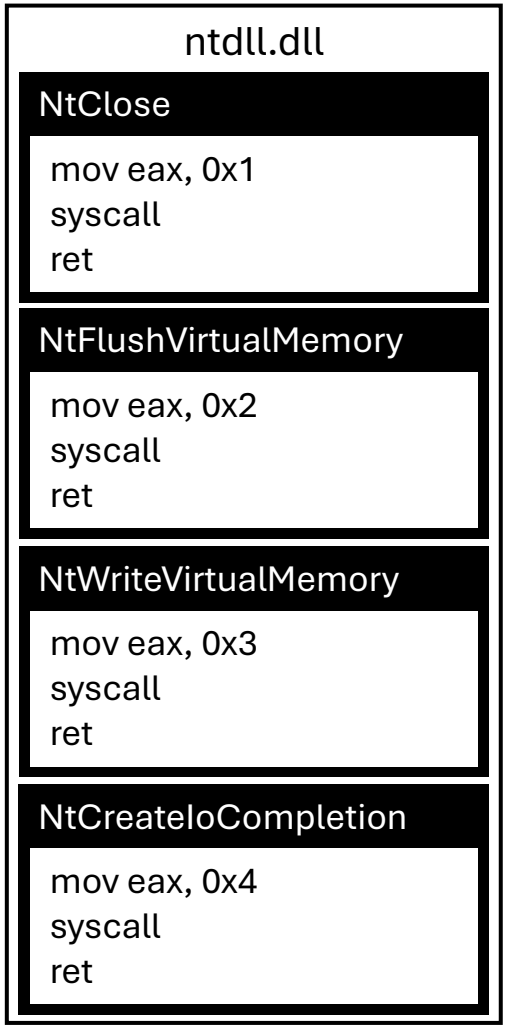


Direct System Call Invocation

(1/5) ntdll.dll Parsing

1. Instructions (pattern)

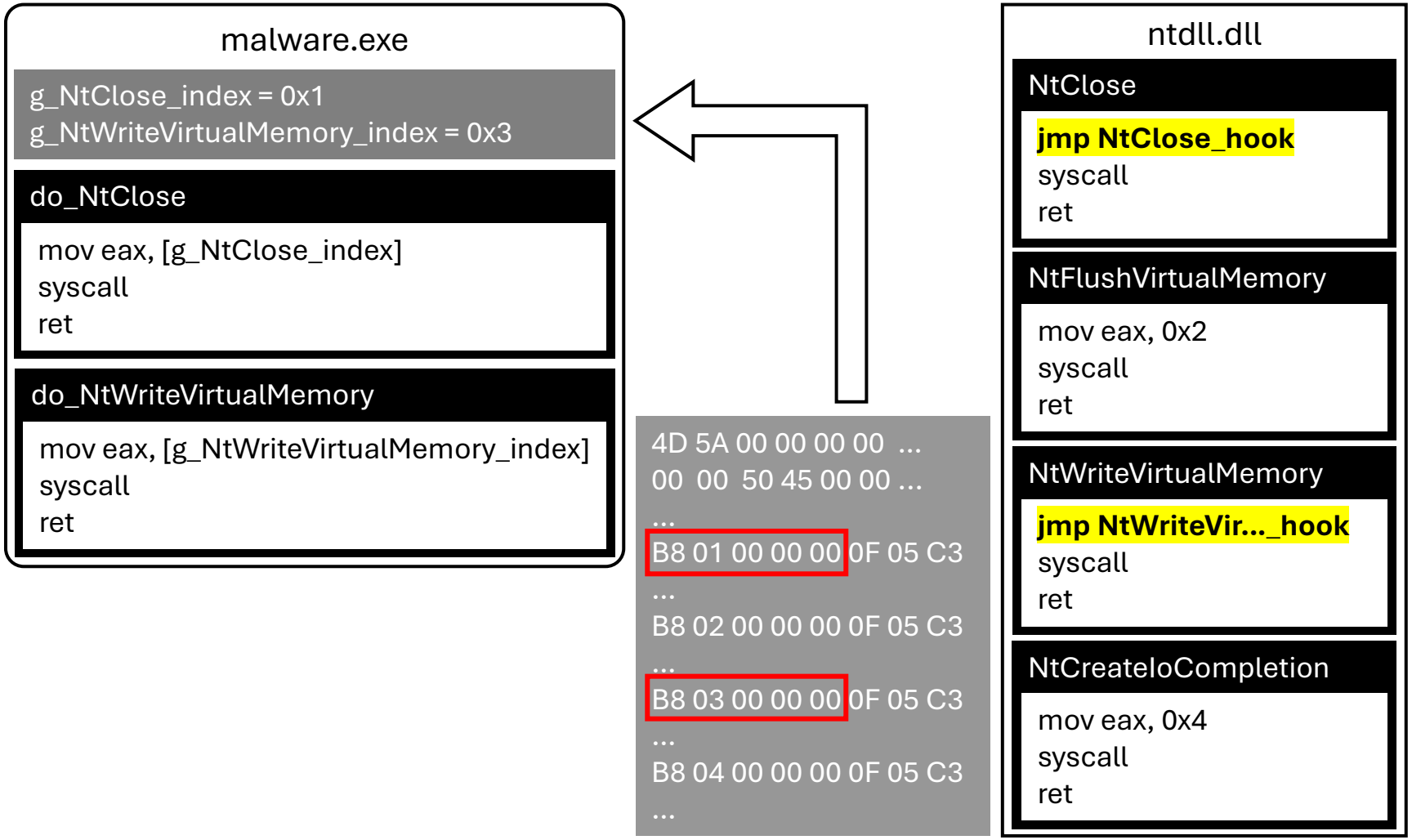




Direct System Call Invocation

(1/5) ntdll.dll Parsing

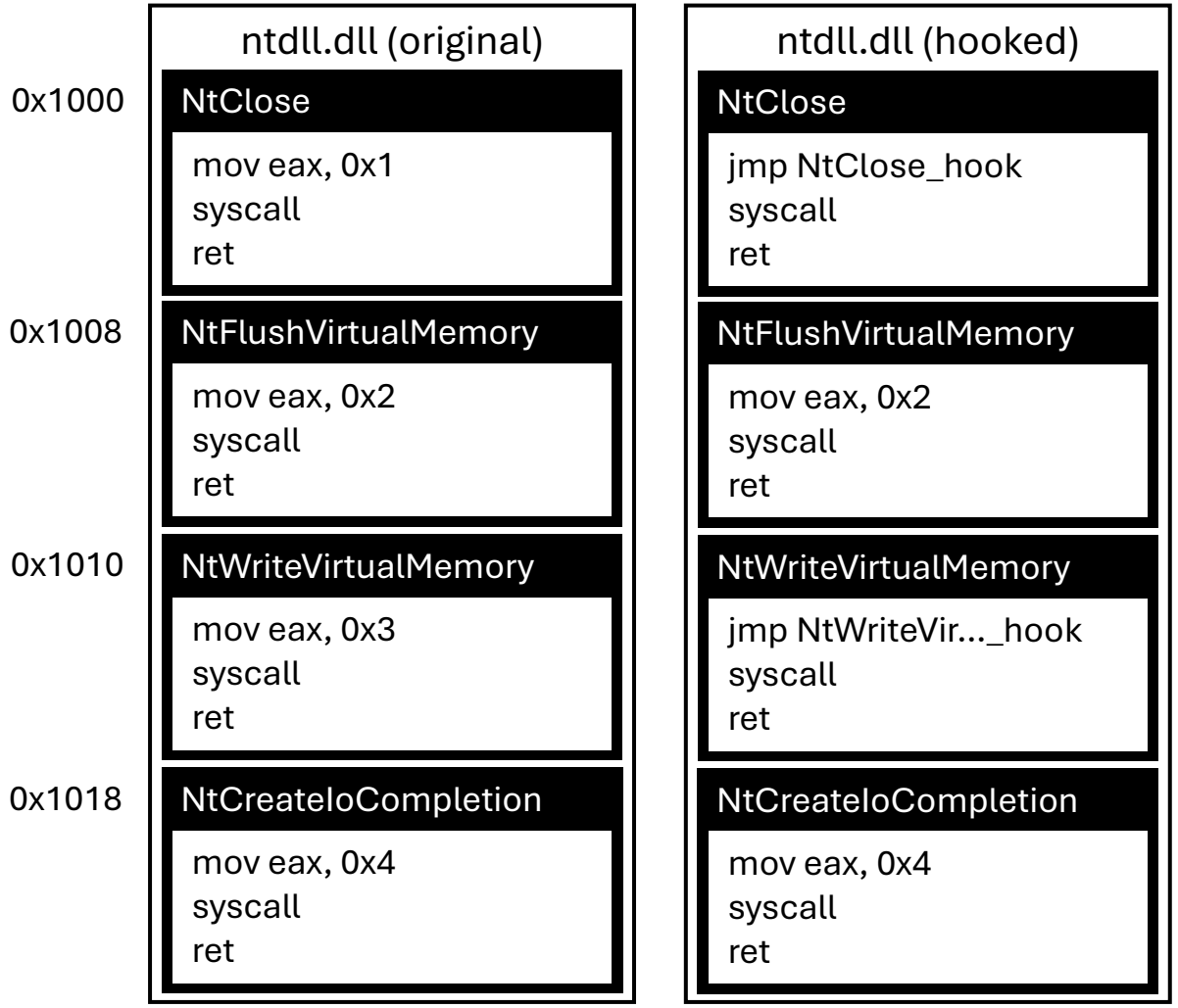
1. Instructions (pattern)
    1. From disk [14]
    2. From memory [15]



Direct System Call Invocation

(1/5) ntdll.dll Parsing

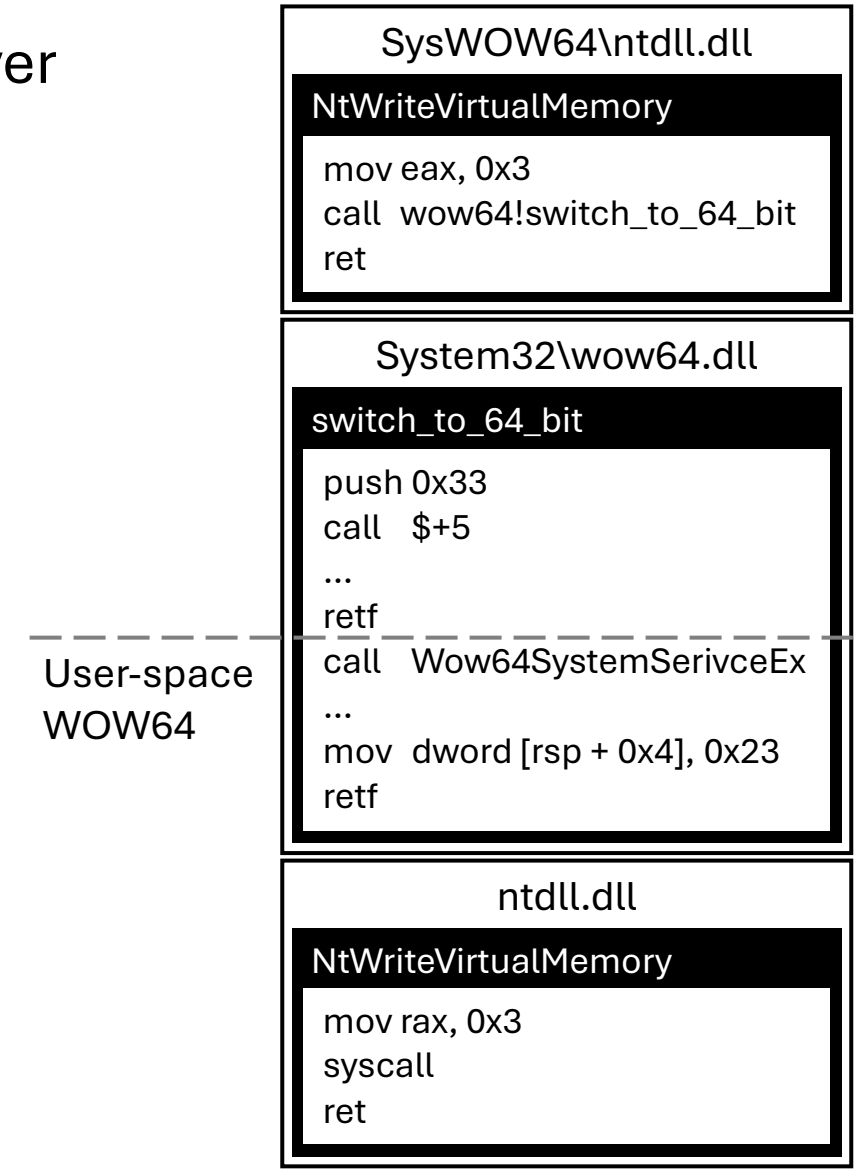
1. Instructions (pattern)
    1. From disk [14]
    2. From memory [15]
2. Count [16]
3. Proximity Check [17]



Direct System Call Invocation

(2/5) Heaven's Gate

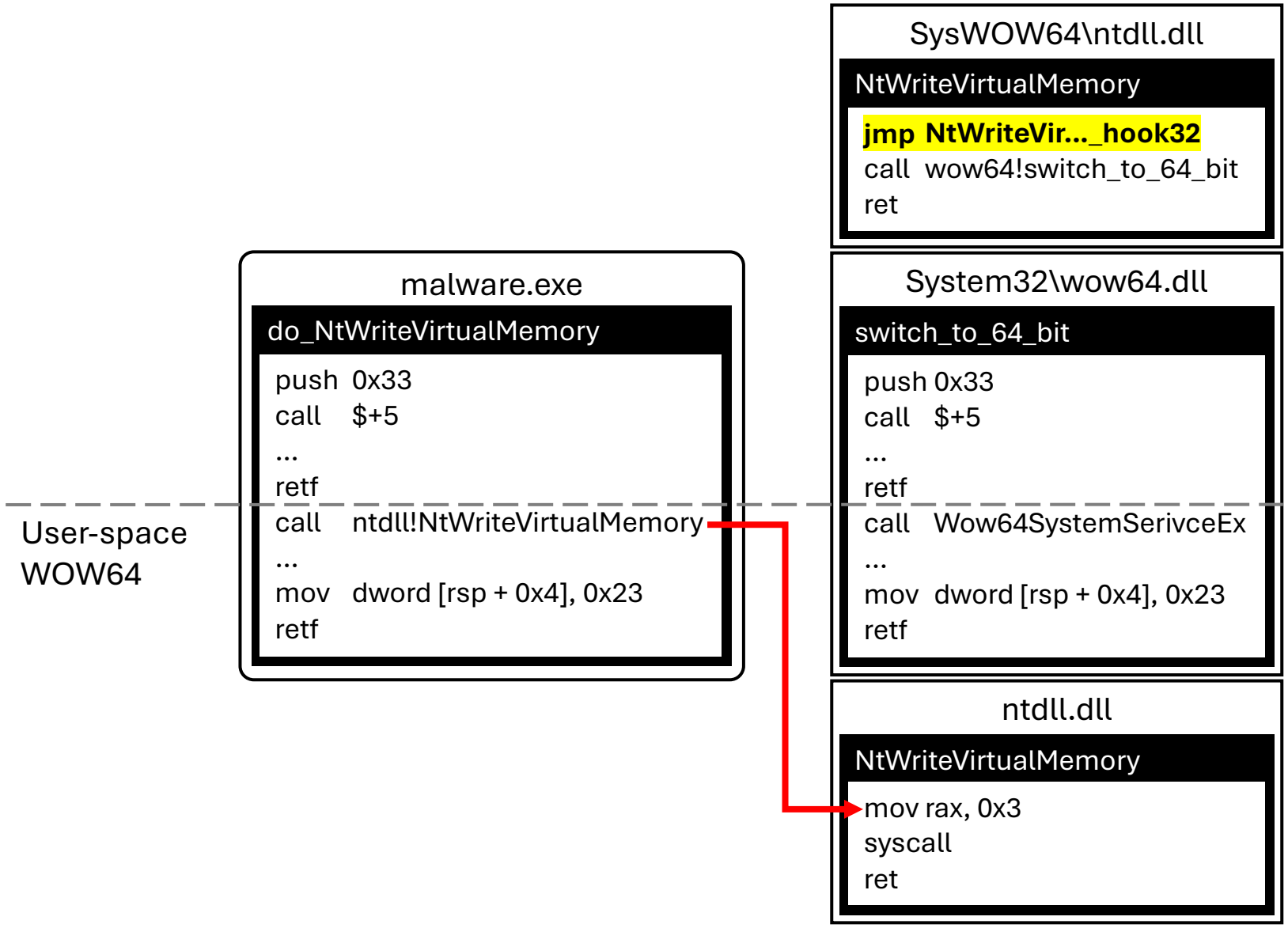
- Make system calls from within WOW64 emulation layer
  - Only 32-bit applications on 64-bit Windows



Direct System Call Invocation

(2/5) Heaven's Gate

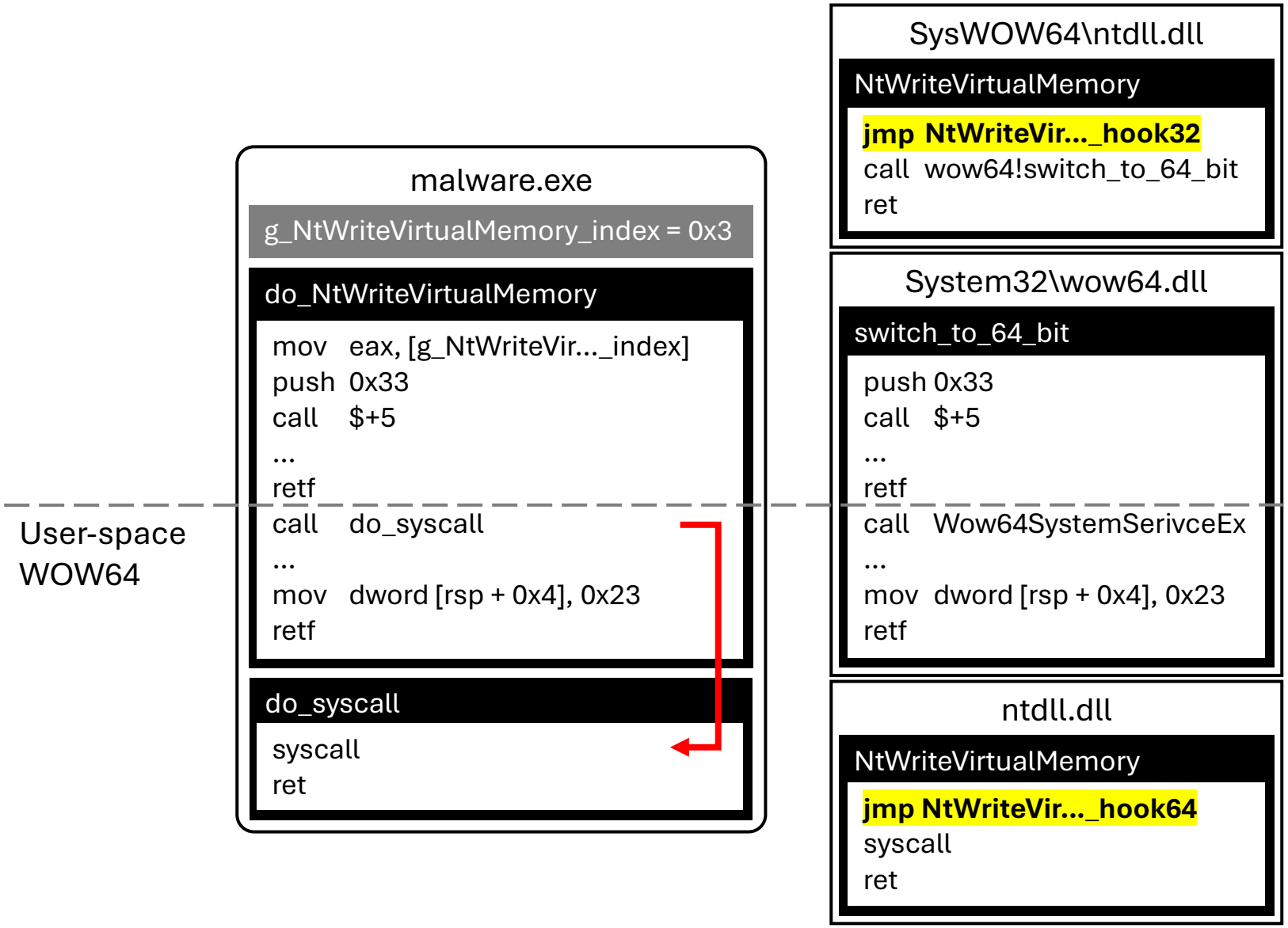
1. Skip WOW64 Layer [18]



Direct System Call Invocation

(2/5) Heaven's Gate

1. Skip WOW64 Layer [18]
2. Completely Native [19]



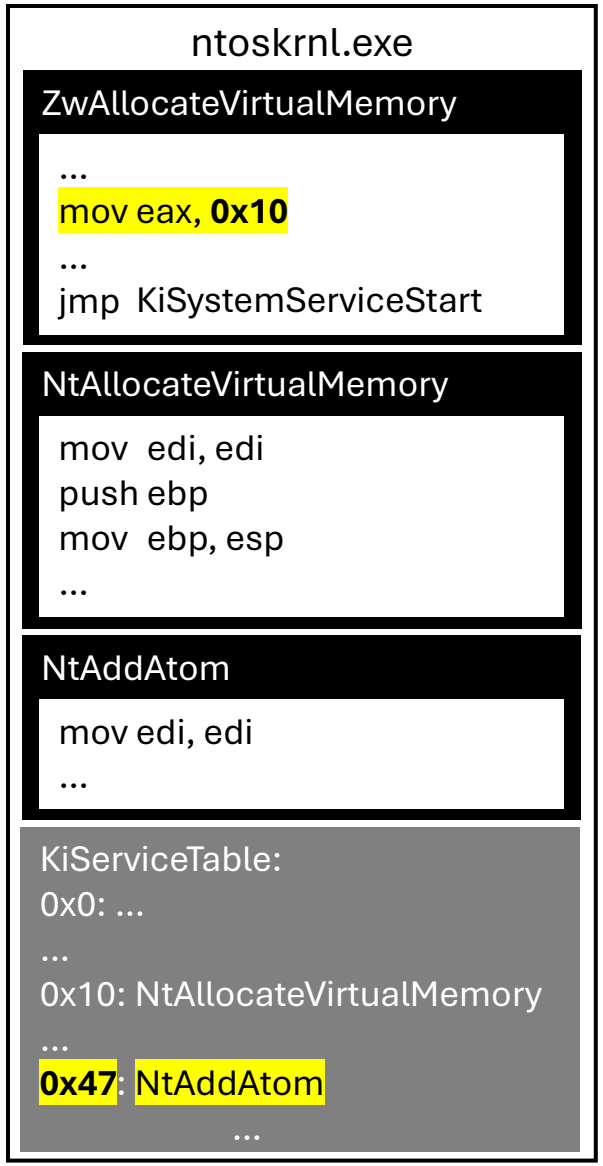
Direct System Call Invocation

(3/5) ntoskrnl.exe Parsing [20]

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

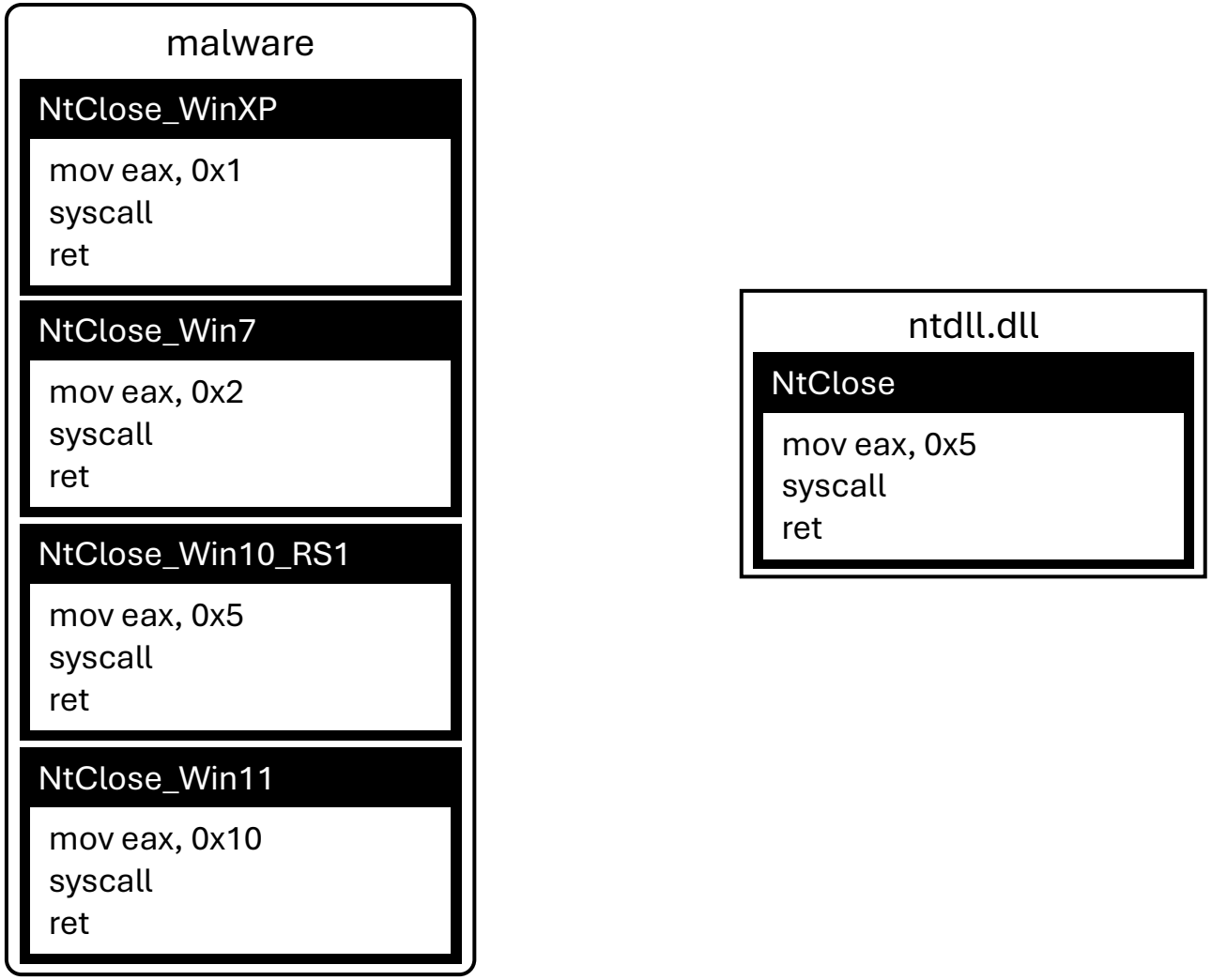
b) Some functions only have the Nt version exported
   1. GetProcAddress(ntoskrnl, “Nt…”)
   2. Locate ntoskrnl!KiServiceTable
   3. Traverse the table to find reference to the target

- 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

(4/5) Bring Your Own Index (BYOI) [21]



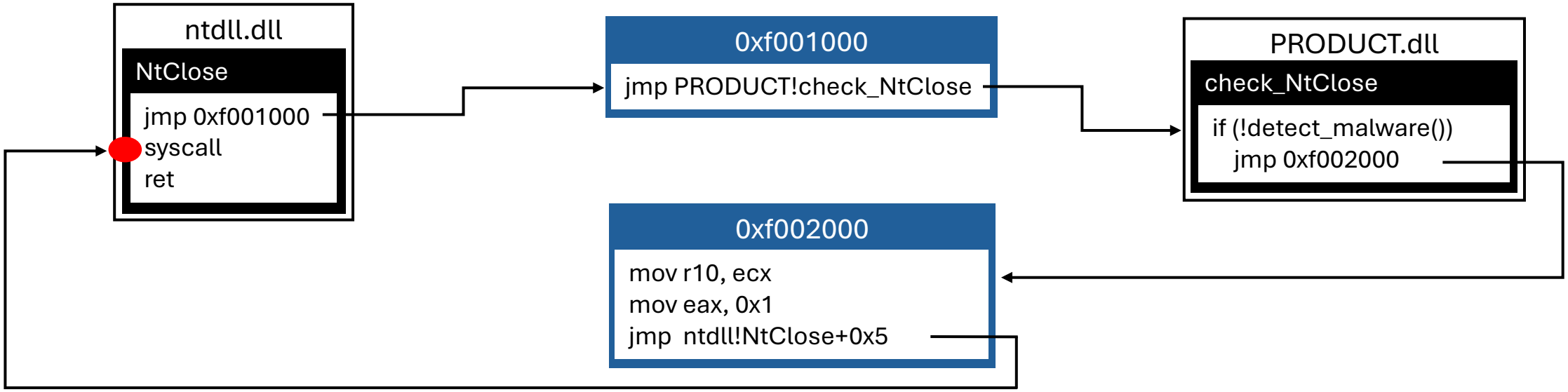
Direct System Call Invocation

(5/5) Dynamic Resolution

a) SetUnhandledExceptionFilter [22] / RtlAddVectoredExceptionHandler

b) Place a breakpoint on the syscall instruction

c) Call the function



Direct System Call Invocation

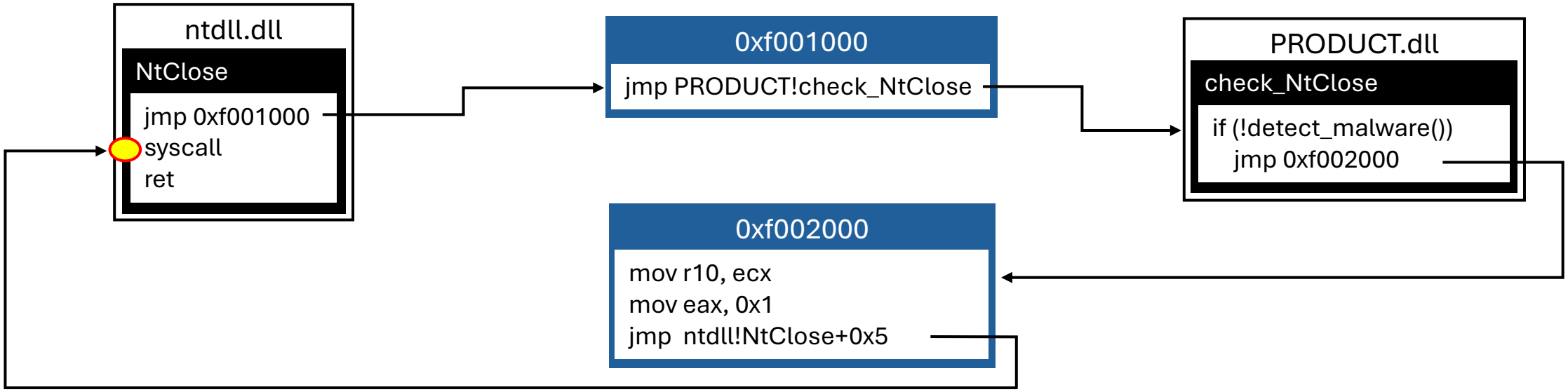
(5/5) Dynamic Resolution

a) SetUnhandledExceptionFilter [22] / RtlAddVectoredExceptionHandler

b) Place a breakpoint on the syscall instruction

c) Call the function

d) Get eax 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 Artifact [13]</th><th>Drawbacks</th></tr>
<tr><td colspan="2">ntdll.dll Parsing</td><td rowspan="4">Call stacks missing ntdll.dll</td><td rowspan="4"></td><td rowspan="4">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><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>Native</td><td>Call stacks missing WOW64 system DLLs and native 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

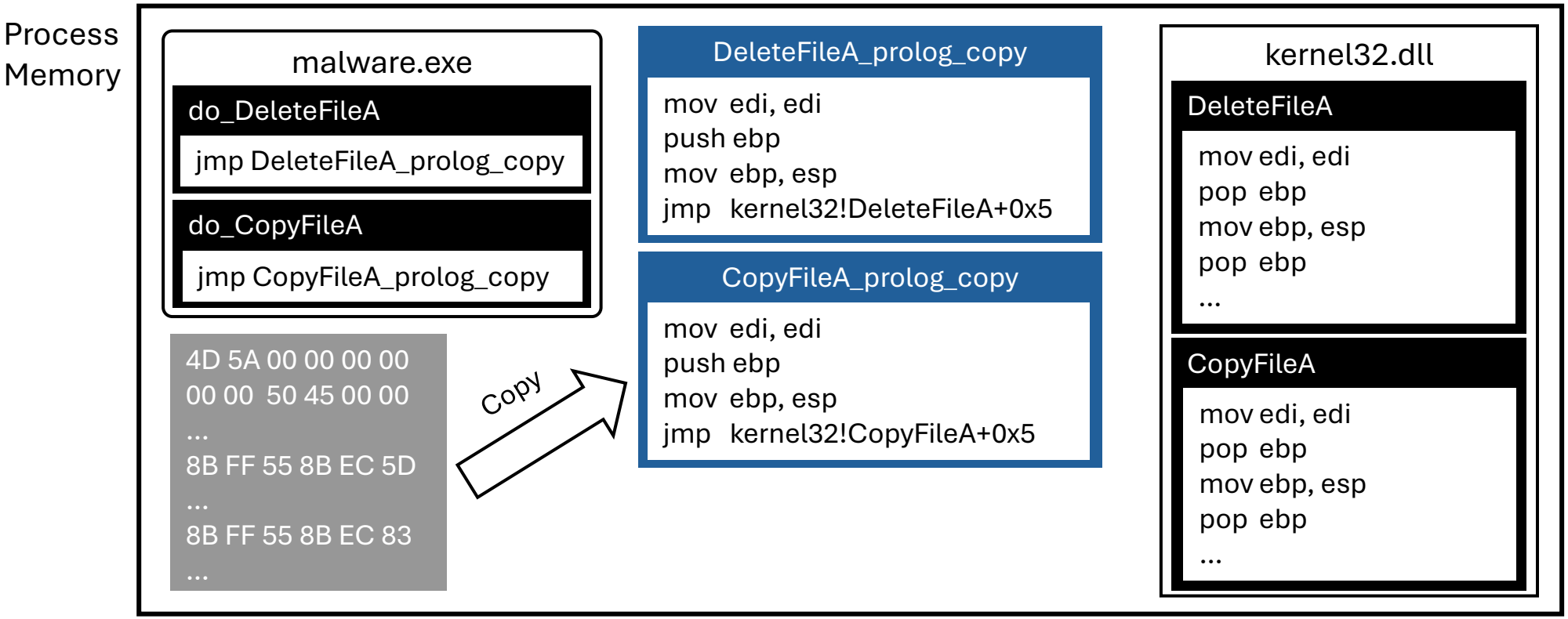
- Rebuild function stubs elsewhere
- Used in the past by packers and anti-cheat solutions

Code Splicing

(1/2) From Disk [23]

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

b) Extract the target function’s instructions

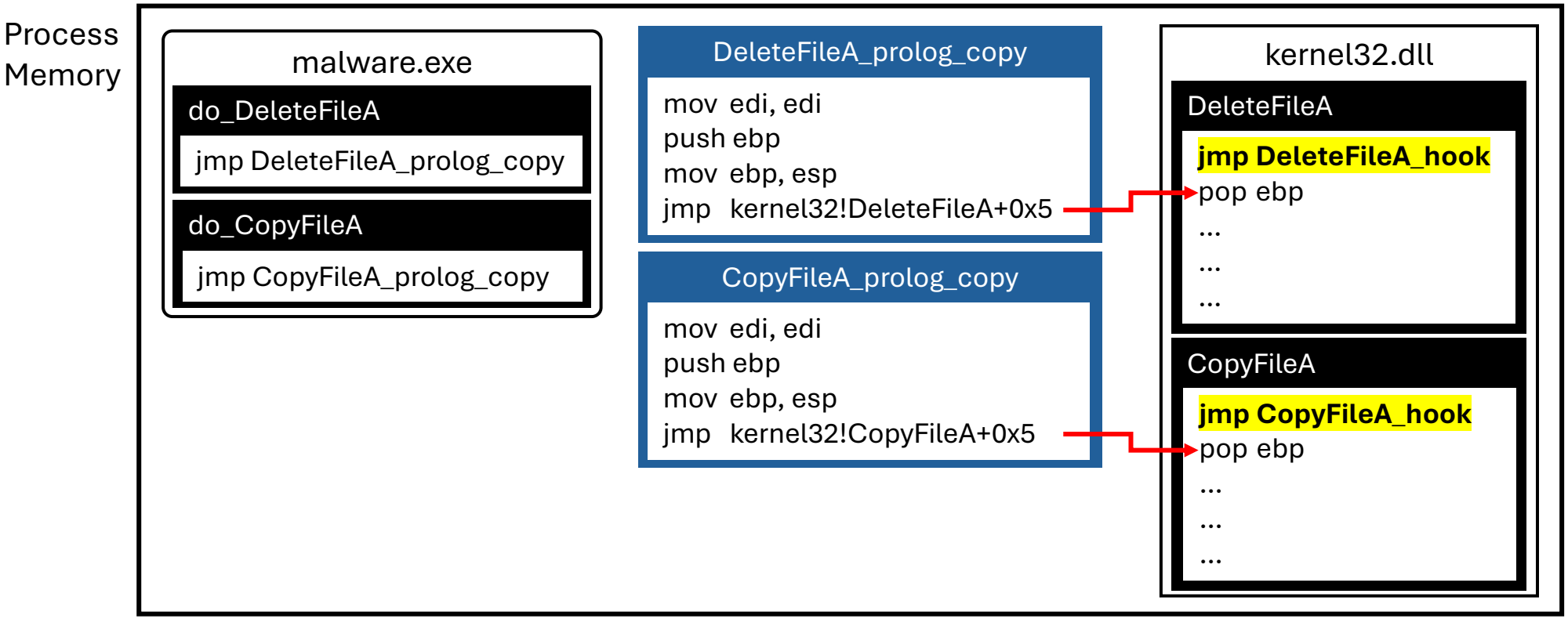


Code Splicing

(1/2) From Disk [23]

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

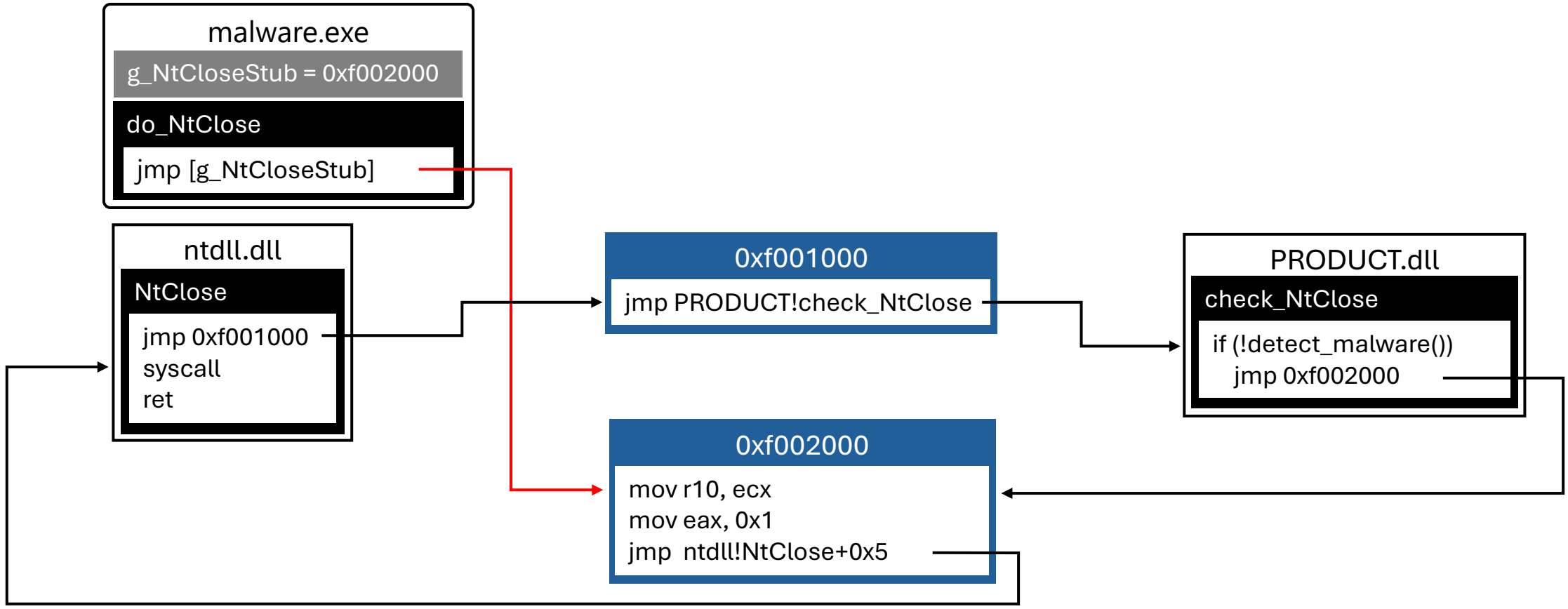
b) Extract the target function’s instructions



Code Splicing

(2/2) From Memory / Stub Reuse [24]

- The stubs are already in memory and it's possible to find them (recall Short-circuiting?)
  - MEM_COMMIT | MEM_PRIVATE | PAGE_EXECUTE
  - The 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
From Memory / Stub Reuse			

Agenda

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

Argument Forgery Tactic

- 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 [25] (like Dynamic Resolution)
  2. Argument that causes the protection engine to trigger an exception [26]
     - Vendor-specific so not a reliable option

Argument Forgery Tactic

Implementation

a) SetUnhandledExceptionFilter [25] / 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 [25] / 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 [25] / 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 [25] / 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
   a) Registers in the CONTEXT structure
   b) 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 [25] / 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
   a) Registers in the CONTEXT structure
   b) 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"
```

Engine Disarming Tactic

1. FreeLibrary [27] or trigger the engine's unload function (vendor-specific)
2. Unmap all DLLs [28] (like "Section Refresh" just without the "Refresh")
3. Allow to load only MS-signed binaries (using process mitigation policies) [29,30]
4. Preloading in new child process
   - Debugged process [31] – break on image load and switch DLL entrypoint to do nothing on load
   - Inject the process – use AppVerifier [32] and Shim Engine [33] callbacks to run before the protection engine and prevent its DLLs from loading into it

- Other vendor-specific implementation issues (e.g. toggle hook disabled flag [34])

Agenda

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

Conclusions

- 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 methods went up from 12 to 20 in last 5 years
  - Generic engine disarming count grew from 2 to 5 in the same time
- MITRE ATT&CK T1562.001 (Impair Defenses: Disable or Modify Tools) briefly mentions unhooking starting from v10 (October 2021)
  - Only “Binary Restoration” or “Engine Disarming”
- Prerequisite for other evasions (as code execution is first required)
  - ETW and AMSI bypasses
  - Removing EDRs’ kernel callbacks
- Call stack spoofing can be chained to thwart detection
  - Leverage CET shadow stack [35] or transition-based code tracing (when controlling the hardware) [36]

Conclusions (cont.)

- User-mode code can still cause stability issues (i.e. “Critical Process”*)
- Using user-mode hooks for security is insufficient
  - Underlying flaw – the reliance on the same execution environment that is intended to be protected

- Check out the whitepaper for more –
  - Listing of malware families
  - References to open-source projects
  - Details on detection

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

References

1. https://www.fireeye.com/blog/threat-research/2017/10/formbook-malware-distribution-campaigns.html
2. https://blog.malwarebytes.com/threat-analysis/2018/03/hancitor-fileless-attack-with-a-copy-trick/
3. https://blog.malwarebytes.com/threat-analysis/2018/08/process-doppelganging-meets-process-hollowing_osiris/
4. https://breakdev.org/defeating-antivirus-real-time-protection-from-the-inside/
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://www.youtube.com/watch?v=Zk_8nQJeOQg&pp=ygUJYnNpZGVzdGx2&t=1080s
11. https://signal-labs.com/analysis-of-edr-hooks-bypasses-amp-our-rust-sample/
12. https://www.secforce.com/blog/whisper2shout-unhooking-technique/
13. https://www.volexity.com/wp-content/uploads/2024/08/Defcon24_EDR_Evasion_Detection_White-Paper_Andrew-Case.pdf
14. https://cybercoding.wordpress.com/2012/12/01/union-api/
15. https://github.com/am0nsec/HellsGate
16. https://github.com/crummie5/FreshyCalls
17. https://blog.sektor7.net/#!res/2021/halosgate.md
18. https://www.iij.ad.jp/en/dev/iir/pdf/iir_vol34_EN.pdf
19. https://web.archive.org/web/20190407064851/https://blog.ensilo.com/globeimposter-ransomware-technical

References

20. https://www.youtube.com/watch?v=Zk_8nQJeOQg&pp=ygUJYnNpZGVzdGx2&t=855s
21. https://secrary.com/posts/bypassuserhooks/
22. https://github.com/rad9800/TamperingSyscalls/blob/stripped/TamperingSyscalls/entry.cpp
23. http://web.archive.org/web/20180628084559/https://security.radware.com/malware/codefork-malware/
24. https://www.mdsec.co.uk/2020/08/firewalker-a-new-approach-to-generically-bypass-user-space-edr-hooking/
25. https://github.com/rad9800/TamperingSyscalls
26. https://malwaretech.com/2023/12/silly-edr-bypasses-and-where-to-find-them.html
27. https://malwarejake.blogspot.com/2013/07/interesting-malware-defense.html
28. https://winternl.com/memfuck/
29. https://blog.xpnsec.com/protecting-your-malware/
30. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10412066
31. https://github.com/CCob/SharpBlock
32. https://malwaretech.com/2024/02/bypassing-edrs-with-edr-preload.html
33. https://www.outflank.nl/blog/2024/10/15/introducing-early-cascade-injection-from-windows-process-creation-to-stealthy-injection/
34. https://www.deepinstinct.com/blog/evading-antivirus-detection-with-inline-hooks
35. https://www.elastic.co/security-labs/finding-truth-in-the-shadows
36. https://www.vmray.com/just-carry-a-ladder-why-your-edr-let-pikabot-jump-through

Questions?

in/omri-misgav

Thank you!

in/omri-misgav