KEYSIGHT

Monitoring malware behavior through kernel syscall tracing

Cristian Done
Mihai Vasilescu

Agenda

- About Us
- Motivation
- Windows API Call Example
- Malware User Space API Call Tracing
- Kernel Space System Call Tracing
- Windows Kernel System Call Handling
- SSDT Hooking
- System Call Number Mapping
- Bypassing Protections
- Summary
- Results

About Us

- Security research engineers
- Sandboxing / exploits and tinkering all around
- Anything network related
- @_mihaiv_
- @cristi_done

Motivation

- An undetected malware is a happy malware
- Many options to hide API calls from userland
- How can we catch this in sandboxes?

Windows API Call Example

- HELLO WORLD

```rust
use std::fs::File;
use std::io::prelude::*;

fn main() -> std::io::Result<()> {
    let mut file = File::create("foo.txt")?;
    file.write_all(b"Hello, world!")?;
    Ok(())
}
```

Windows API Call Example

CreateFile

```
18  v14 = -2i64;
19  v0 = std::fs::File::create::h2ff97f231ad7c4b1(
20          (enum2$<core::result::Result<std::fs::File,std::io::error::Error> > *)"foo.txtsrc\\main.rs",
21          (ref$<str$> *)7);
```

```
26  return (enum2$<core::result::Result<std::fs::File,std::io::error::Error> > *)std::fs::OpenOptions::_open::hc7e6bd581cc8a799(
27                                                                                v7,
28                                                                                v3,
29                                                                                v4);
30  }
```

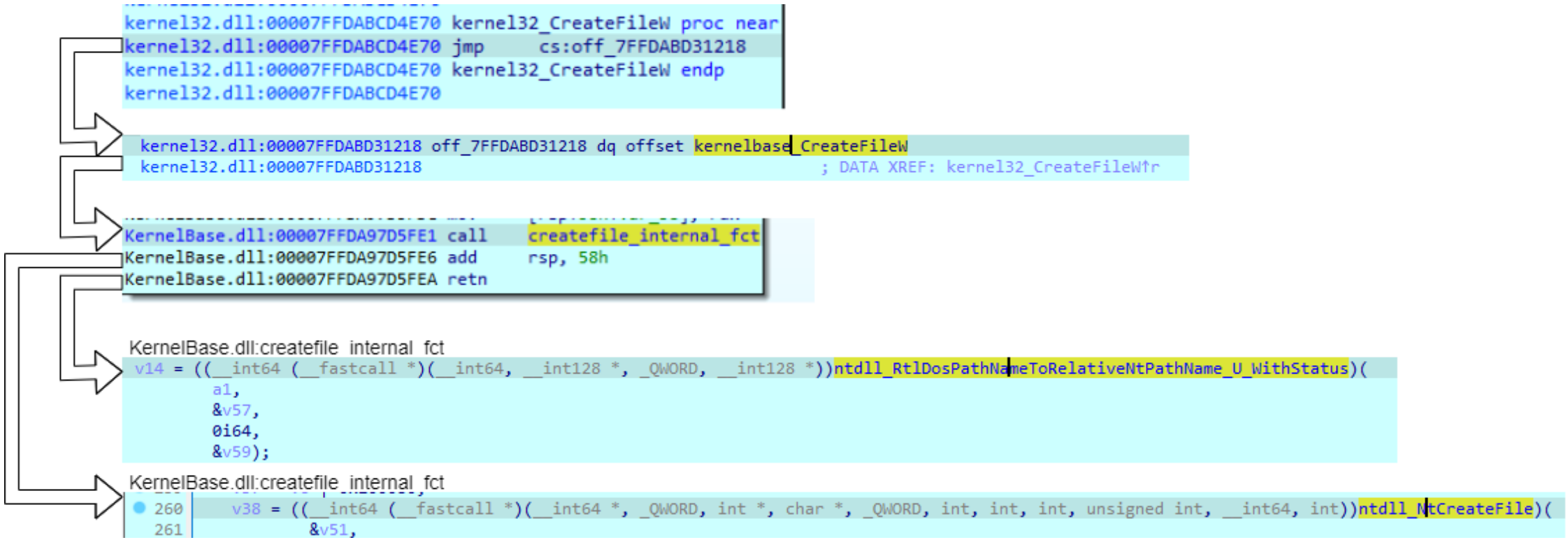
```
1   __int64 __fastcall std::fs::OpenOptions::_open::hc7e6bd581cc8a799(__int64 a1, __int64 a2, __int64 a3)
2   {
3     return std::sys::windows::fs::File::open(a2, a3, a1);
4   }
```

```
131 if ( CreateFileW(
132         lpFileName,
133         v11,
134         v12,
135         v5,
136         dwCreationDisposition,
137         ((v13 != 0) << 21) | *(_DWORD *)(a3 + 28) | (unsigned int)(*(_DWORD *)(a3 + 16) | *(_DWORD *)(a3 + 20)),
138         0i64) == (HANDLE)-1i64 )
```

Address	Ordinal	Name	Library
000000014001F0D8		CreateFileW	KERNEL32

Windows API Call Example

CreateFile



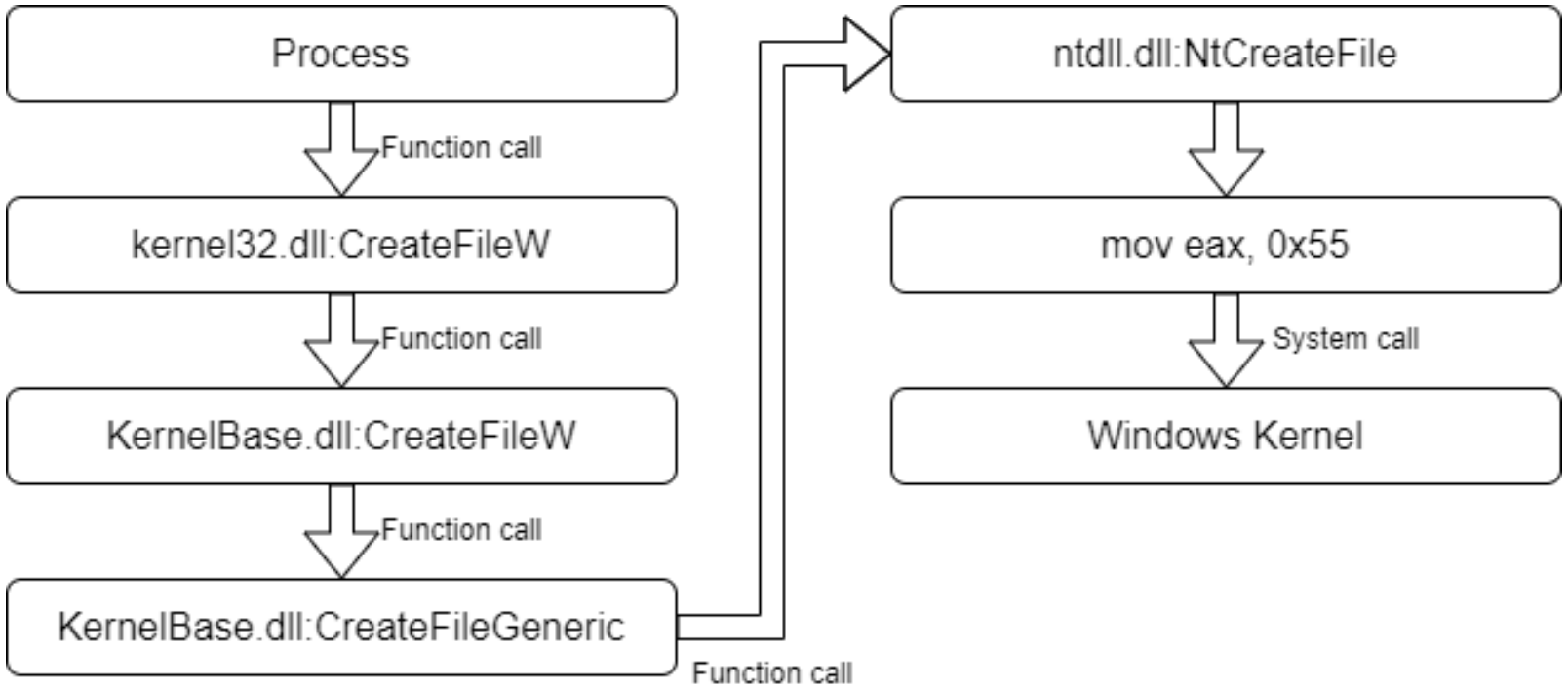
Windows API Call Example

CreateFile

```
ntdll.dll:00007FFDAC12DA70 ; =============== S U B R O U T I N E =======================================
ntdll.dll:00007FFDAC12DA70
ntdll.dll:00007FFDAC12DA70
ntdll.dll:00007FFDAC12DA70 ntdll_NtCreateFile proc near
ntdll.dll:00007FFDAC12DA70 mov     r10, rcx                    ; CODE XREF: createfile_internal_fct+572↑p
ntdll.dll:00007FFDAC12DA70                                     ; createfile_internal_fct+5E2↑p
ntdll.dll:00007FFDAC12DA70                                     ; DATA XREF: ...
ntdll.dll:00007FFDAC12DA73 mov     eax, 55h ; 'U'
ntdll.dll:00007FFDAC12DA78 test    byte_7FFE0308, 1
ntdll.dll:00007FFDAC12DA80 jnz     short loc_7FFDAC12DA85
ntdll.dll:00007FFDAC12DA82 syscall                             ; Low latency system call
ntdll.dll:00007FFDAC12DA84 retn
ntdll.dll:00007FFDAC12DA85 ; ---------------------------------------------------------------------------
ntdll.dll:00007FFDAC12DA85
ntdll.dll:00007FFDAC12DA85 loc_7FFDAC12DA85:                   ; CODE XREF: ntdll_NtCreateFile+10↑j
ntdll.dll:00007FFDAC12DA85 int     2Eh                         ; DOS 2+ internal - EXECUTE COMMAND
ntdll.dll:00007FFDAC12DA85                                     ; DS:SI -> counted CR-terminated command string
ntdll.dll:00007FFDAC12DA87 retn
ntdll.dll:00007FFDAC12DA87 ntdll_NtCreateFile endp
```

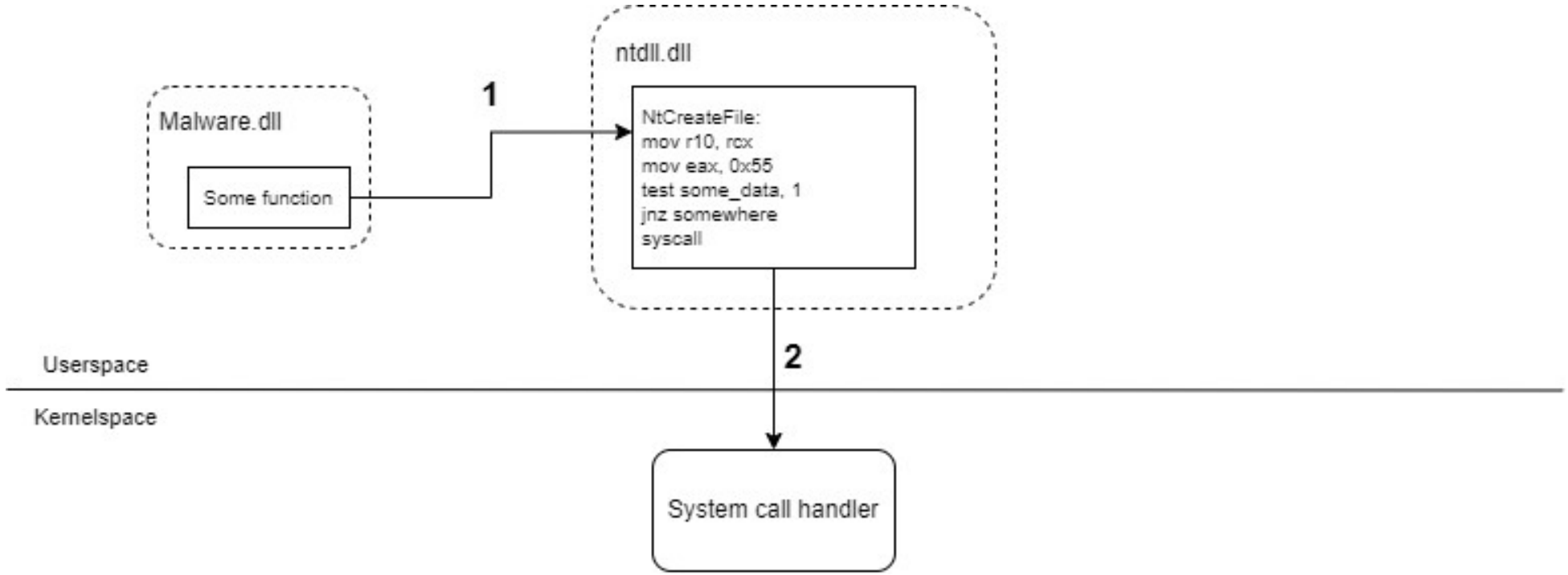
Windows API Call Example

CreateFile



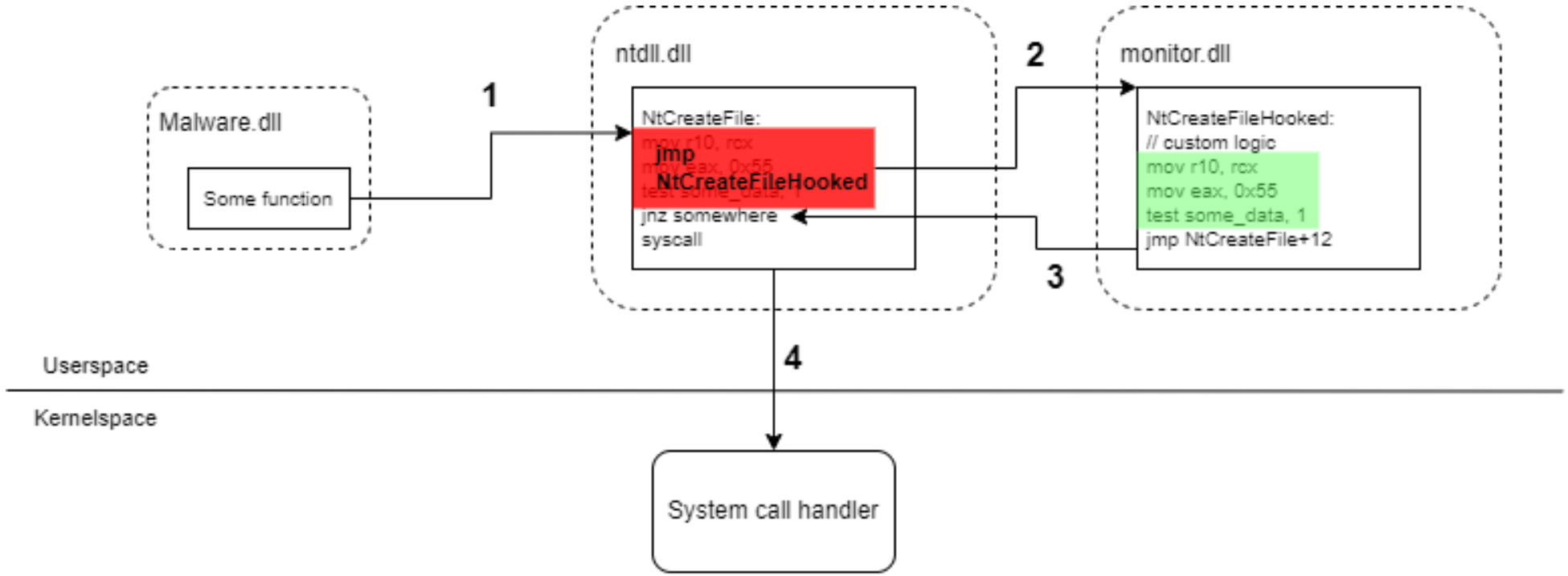
Malware User Space API Call Tracing

CreateFile (Simplified)



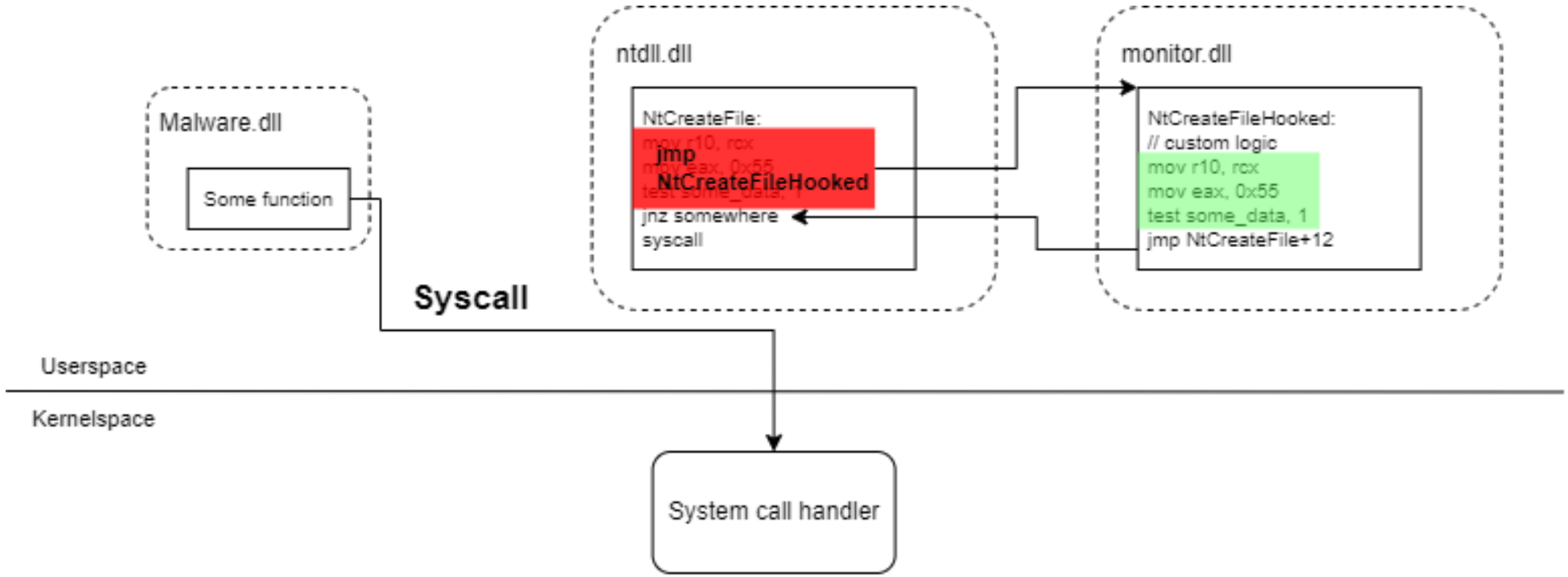
Malware User Space API Call Tracing

Intercepted System calls through ntdll.dll



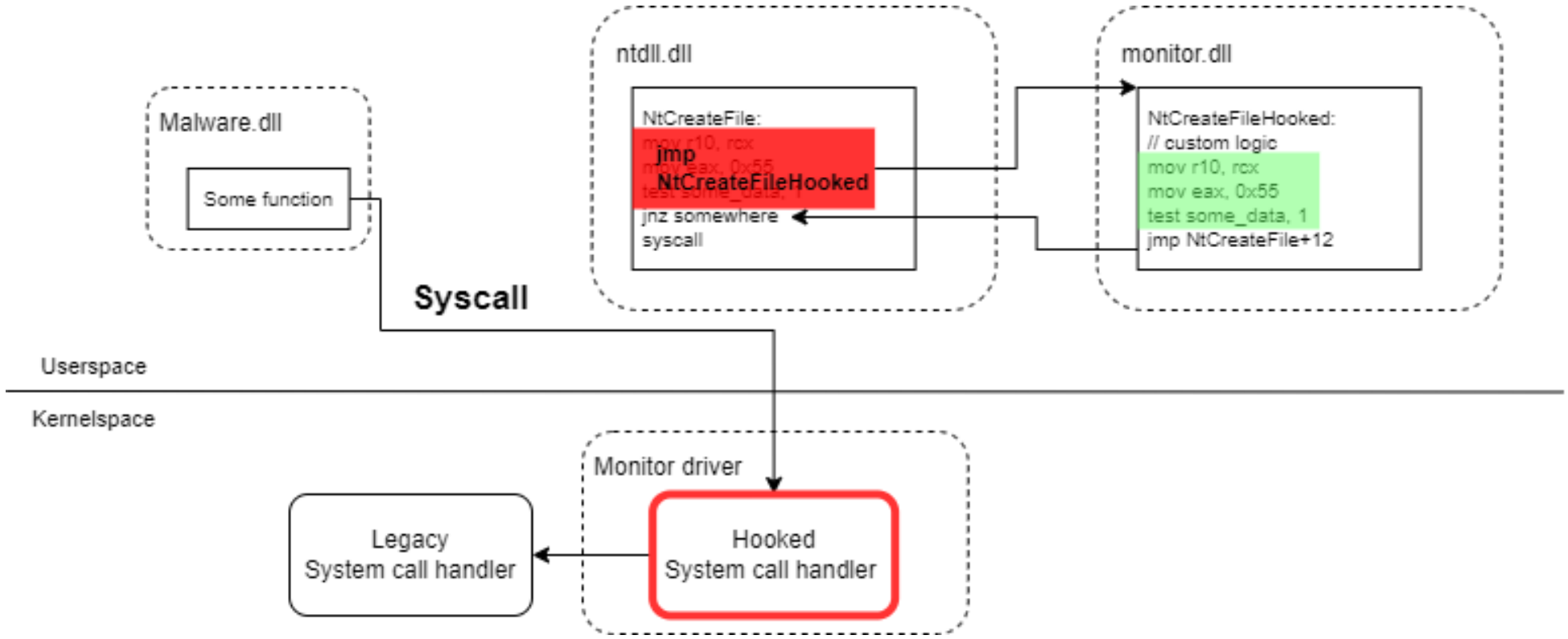
Malware User Space API Call Tracing

Untracked behavior



Kernel Space System Call Tracing

Tracked behavior



Kernel Space System Call Handling

Syscall Instruction

Operation

```
IF (CS.L ≠ 1 ) or (IA32_EFER.LMA ≠ 1) or (IA32_EFER.SCE ≠ 1)
(* Not in 64-Bit Mode or SYSCALL/SYSRET not enabled in IA32_EFER *)
    THEN #UD;
FI;
RCX := RIP; (* Will contain address of next instruction *)
RIP := IA32_LSTAR;
R11 := RFLAGS;
RFLAGS := RFLAGS AND NOT(IA32_FMASK);
CS.Selector := IA32_STAR[47:32] AND FFFCH (* Operating system provides CS; RPL forced to 0 *)
(* Set rest of CS to a fixed value *)
CS.Base := 0;
                (* Flat segment *)
CS.Limit := FFFFFH;
                (* With 4-KByte granularity, implies a 4-GByte limit *)
CS.Type := 11;
                (* Execute/read code, accessed *)
CS.S := 1;
CS.DPL := 0;
CS.P := 1;
```

Windows Kernel System Call Handling

LSTAR content

Registers

MSRs

These must be accessed through rdmsr and wrmsr

- STAR (0xC0000081) - Ring 0 and Ring 3 Segment bases, as well as SYSCALL EIP.

Low 32 bits = SYSCALL EIP, bits 32-47 are kernel segment base, bits 48-63 are user segment base.

- LSTAR (0xC0000082) - The kernel's RIP SYSCALL entry for 64 bit software.
- CSTAR (0xC0000083) - The kernel's RIP for SYSCALL in compatibility mode.
- SFMASK (0xC0000084) - The low 32 bits are the SYSCALL flag mask. If a bit in this is set, the cor

```
0: kd> rdmsr c0000082
msr[c0000082] = ffffff806`5a410100
0: kd> ln ffffff806`5a410100
Browse module
Set bu breakpoint

(ffffff806`5a410100)   nt!KiSystemCall64
Exact matches:
```

Windows Kernel System Call Handling

Nt!KiSystemCall64 internals

```
CurrentThread->FirstArgument = (void *)v26;
CurrentThread->SystemCallNumber = syscall_nr;
CurrentThread->TrapFrame = (_KTRAP_FRAME *)&v61;
syscall_table_indentifier = (syscall_nr >> 7) & 0x20;
syscall_nr_low = syscall_nr & 0xFFF;
do
{
  ke_service_desc_table_addr = &KeServiceDescriptorTable;
  v31 = &KeServiceDescriptorTableShadow;
  if ( (*((_DWORD *)&CurrentThread->0 + 1) & 0x80) != 0 )
  {
    if ( (*((_DWORD *)&CurrentThread->0 + 1) & 0x200000) != 0 )
      v31 = KeServiceDescriptorTableFilter;
    ke_service_desc_table_addr = v31;
  }
  if ( (unsigned int)syscall_nr_low < *(_DWORD *)((char *)ke_service_desc_table_addr
                                                + syscall_table_indentifier
                                                + 0x10) )
  {
    syscall_table = *(_QWORD *)((char *)ke_service_desc_table_addr + syscall_table_indentifier);
    rel_offset_fct = *(int *)(syscall_table + 4 * syscall_nr_low);
    addr_fct = (__int64 (__fastcall *)(_QWORD, _QWORD, _QWORD, _QWORD))((rel_offset_fct >> 4) + syscall_table);
    if ( (_DWORD)syscall_table_indentifier == 0x20 && *((_DWORD *)CurrentThread->Teb + 1488) )
    {
      syscall_index = rel_offset_fct;

{
  v22 = addr_fct(v26, v27, a3, a4);
}
```

SSDT or Shadow SSDT?

is syscall index valid?

get syscall relative offset from table

get actual address of syscall handler

Call the actual handler

Windows Kernel System Call Handling

SSDT entries

```
0: kd> dps nt!keservicedescriptortable L4
fffff802`600018c0  fffff802`5f2c79d0 nt!KiServiceTable
fffff802`600018c8  00000000`00000000
fffff802`600018d0  00000000`000001d9
fffff802`600018d8  fffff802`5f2c8138 nt!KiArgumentTable
```

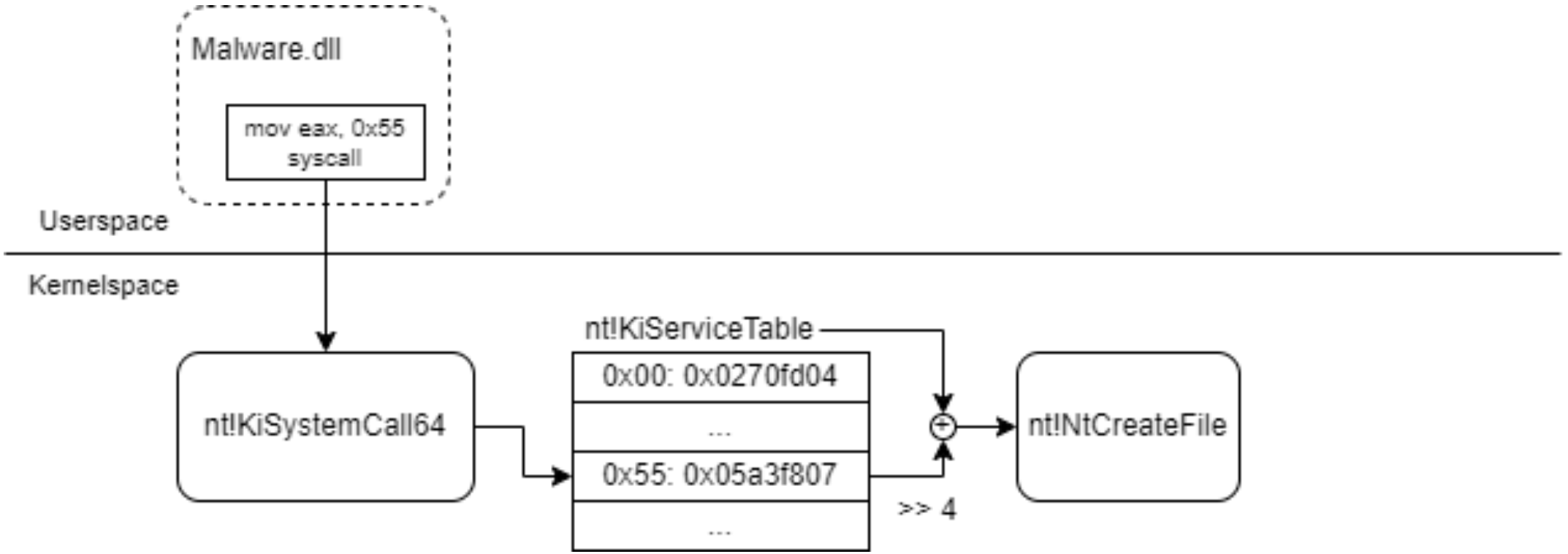
Syscall table address

Number of system calls

```
0: kd> dd /c1 nt!KiServiceTable+4*0x55 L1
fffff802`5f2c7b24  05a3f807
0: kd> u nt!KiServiceTable + (05a3f807>>>4) L1
nt!NtCreateFile:
fffff802`5f86b950 4881ec88000000  sub     rsp,88h
```

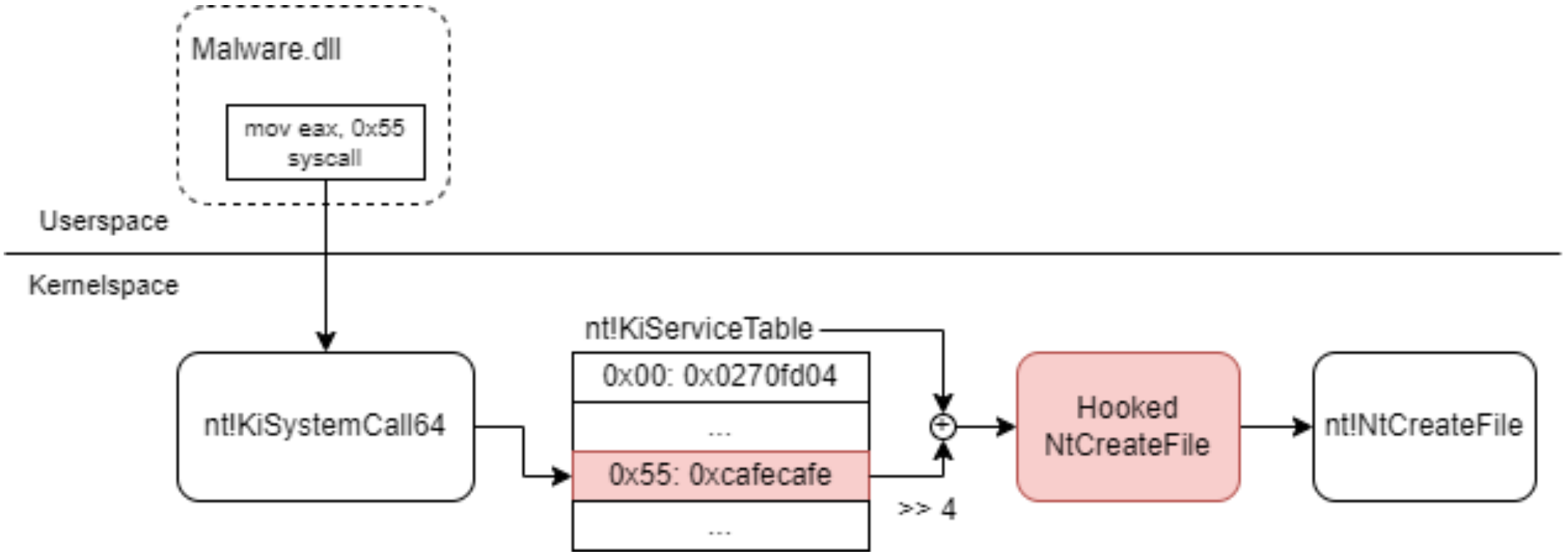
Windows Kernel System Call Handling

Summary



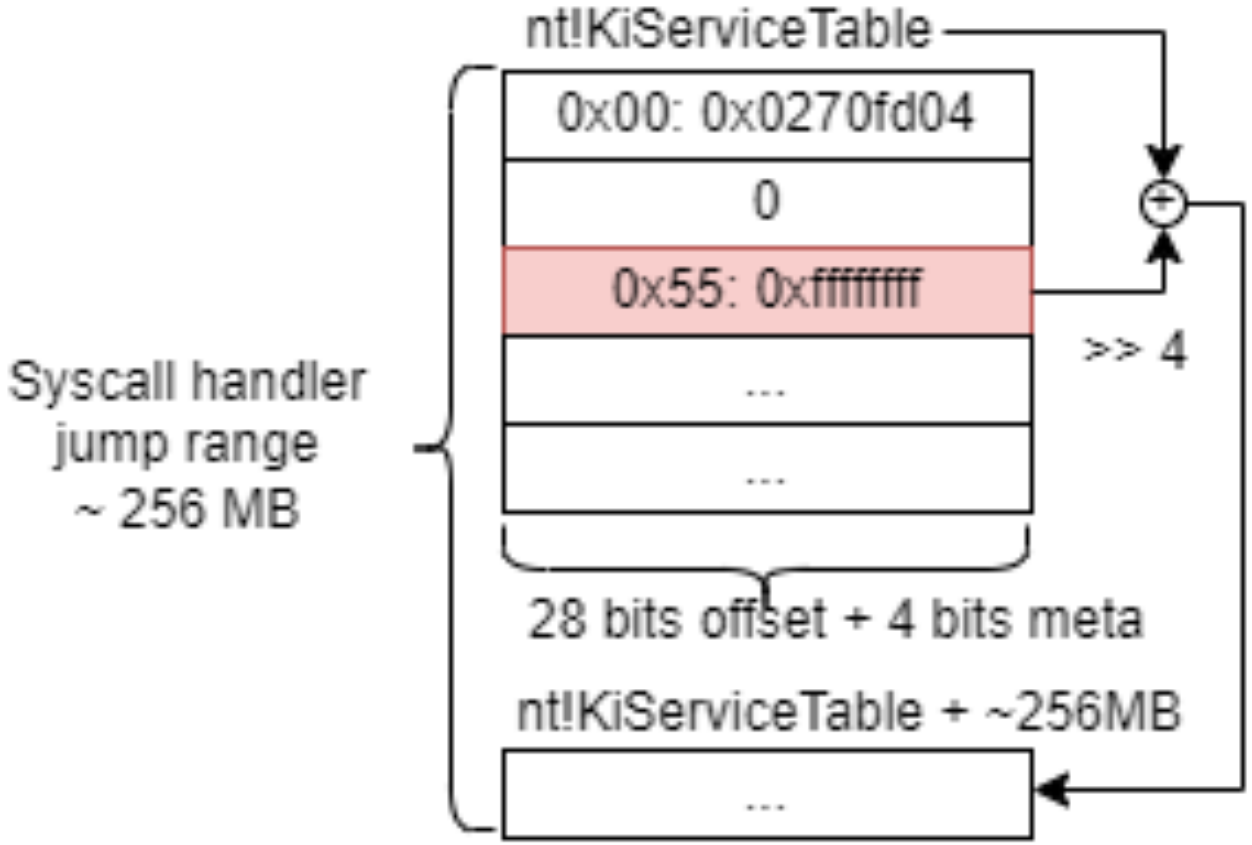
SSDT Hooking

Core idea



SSDT Hooking

Relative jump: How far can we jump?



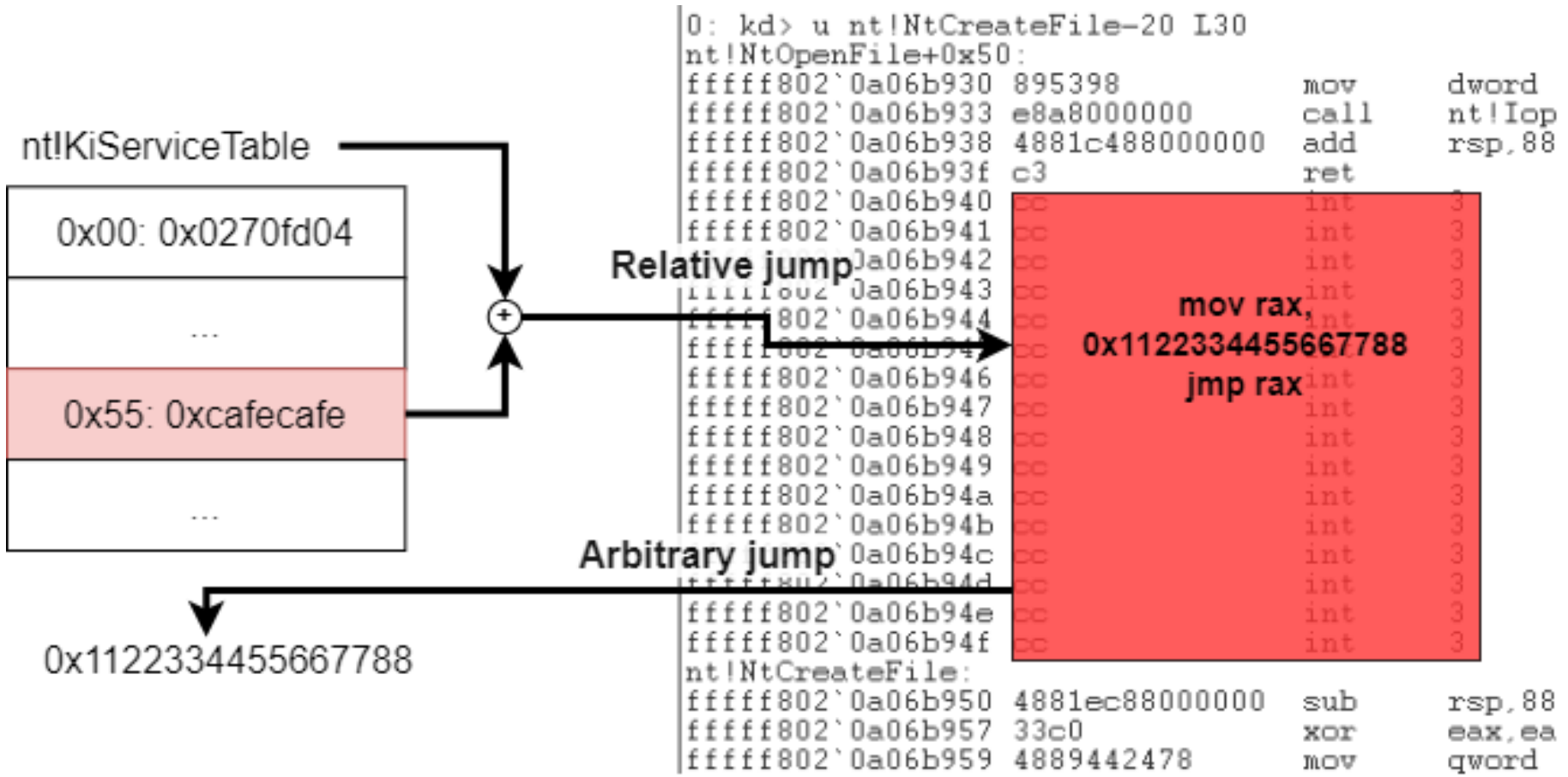
SSDT Hooking

Absolute jump

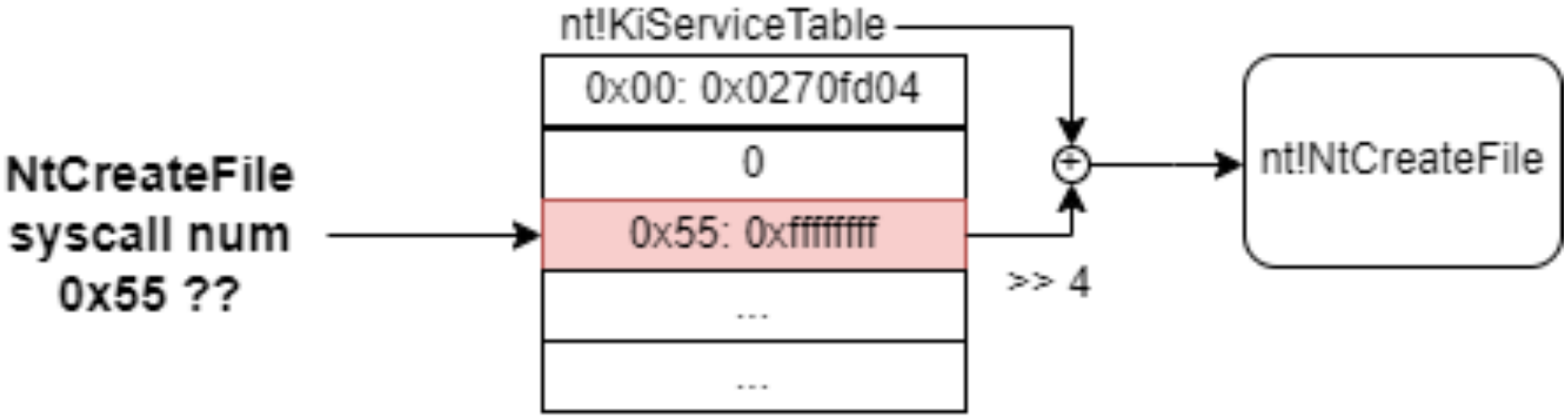
```
0: kd> u nt!NtCreateFile-20 L30
nt!NtOpenFile+0x50:
fffff802`0a06b930 895398          mov     dword
fffff802`0a06b933 e8a8000000      call    nt!Iop
fffff802`0a06b938 4881c488000000  add     rsp,88
fffff802`0a06b93f c3              ret
fffff802`0a06b940 cc              int     3
fffff802`0a06b941 cc              int     3
fffff802`0a06b942 cc              int     3
fffff802`0a06b943 cc              int     3
fffff802`0a06b944 cc              int     3
fffff802`0a06b945 cc              int     3
fffff802`0a06b946 cc              int     3
fffff802`0a06b947 cc              int     3
fffff802`0a06b948 cc              int     3
fffff802`0a06b949 cc              int     3
fffff802`0a06b94a cc              int     3
fffff802`0a06b94b cc              int     3
fffff802`0a06b94c cc              int     3
fffff802`0a06b94d cc              int     3
fffff802`0a06b94e cc              int     3
fffff802`0a06b94f cc              int     3
nt!NtCreateFile:
fffff802`0a06b950 4881ec88000000  sub     rsp,88
fffff802`0a06b957 33c0            xor     eax,ea
fffff802`0a06b959 4889442478      mov     qword
```

SSDT Hooking

Absolute jump



System Call Number Mapping



System Call Number Mapping

Public unofficial data

Windows X86-64 System Call Table (XP/2003/Vista/2008/7/2012/8/10)

Author: Mateusz "j00ru" Jurczyk (j00ru.vx tech blog)

See also: Windows System Call Tables in CSV/JSON formats on GitHub

Special thanks to: MeMek, Wandering Glitch

Layout by Metasploit Team

Enter the Syscall ID to highlight (hex):

Highlight

Show all Hide all

System Call Symbol	Windows XP (show)		Windows Server 2003 (show)				Windows Vista (show)			Windows Server 2008 (show)				Windows 7 (show)		Windows Server 2012 (show)		Windows 8 (show)		Windows 10 (hide)									
																				1507	1511	1607	1703	1709	1803	1809	1903	1909	2004	20H2
NtAcceptConnectPort																				0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
NtAccessCheck																				0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
NtAccessCheckAndAuditAlarm																				0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029	0x0029
NtAccessCheckByType																				0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063	0x0063
NtAccessCheckByTypeAndAuditAlarm																				0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059	0x0059
NtAccessCheckByTypeResultList																				0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064	0x0064
NtAccessCheckByTypeResultListAndAuditAlarm																				0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065	0x0065
NtAccessCheckByTypeResultListAndAuditAlarmByHandle																				0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066	0x0066
NtAcquireCMFViewOwnership																														

System Call Number Mapping

Manual extraction by reverse engineering ntdll.dll

```
ntdll.dll:00007FFDAC12DA70 ; =============== S U B R O U T I N E =======================================
ntdll.dll:00007FFDAC12DA70
ntdll.dll:00007FFDAC12DA70
ntdll.dll:00007FFDAC12DA70 ntdll_NtCreateFile proc near
ntdll.dll:00007FFDAC12DA70 mov     r10, rcx                    ; CODE XREF: createfile_internal_fct+572↑p
ntdll.dll:00007FFDAC12DA70                                     ; createfile_internal_fct+5E2↑p
ntdll.dll:00007FFDAC12DA70                                     ; DATA XREF: ...
ntdll.dll:00007FFDAC12DA73 mov     eax, 55h ; 'U'
ntdll.dll:00007FFDAC12DA78 test    byte_7FFE0308, 1
ntdll.dll:00007FFDAC12DA80 jnz     short loc_7FFDAC12DA85
ntdll.dll:00007FFDAC12DA82 syscall                             ; Low latency system call
ntdll.dll:00007FFDAC12DA84 retn
ntdll.dll:00007FFDAC12DA85 ; ---------------------------------------------------------------------------
ntdll.dll:00007FFDAC12DA85
ntdll.dll:00007FFDAC12DA85 loc_7FFDAC12DA85:                   ; CODE XREF: ntdll_NtCreateFile+10↑j
ntdll.dll:00007FFDAC12DA85 int     2Eh                         ; DOS 2+ internal - EXECUTE COMMAND
ntdll.dll:00007FFDAC12DA85                                     ; DS:SI -> counted CR-terminated command string
ntdll.dll:00007FFDAC12DA87 retn
ntdll.dll:00007FFDAC12DA87 ntdll_NtCreateFile endp
```

System Call Number Mapping

Hell's Gate

```
if (*((PBYTE)pFunctionAddress + cw) == 0x4c
                    && *((PBYTE)pFunctionAddress + 1 + cw) == 0x8b
                    && *((PBYTE)pFunctionAddress + 2 + cw) == 0xd1
                    && *((PBYTE)pFunctionAddress + 3 + cw) == 0xb8
                    && *((PBYTE)pFunctionAddress + 6 + cw) == 0x00
                    && *((PBYTE)pFunctionAddress + 7 + cw) == 0x00) {
                    BYTE high = *((PBYTE)pFunctionAddress + 5 + cw);
                    BYTE low = *((PBYTE)pFunctionAddress + 4 + cw);
                    pVxTableEntry->wSystemCall = (high << 8) | low;
                    break;
}
```

Bypassing Protections

Driver Signature Check

```
C:\Windows\system32>sc create kerndriver type= kernel binpath=C:\Users\Admin\Desktop\kerndriver.sys
[SC] CreateService SUCCESS

C:\Windows\system32>sc start kerndriver
[SC] StartService FAILED 577:

Windows cannot verify the digital signature for this file. A recent hardware or software change might have installed a f
ile that is signed incorrectly or damaged, or that might be malicious software from an unknown source.
```

```
C:\Windows\system32>bcdedit /set testsigning on
The operation completed successfully.
```

Bypassing Protections

Write Page Protection

```
PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced.  This cannot be protected by try-except.
Typically the address is just plain bad or it is pointing at freed memory.
Arguments:
Arg1: fffff80478600000, memory referenced.
Arg2: 0000000000000003, value 0 = read operation, 1 = write operation.
Arg3: fffff8050cd54297, If non-zero, the instruction address which referenced the bad memory
      address.
Arg4: 0000000000000002, (reserved)
```

Bypassing Protections

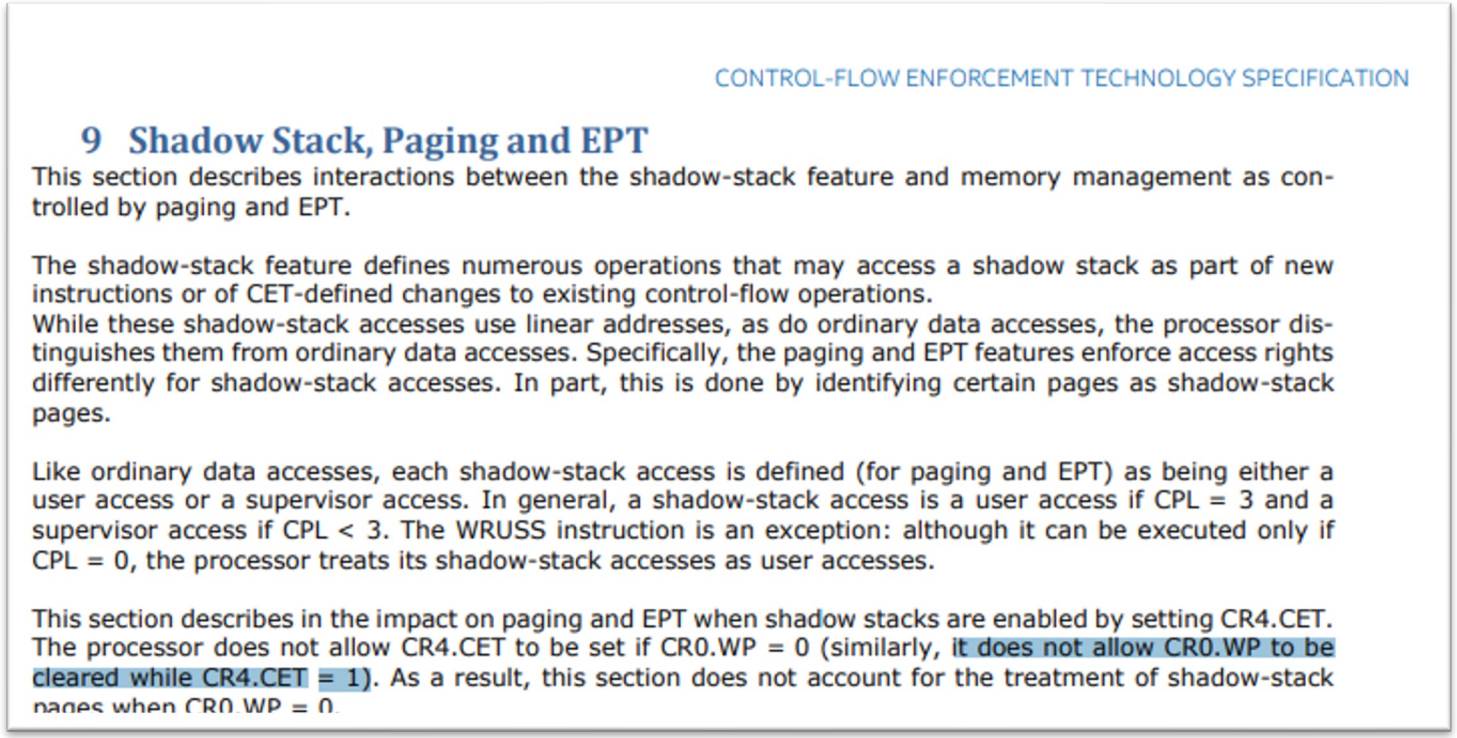
Write Page Protection

```
47      KIRQL WPOFF( )
48      {
49          KIRQL Irql = KeRaiseIrqlToDpcLevel();
50          UINT_PTR cr0 = __readcr0();
51
52          cr0 &= ~0x10000;
53          __writecr0( cr0 );
54          _disable();
55
56          return Irql;
57      }
```

```
PROCESS_NAME:  System
ERROR_CODE: (NTSTATUS) 0xc0000096 - {EXCEPTION}  Privileged instruction.
EXCEPTION_CODE_STR:  c0000096
EXCEPTION STR:  0xc0000096
```

Bypassing Protections

Intel CET



CONTROL-FLOW ENFORCEMENT TECHNOLOGY SPECIFICATION

9 Shadow Stack, Paging and EPT

This section describes interactions between the shadow-stack feature and memory management as controlled by paging and EPT.

The shadow-stack feature defines numerous operations that may access a shadow stack as part of new instructions or of CET-defined changes to existing control-flow operations.
While these shadow-stack accesses use linear addresses, as do ordinary data accesses, the processor distinguishes them from ordinary data accesses. Specifically, the paging and EPT features enforce access rights differently for shadow-stack accesses. In part, this is done by identifying certain pages as shadow-stack pages.

Like ordinary data accesses, each shadow-stack access is defined (for paging and EPT) as being either a user access or a supervisor access. In general, a shadow-stack access is a user access if CPL = 3 and a supervisor access if CPL < 3. The WRUSS instruction is an exception: although it can be executed only if CPL = 0, the processor treats its shadow-stack accesses as user accesses.

This section describes in the impact on paging and EPT when shadow stacks are enabled by setting CR4.CET. The processor does not allow CR4.CET to be set if CR0.WP = 0 (similarly, it does not allow CR0.WP to be cleared while CR4.CET = 1). As a result, this section does not account for the treatment of shadow-stack pages when CR0.WP = 0.

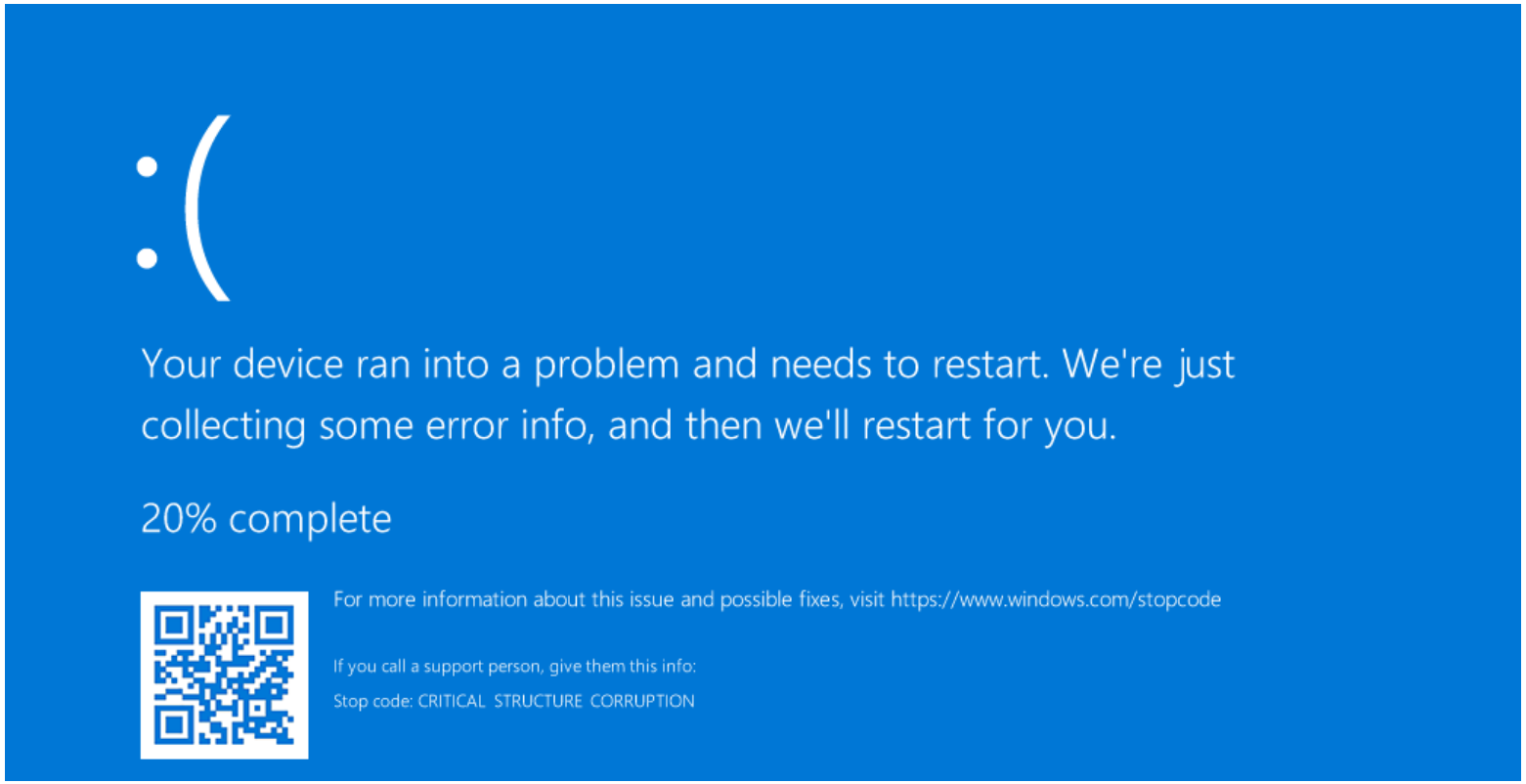
Bypassing Protections

Bypass Intel CET

- Disable interrupts, Disable CET, Disable WP
- Write data into memory
- Enable WP, Enable CET, Enable interrupts

Bypassing Protections

PatchGuard



Bypassing Protections

Disable PatchGuard using EFIGuard

```
[PatchNtoskrnl] ntoskrnl.exe at 0xFFFFF80354408000, size 0x1046000
[PatchNtoskrnl] Patching ntoskrnl.exe v10.0.19041.3324...
[PatchNtoskrnl] Disabling PatchGuard... [INIT RVA: 0xA1F000 - 0xAA9A00]

== Searching for nt!KeInitAmd64SpecificState pattern in INIT ==
    Found KeInitAmd64SpecificState pattern at 0xFFFFF80354E5F12E.
== Disassembling INIT to find nt!CcInitializeBcbProfiler ==
    Found CcInitializeBcbProfiler pattern at 0xFFFFF80354E2736D.
== Disassembling INIT to find nt!ExpLicenseWatchInitWorker ==
    Found ExpLicenseWatchInitWorker pattern at 0xFFFFF80354E704A4.
== Searching for nt!KiVerifyScopesExecute pattern in INIT ==
    Found KiVerifyScopesExecute pattern at 0xFFFFF80354E6D422.
== Searching for nt!KiMcaDeferredRecoveryService pattern in .text ==
    Found KiMcaDeferredRecoveryService pattern at 0xFFFFF80354810BE0.
== Searching for nt!KiSwInterrupt pattern in .text ==
    Found KiSwInterrupt pattern at 0xFFFFF8035480BD08.

    Patched KeInitAmd64SpecificState [RVA: 0xA57108].
    Patched CcInitializeBcbProfiler [RVA: 0xA1F354].
    Patched ExpLicenseWatchInitWorker [RVA: 0xA68474].
    Patched KiVerifyScopesExecute [RVA: 0xA65400].
    Patched KiMcaDeferredRecoveryService [RVAs: 0x5BF400, 0x5BF430].
    Patched KiSwInterrupt [RVA: 0x403D08].

[PatchNtoskrnl] Successfully disabled PatchGuard.

Successfully patched ntoskrnl.exe.
```

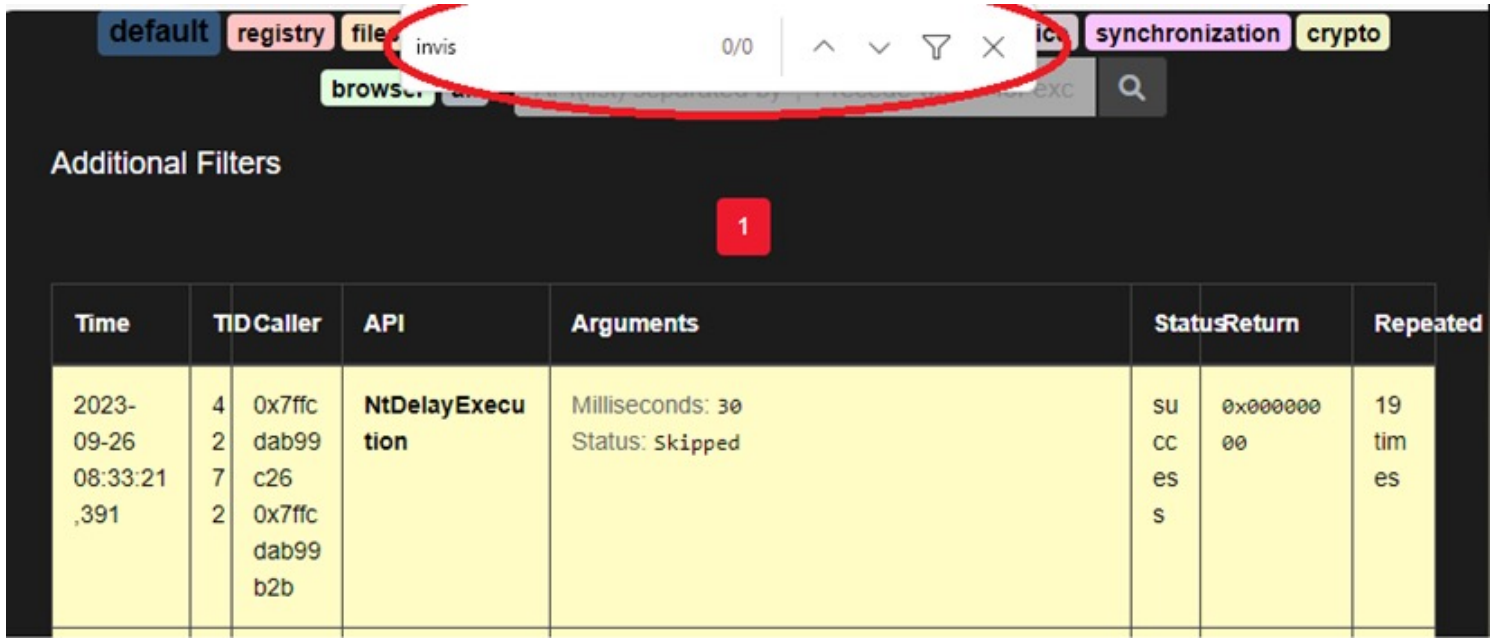
Summary

Challenges

- Insufficient public documentation
  - Reverse engineering required
  - Some Kernel APIs were not exposed, had to rely on reverse engineering forums
- A lot of protections(Hardware and Software)
  - System call numbers change across versions
  - SSDT entries allow only small relative jumps
  - Driver Signature Check
  - Page Write Protection
  - Intel CET from Intel gen 11
  - PatchGuard

Results

Public CAPEv2 vs Kernel Syscall Tracing




```
KST_KMDF: hooked_NtCreateUserProcess: PID:6904 ChildPID:1160 ImagePathName:C:\Users\Admin\Desktop\demo_writefile.exe
CommandLine:demo_writefile.exe
KST_KMDF: hooked_NtCreateFile: PID:5028 ObjectName:\??\C:\Users\Admin\AppData\Roaming\Microsoft\Windows\Recent
\AutomaticDestinations
KST_KMDF: hooked_NtCreateFile: PID:6904 ObjectName:\SystemRoot\AppPatch\sysmain.sdb
KST_KMDF: hooked_NtCreateFile: PID:524 ObjectName:\SystemRoot\AppPatch\sysmain.sdb
KST_KMDF: hooked_NtCreateFile: PID:1160 ObjectName:\Connect
KST_KMDF: hooked_NtCreateFile: PID:3636 ObjectName:\??\C:\Users\Admin\Desktop\demo_writefile.exe
KST_KMDF: hooked_NtCreateFile: PID:1160 ObjectName:\??\c:\temp\invisible_file_write.txt
```

KEYSIGHT

Q&A

KEYSIGHT

Thank you

References

- https://www.felixcloutier.com/x86/syscall
- https://wiki.osdev.org/SYSENTER
- https://www.ired.team/miscellaneous-reversing-forensics/windows-kernel-internals/glimpse-into-ssdt-in-windows-x64-kernel
- https://alice.climent-pommeret.red/posts/a-syscall-journey-in-the-windows-kernel/
- https://j00ru.vexillium.org/syscalls/nt/64/
- https://alice.climent-pommeret.red/posts/direct-syscalls-hells-halos-syswhispers2/
- https://m0uk4.gitbook.io/notebooks/mouka/windowsinternal/ssdt-hook
- https://github.com/klezVirus/SysWhispers3

References

- https://github.com/conix-security/zer0m0n
- https://gist.github.com/ksose/1064568
- https://kib.kiev.ua/x86docs/Intel/CET/334525-003.pdf