Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the presenters individually and, unless expressly stated to the contrary, are not the opinion or position of RSA Conference LLC or any other co-sponsors. RSA Conference does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented.

Attendees should note that sessions may be audio- or video-recorded and may be published in various media, including print, audio and video formats without further notice. The presentation template and any media capture are subject to copyright protection.

RSAConference2022
San Francisco & Digital | June 6 – 9

TRANSFORM

SESSION ID: **HTA-T08**

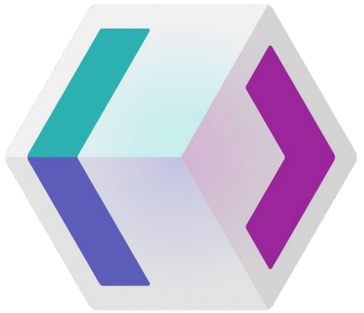
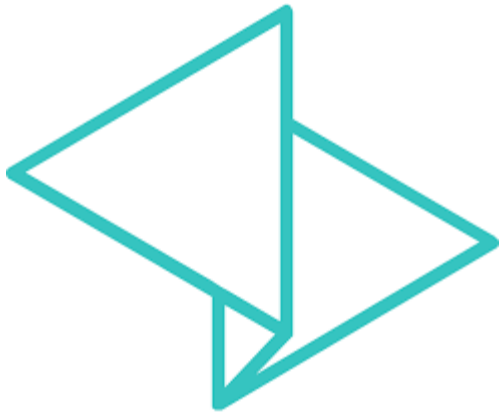
Defeating Windows Anti-Exploit & Security Features with WHQL Kernel Drivers

Arush Agarampur

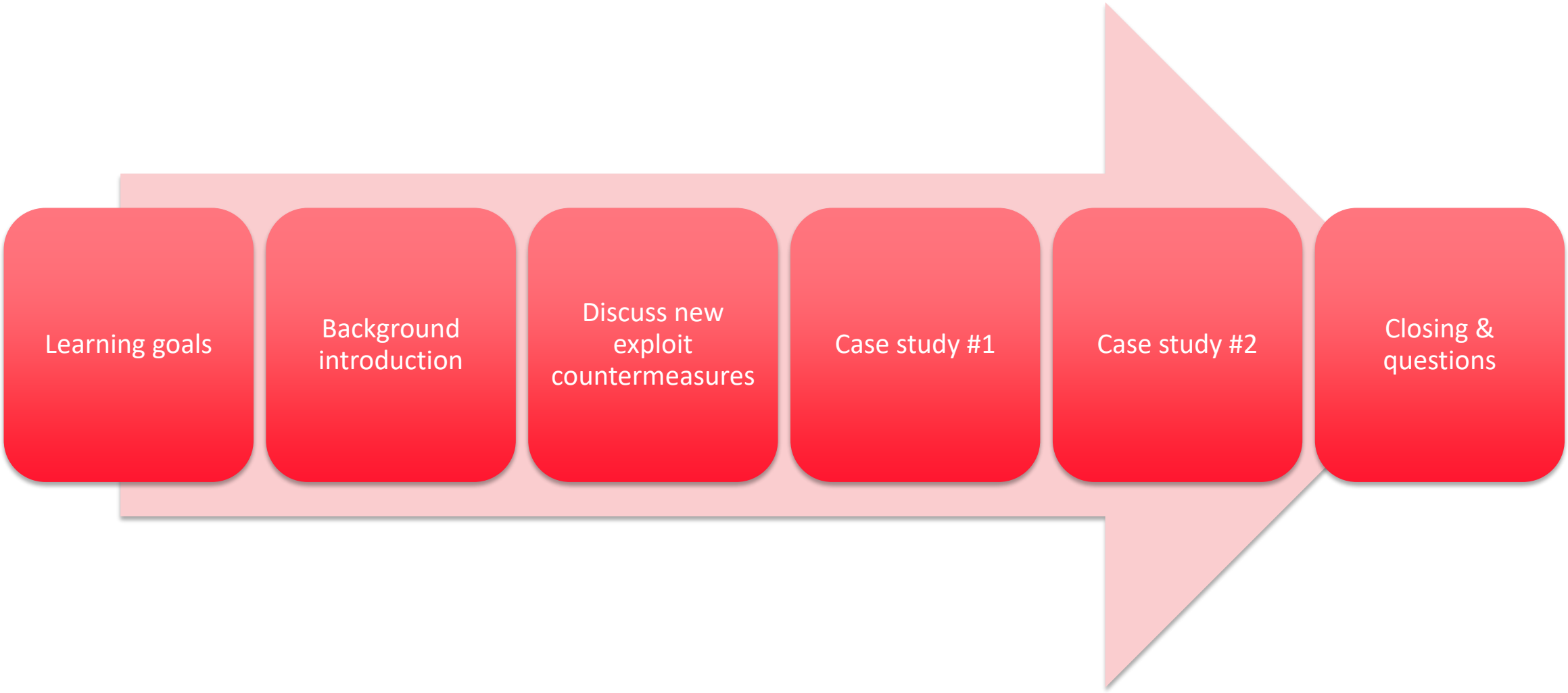
Student at Rutgers University | New Brunswick
@axagarampur

About Me

- Self-taught programmer
- Low-level code enthusiast
  - Operating systems
  - Reverse engineering
  - Security
- Work with Windows
  - NT Kernel
  - OS technologies & internals
  - Windows Runtime
  - UI development

HTA-T08 — Agenda



Learning goals

You will learn how:

- One may circumvent new OS anti-exploit features
- “Trusted code” may not be so trustworthy
- One can use security products to bypass OS security boundaries (isolation, hypervisors, etc.)

You can apply this information to:

- Your organization’s software dependencies
- Your group’s security research & operations
- Decisions about trusting external security products

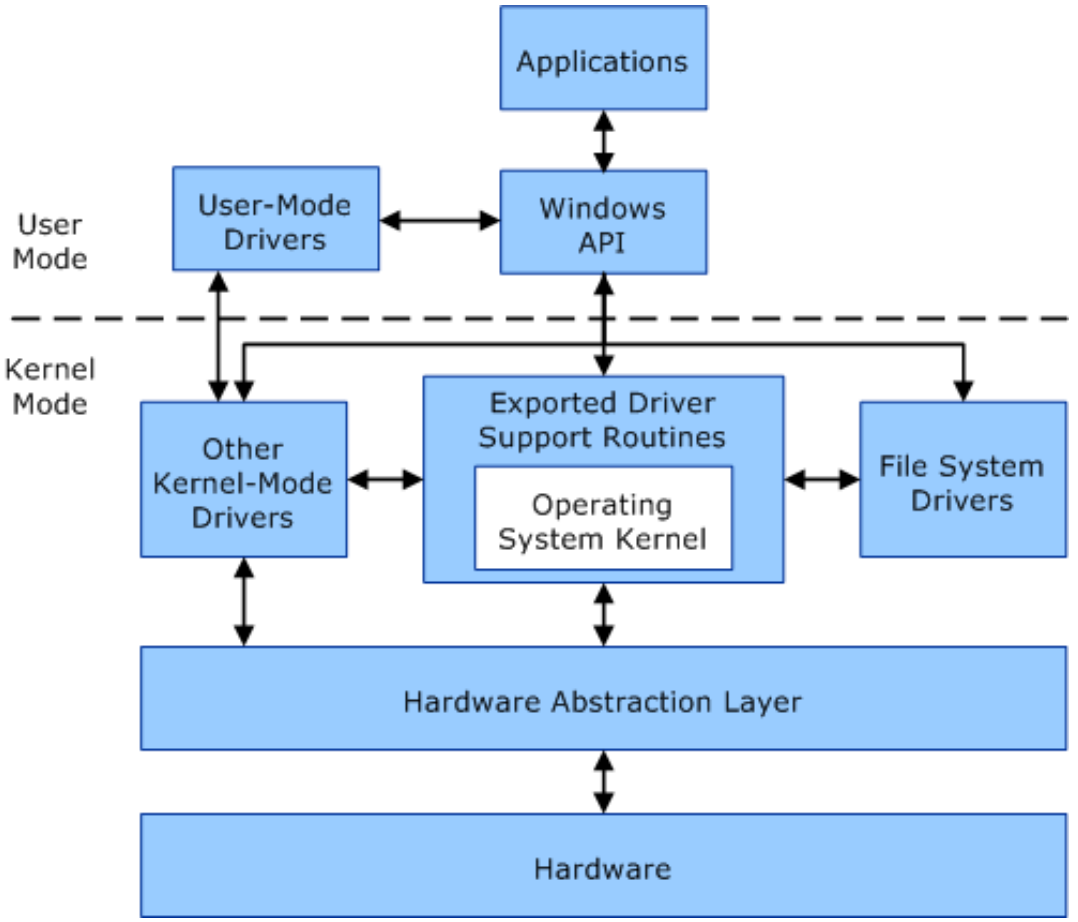
RSAConference2022

Background information

Quick rundown about Windows memory spaces, device drivers, and WHQL signing

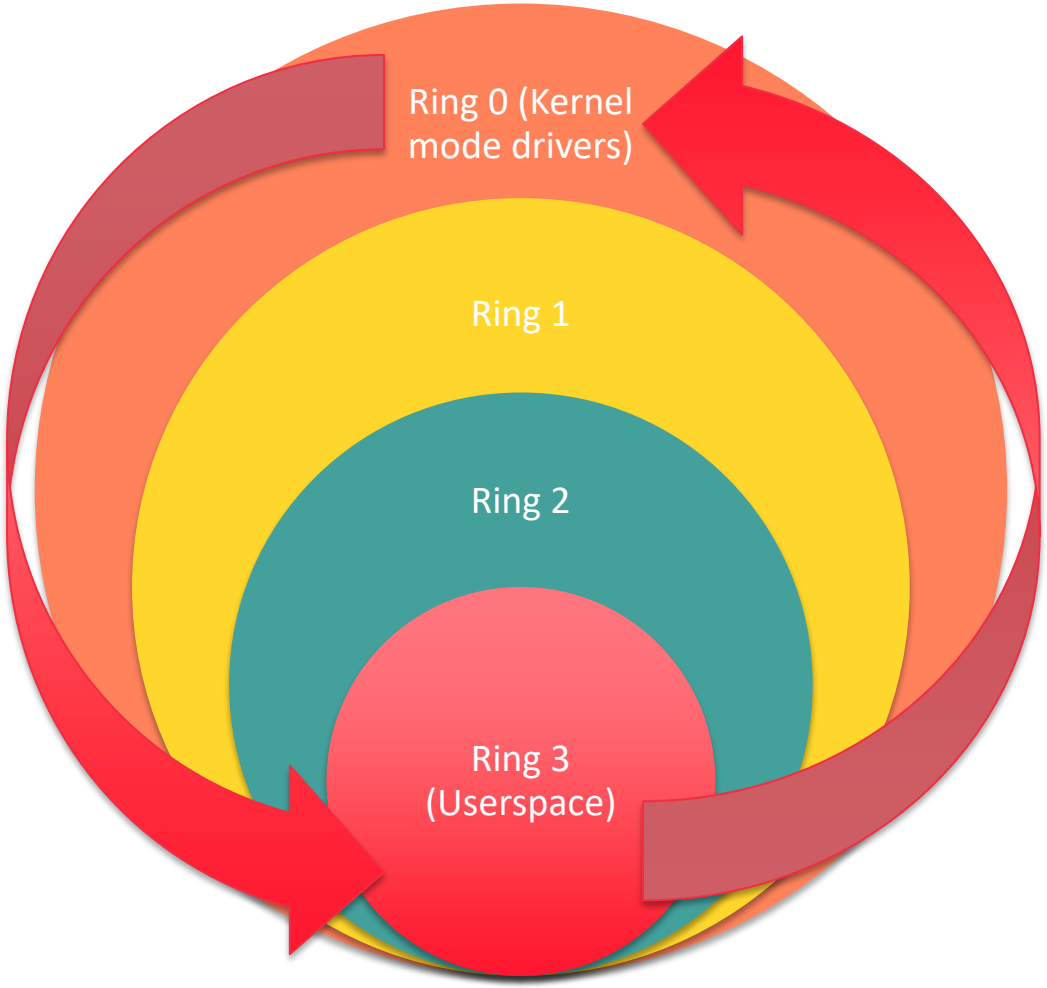
Windows memory spaces

- Userspace (everything **you** do is here)
  - Isolated by hardware (ring 3)
  - Communicates with kernelspace (ring 0) via **system calls**
    - Special CPU instructions to move program context into ring 0 from ring 3
- Kernelspace contains OS core
  - Controls **everything**
  - Manages hardware, other programs, private information, etc.



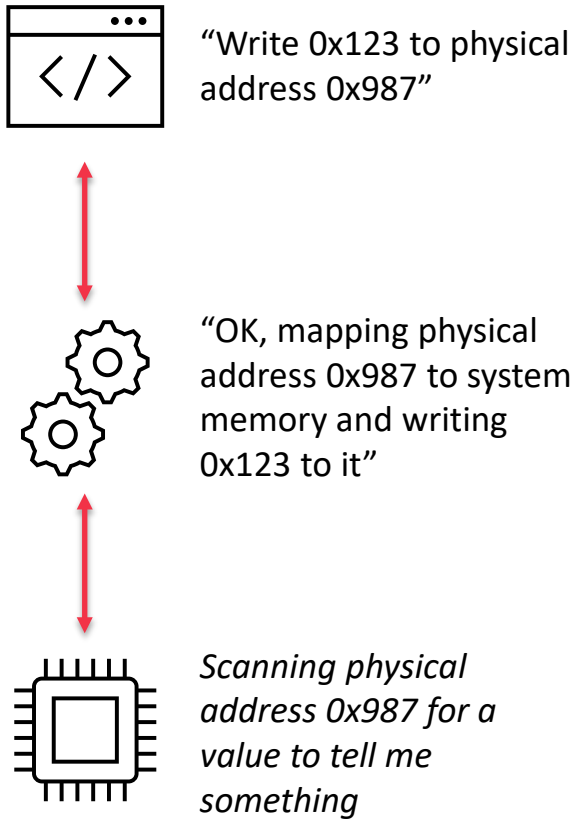
Device driver architecture

- Driver – loadable code that performs privileged operations
  - Hardware control
  - Privileged software
    - Hypervisors, antivirus drivers, etc.
- Accepts userspace “requests”
  - I/O Control
  - Perform driver-specific task(s)
  - Make hardware perform task(s)



Common device driver functionality

- Query privileged information
- Interact with hardware components
- Memory management
  - Read/Write system memory
  - Modify device-specific memory
  - Map physical RAM to system memory space
- Accepts these “commands” from userspace
  - Easier to program in userspace
  - Driver does bare-minimum functionality



Device driver implementation

- Use the OS kernel
  - Call specific functions
- Setup fake “device object”
  - Communication with userspace via I/O control commands to this “device object”
  - Treated as a file

IoCreateDevice function (wdm.h)

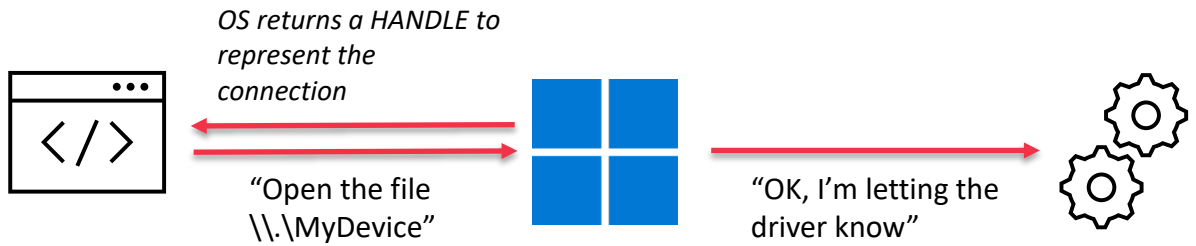
Article • 04/17/2022 • 3 minutes to read

The **IoCreateDevice** routine creates a device object for use by a driver.

DeviceIoControl function (ioapiset.h)

Article • 01/27/2022 • 4 minutes to read

Sends a control code directly to a specified device driver, causing the corresponding device to perform the corresponding operation.



Device driver implementation

- Memory management
  - Map physical addresses → system virtual addresses via system PTEs
  - Access physical memory directly
    - Section objects
    - Map physical memory section
- Other functionality
  - Call kernel functions as normal

MmMapIoSpace function (wdm.h)

Article • 02/24/2022 • 2 minutes to read

The **MmMapIoSpace** routine maps the given physical address range to nonpaged system space.

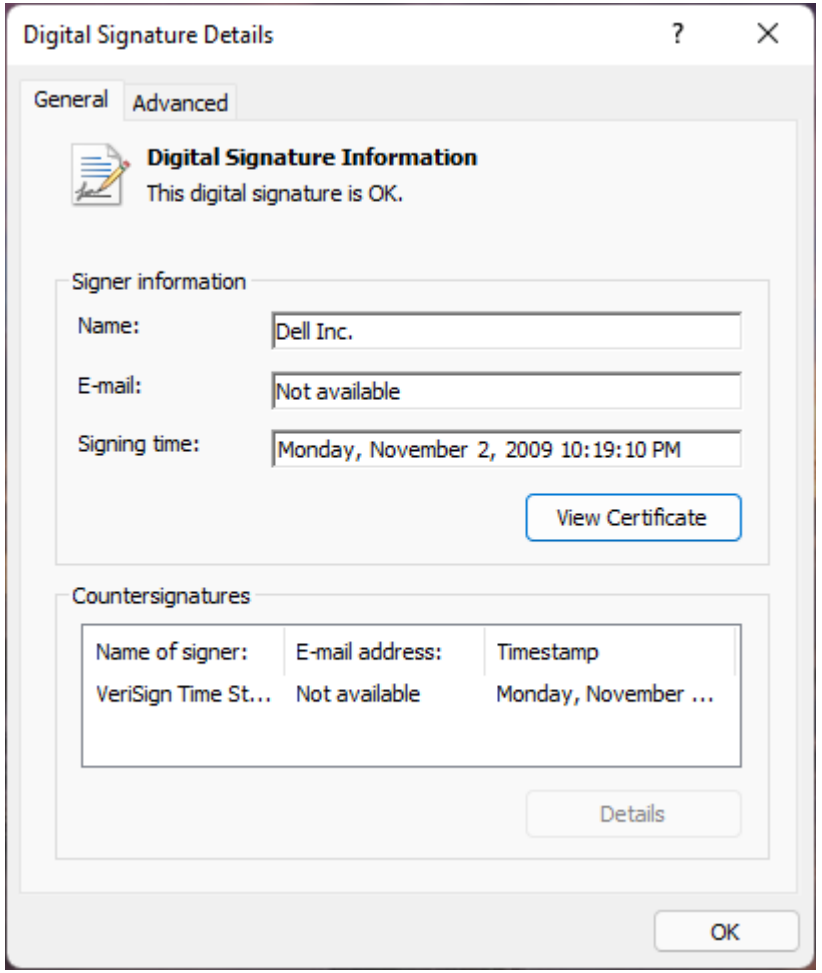
ZwMapViewOfSection function (wdm.h)

Article • 04/17/2022 • 4 minutes to read

The **ZwMapViewOfSection** routine maps a view of a section into the virtual address space of a subject process.

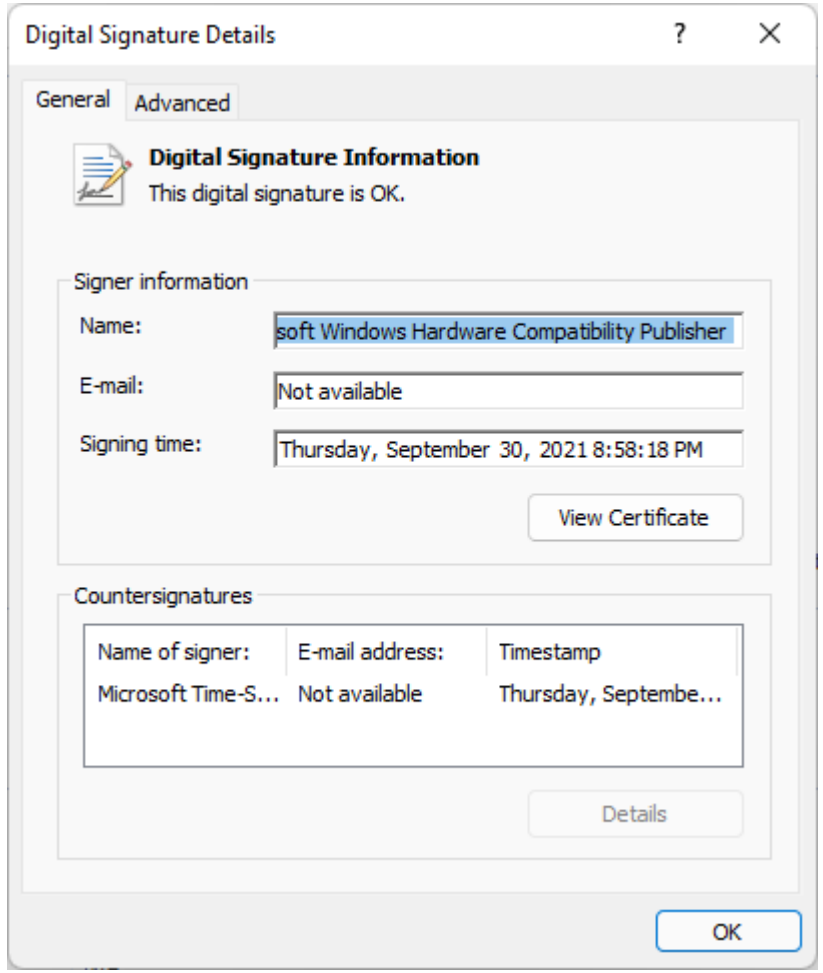
Exploitable much?

- Hundreds of exploits found
  - Based off direct memory access
  - Been going on for years: **CVE-2021-41285, CVE-2022-21814, CVE-2022-22516**
- Most driver vendors don't produce high quality, safe drivers
- Modify kernel memory to steal information, gain elevated privileges, etc.
- **These drivers are signed**



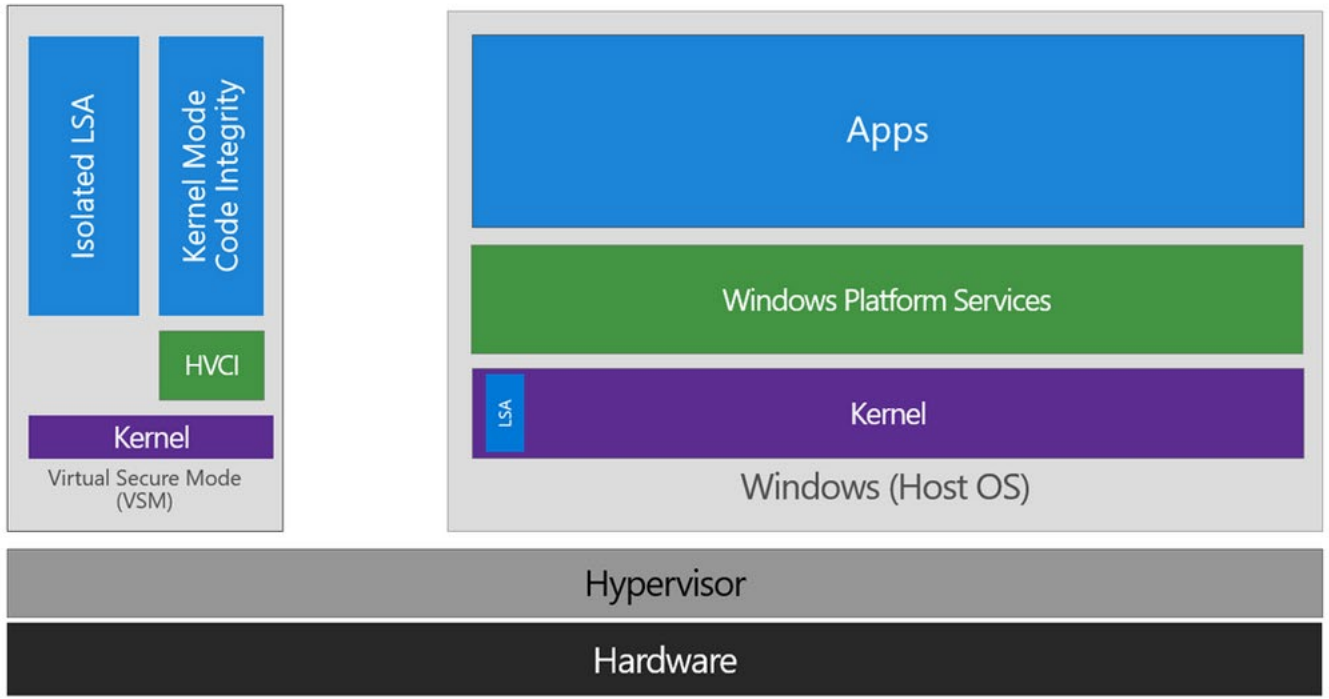
WHQL Signing

- Different type of driver signing
  - “Tested from Microsoft”
  - “Robust reliability, quality, and compatibility”
  - Gets a “Microsoft Windows” publisher name
- Other benefits
  - Automatically installs
  - Widespread: downloadable from Windows update
- **Do they deserve the name?**



Anti-Exploit features

- Hardened kernel routines
  - Can’t map page tables (MmMapIoSpace)
  - Default memory pool is NoExecute
- Kernel address space layout randomization (KASLR)
  - Address of kernel structures are selected at random
  - Change every boot cycle
- Virtualization Based Security
  - Runs the OS under a specially designed hypervisor
  - Provides multiple services
    - Hypervisor enforced code integrity (HVCI)
    - Kernel mode code integrity (KMCI)
    - Isolated User Mode (IUM)
    - And more



Case study #1

Goal: bypass kernel routine hardening and KASLR to execute unsigned kernel code stealthily

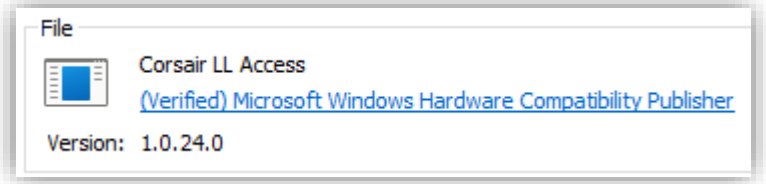
VBS is turned OFF

Corsair LL Access — first glance

HAL.dll	HalGetBusDataByOffset	25
HAL.dll	HalSetBusDataByOffset	61
ntoskrnl.exe	IoAllocateMdl	619
ntoskrnl.exe	IoCreateDevice	659
ntoskrnl.exe	IoCreateSymbolicLink	670
ntoskrnl.exe	IoDeleteDevice	682
ntoskrnl.exe	IoDeleteSymbolicLink	684
ntoskrnl.exe	IoFreeMdl	703
ntoskrnl.exe	IoGetRequestorProcessId	749
ntoskrnl.exe	IofCompleteRequest	938
ntoskrnl.exe	KeBugCheckEx	985
ntoskrnl.exe	KeInitializeMutex	1055
ntoskrnl.exe	KeReleaseMutex	1146
ntoskrnl.exe	KeWaitForSingleObject	1221
ntoskrnl.exe	MmBuildMdlForNonPagedPool	1273
ntoskrnl.exe	MmMapIoSpace	1317
ntoskrnl.exe	MmMapLockedPagesSpecifyCache	1320

Multiple memory APIs in use
- I/O space mappings
- MDL mappings

Simple device creation



WHQL Signed

- Can we overwrite kernel memory using this driver?
  - Maybe
- Good initial impression, more analysis needed

Corsair LL Access — static analysis

```
TokenInformation = 0i64;
v5 = SeQueryInformationToken(v9, TokenElevation, &TokenInformation);
if ( v5 >= 0 )
{
  v10 = *(_DWORD *)TokenInformation;
  ExFreePoolWithTag(TokenInformation, 0);
  LODWORD(v12) = 0;
  v5 = SeQueryInformationToken(v9, TokenIntegrityLevel, &v12);
  if ( v5 >= 0 )
  {
    if ( !v10 || (unsigned int)v12 < 0x3000 )
      v3 = 0;
    v5 = v3 == 0 ? 0xC0000022 : 0;
  }
}
```

Checks if process is at least administrator upon opening device ☺

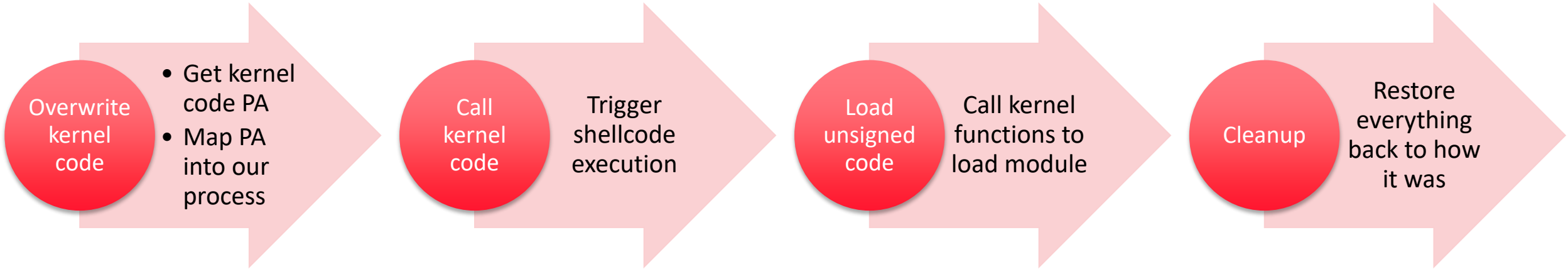
```
v10 = MmMapIoSpace(*a3, a3[1].LowPart, MmNonCached);
v11 = v10;
if ( v10 )
{
  v12 = IoAllocateMdl(v10, a3[1].LowPart, 0, 0, 0i64);
  v13 = v12;
  if ( !v12 )
  {
    MmUnmapIoSpace(v11, a3[1].LowPart);
    v8 = -1073741670;
    goto LABEL_17;
  }
  MmBuildMdlForNonPagedPool(v12);
  v14 = (unsigned __int64)MmMapLockedPagesSpecifyCache(v13, 1, MmNonCached, 0i64, 0, dword_140004344 | 0x20u);
  v15 = v13->ByteOffset;
  v14 &= 0xFFFFFFFFFFFFF000ui64;
```

Can access physical memory using a specific IOCTL:

- Maps I/O space to system space
- Allocates MDL to represent I/O mapping
- Maps the MDL into ***userspace***

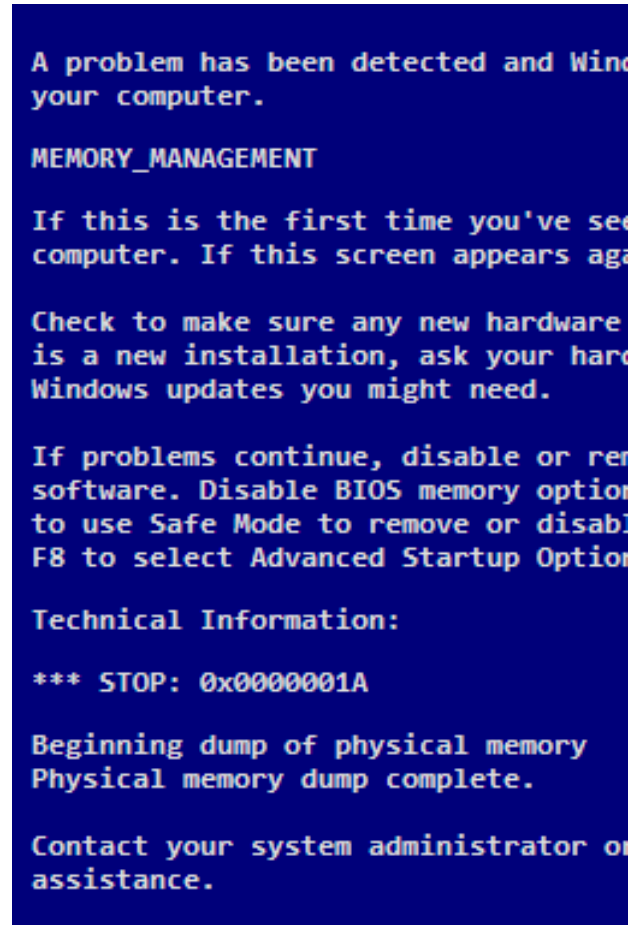
Designing an attack

- Driver allows us to map physical RAM into our userspace program
  - Attack based off this core functionality



Immediate problems

- Working with PA
  - We have kernel memory VA
  - Convert into PA
  - MmGetPhysicalAddress is in kernelspace ☹
- Ideas to get kernel memory PA
  - ~~Brute force scan from start/random areas of RAM~~
    - Slow, inefficient, and causes BSODs
  - ~~Use known PML4/kernel data structures PA~~
    - KASLR exists
  - ~~Scan for self-referencing PML4 entry and convert VA to PA manually~~
    - Hardened MmMapIoSpace will not map page tables



AMD64 paging abuse

- Narrow down possible PA that have desired kernel code
  - Use VA information of desired kernel function
  - Function I chose to use: **NtVdmControl**
    - System call implementation
    - Is not used – simply returns STATUS_NOT_IMPLIMENTED
    - Plenty of space for shellcode
  - Use relative virtual address (RVA) of function to narrow down PA

```
; Exported entry 1625. NtVdmControl
; Exported entry 2470. SeAdjustObjectSecurity

; =============== S U B R O U T I N E =======================================


                public NtWaitLowEventPair
NtWaitLowEventPair proc near            ; CODE XREF: MiKernelWriteToExecutableMemory+68↑p
                                        ; DATA XREF: .rdata:0000000140006130↑o ...
                mov     eax, 0C0000002h ; NtVdmControl
                retn
; ---------------------------------------------------------------------------
                db 0CCh
NtWaitLowEventPair endp
```

AMD64 paging abuse — cont.

#	RVA	Name
1692	0x7dda70	NtVdmControl

Scan every physical page frame with offset **0xA70** for NtVdmControl bytes

Every physical page frame with offset **0xA70** is guaranteed to have the same data as VA

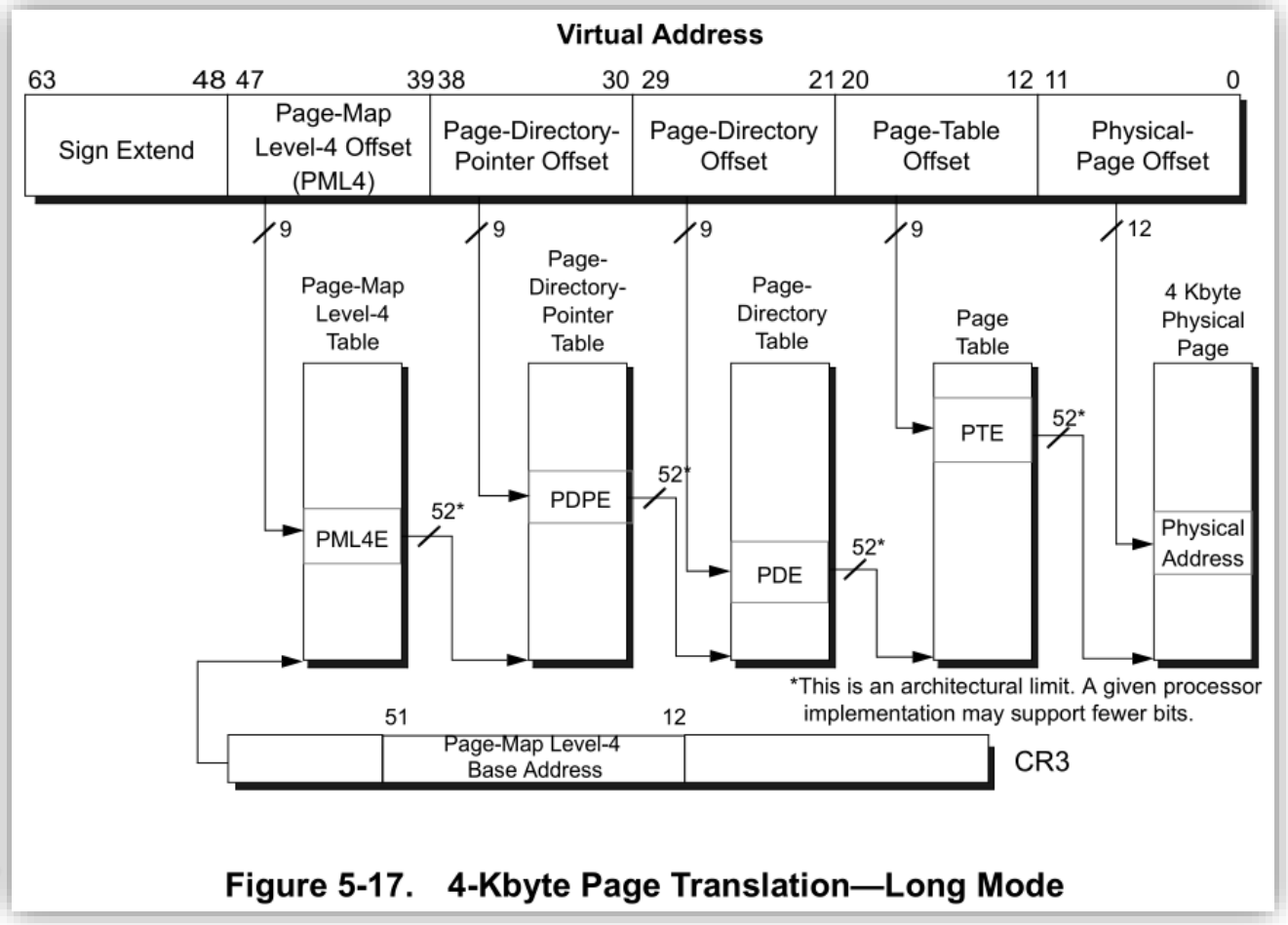


Figure 5-17. 4-Kbyte Page Translation—Long Mode

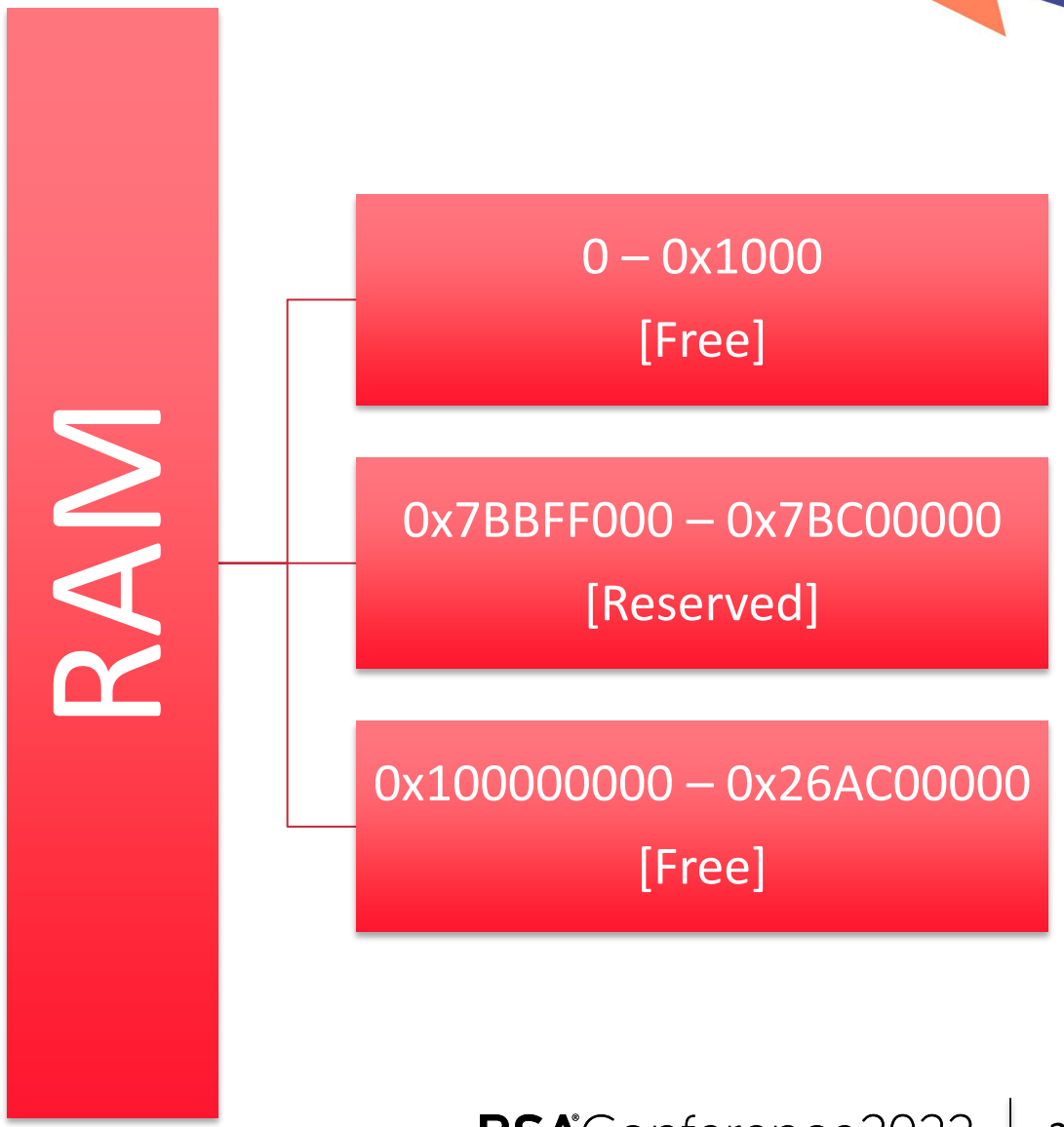
AMD64 paging abuse — cont.

- Probe individual machine words
  - Faster than creating MDLs and userspace mappings
  - Immediate result returned
- Corsair driver provides this functionality via another IOCTL ☺

```
mapAddress = MmMapIoSpace((PHYSICAL_ADDRESS)PhysicalAddress.LowPart, size, MmNonCached);
if ( !mapAddress )
  return 0xC0000017i64;
v10 = 0;
if ( !a2 )
{
  switch ( size )
  {
    case 1u:
      *(_BYTE *)sysBuffer = *mapAddress;
      goto LABEL_21;
    case 2u:
      *(_WORD *)sysBuffer = *(_WORD *)mapAddress;
      goto LABEL_21;
    case 4u:
      *(_DWORD *)sysBuffer = *(_DWORD *)mapAddress;
      goto LABEL_21;
  }
  goto LABEL_17;
}
```

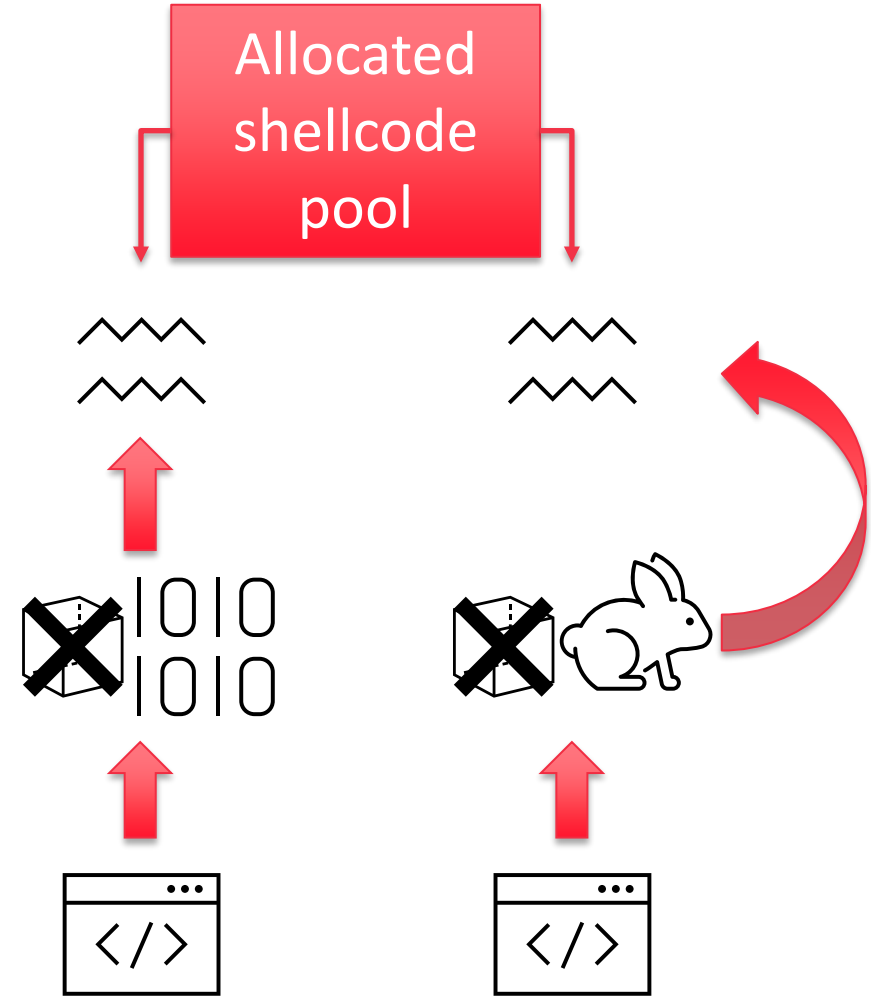
AMD64 paging abuse — cont.

- Search for “hot” signature
  - Prevents landing on other kernel image mappings
- Skip reserved RAM regions
  - Increases speed
  - No BSOD
  - Only searching RAM used by OS
- Query memory ranges via undocumented OS system call
  - NtQuerySystemInformation with **SystemMemoryTopologyInformation**



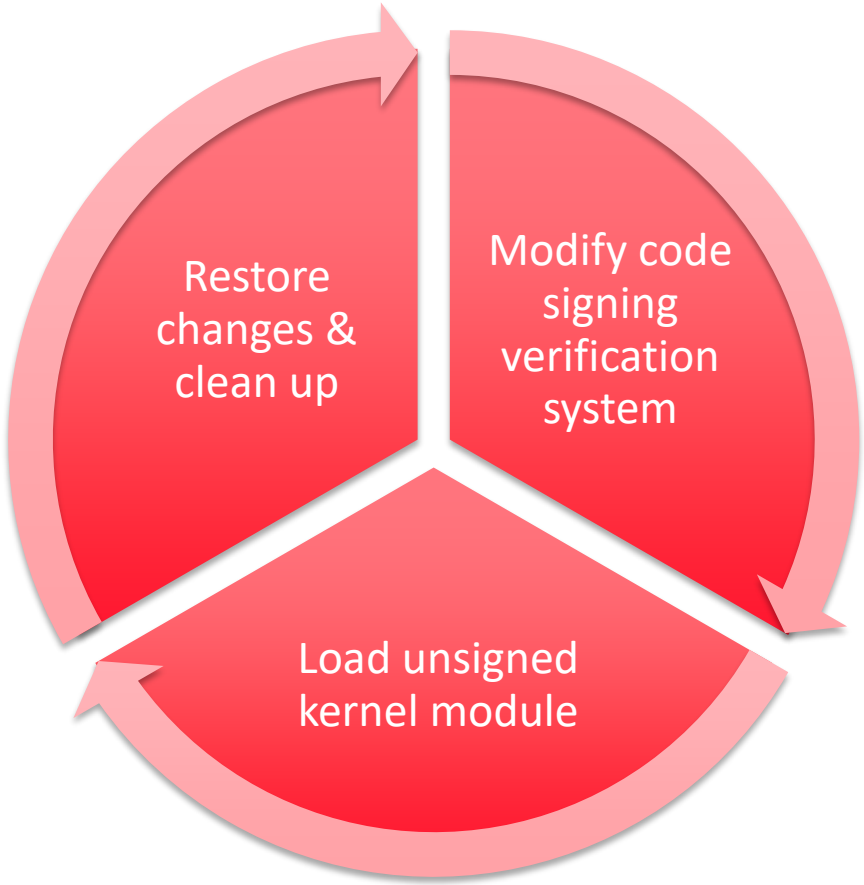
Shellcode design

- Once NtVdmControl PA is found, it can be mapped, and shellcode can be written
  - Write & execute “allocator” shellcode
  - Allocator allocates executable memory from nonpaged pool to hold real shellcode
- Shellcode is just a function in attack program
  - Uses a table of pointers to call kernel functions
  - “Copy function” from attack process to kernel memory
  - Eliminates need for ASM
- Syscall is “hijacked”
  - It is now “our system call,” we can pass anything we want into it, including userspace pointers



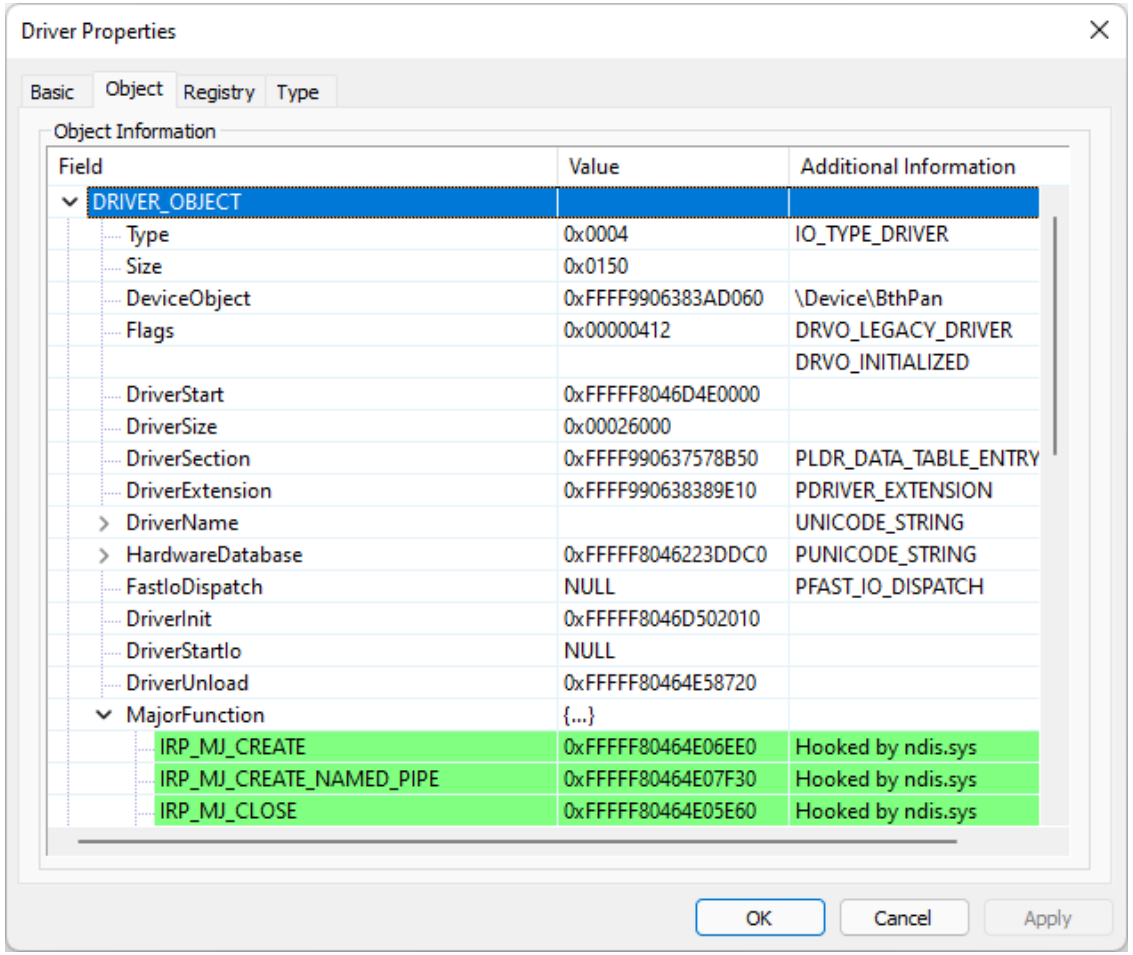
Shellcode design — loading unsigned code

- Load unsigned code in form of driver
  - MmLoadSystemImage is the best way to properly load a module
  - Function will fail because image is unsigned
- Bypassing image signature enforcement
  - Overwrite internal kernel function that checks image signing
    - Modify **MiValidateSectionSigningPolicy**
    - Always return success
  - Use MDLs to modify the physical memory backing the kernel function



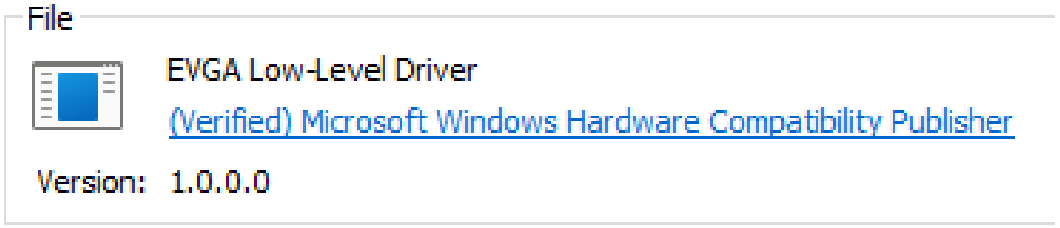
Shellcode design — final steps

- All drivers have a driver object
  - Create one manually due to nonstandard driver loading
- Shellcode calls IoCreateDriver
  - Private (but exported) kernel function
  - Function also calls driver's entry point
- After this step, shellcode is finished
  - Cleanup MDL mappings and return
  - Write jump to ExFreePoolWithTag to free allocated shellcode memory



Another vulnerable driver?

- Are there more drivers like this on my system?
  - Found the EVGA X1 driver
- EVGA driver analysis
  - Like Corsair LL Access
  - Lacks userspace memory mappings (probably for the better)
  - We can still probe/write individual bytes to I/O space
- Modified exploit to work with EVGA X1 driver



HAL.dll	HalGetBusDataByOffset	17
HAL.dll	HalSetBusDataByOffset	37
ntoskrnl.exe	IoCreateDevice	462
ntoskrnl.exe	IoCreateSymbolicLink	341
ntoskrnl.exe	IoDeleteDevice	351
ntoskrnl.exe	IoDeleteSymbolicLink	353
ntoskrnl.exe	IoDeviceObjectType	489
ntoskrnl.exe	IoIsWdmVersionAvailable	556
ntoskrnl.exe	IoRegisterShutdownNotification	433
ntoskrnl.exe	IoUnregisterShutdownNotification	474
ntoskrnl.exe	IofCompleteRequest	502
ntoskrnl.exe	KeBugCheckEx	531
ntoskrnl.exe	MmGetSystemRoutineAddress	959
ntoskrnl.exe	MmMapIoSpace	720

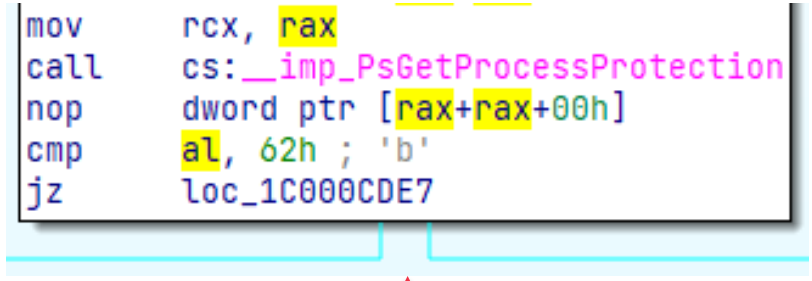
Case study #2

Goal: read isolated memory from an IUM process using only built-in Windows components

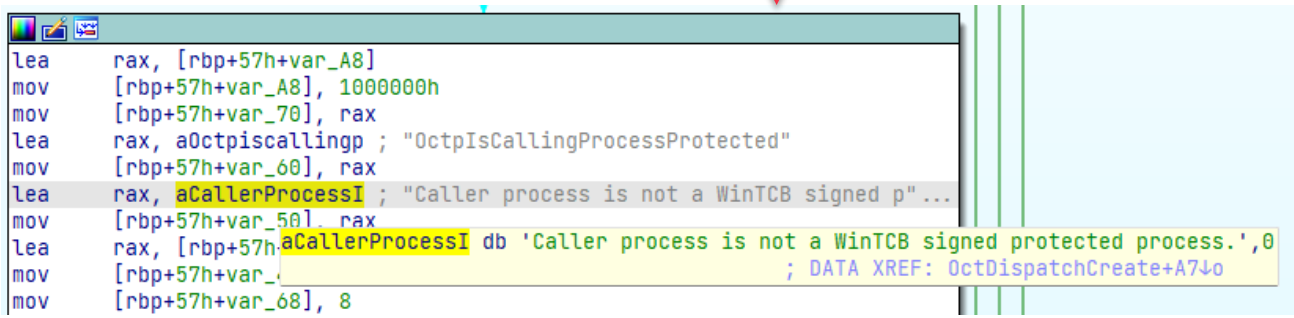
VBS is turned ON

Choosing a component to abuse

- Requirements
  - Low-level memory manipulation
  - Accepts parameters from userspace
  - No hardened API usage
- Where to look
  - ~~System protection/guard runtime~~
    - Uses PPL process protection
  - ~~KMDF drivers~~
    - Non-user processing
  - Antivirus/Windows Defender
    - Maybe



Checks for WinTCB signing upon device HANDLE creation



Windows Defender

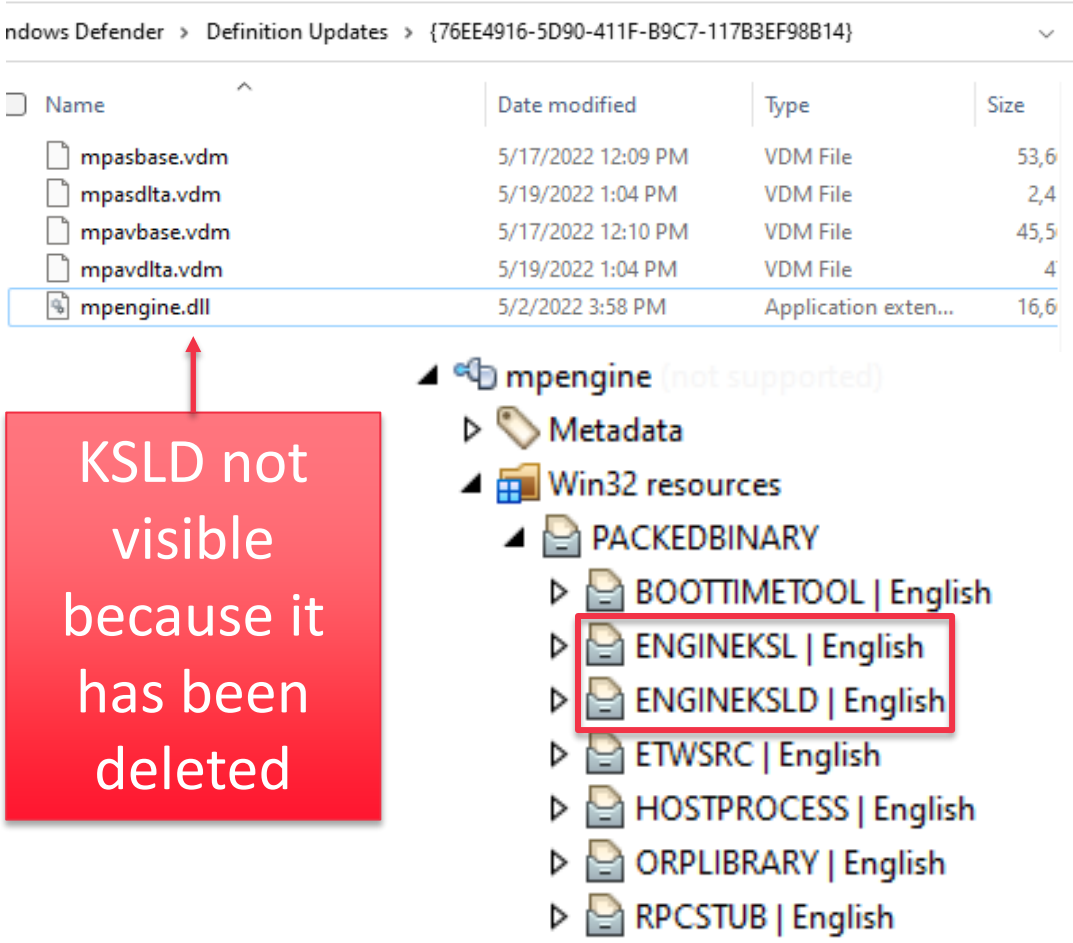
- Several kernelspace components
  - Main kernel driver *WdFilter.sys*
  - Boot-time, device filter, and network inspection drivers
- Interesting behavior
  - Randomly creates kernel driver service *MpKslXxx.sys*
    - *Usually during new driver load or definition updates*
  - *Service deleted randomly after some time*
  - Can we use this?

Name	Display name	Type
WinDefend	Microsoft Defender Antivirus Service	Own process
WdNisSvc	Microsoft Defender Antivirus Networ...	Own process
WdNisDrv	Microsoft Defender Antivirus Networ...	Driver
WdFilter	Microsoft Defender Antivirus Mini-Fil...	FS driver
WdBoot	Microsoft Defender Antivirus Boot Dri...	Driver
mpssvc	Windows Defender Firewall	Share process
mpsdrv	Windows Defender Firewall Authoriza...	Driver
BFE	Base Filtering Engine	Share process

```
Process created: wuauclt.exe (6512) started by svchost.exe (13200)
Process created: AM_Delta_Patch_1.367.123.0.exe (14776) started by wuauclt.exe (6512)
Process created: MpSigStub.exe (3820) started by AM_Delta_Patch_1.367.123.0.exe (14776)
Process created: firefox.exe (11492) started by firefox.exe (5784)
Process terminated: dllhost.exe (11780); exit status 0
Process terminated: wuauclt.exe (6512); exit status 0
Process terminated: AM_Delta_Patch_1.367.123.0.exe (14776); exit status 0
Process terminated: MpCmdRun.exe (2576); exit status 0
Process terminated: conhost.exe (11968); exit status 0
Process terminated: MpSigStub.exe (3820); exit status 0
Service modified: WdNisDrv (Microsoft Defender Antivirus Network Inspection System Driver)
Process created: NisSrv.exe (13224) started by services.exe (800)
Service deleted: MpKslcdd61d2f (MpKslcdd61d2f)
Service started: WdNisDrv (Microsoft Defender Antivirus Network Inspection System Driver)
Service started: WdNisSvc (Microsoft Defender Antivirus Network Inspection Service)
```

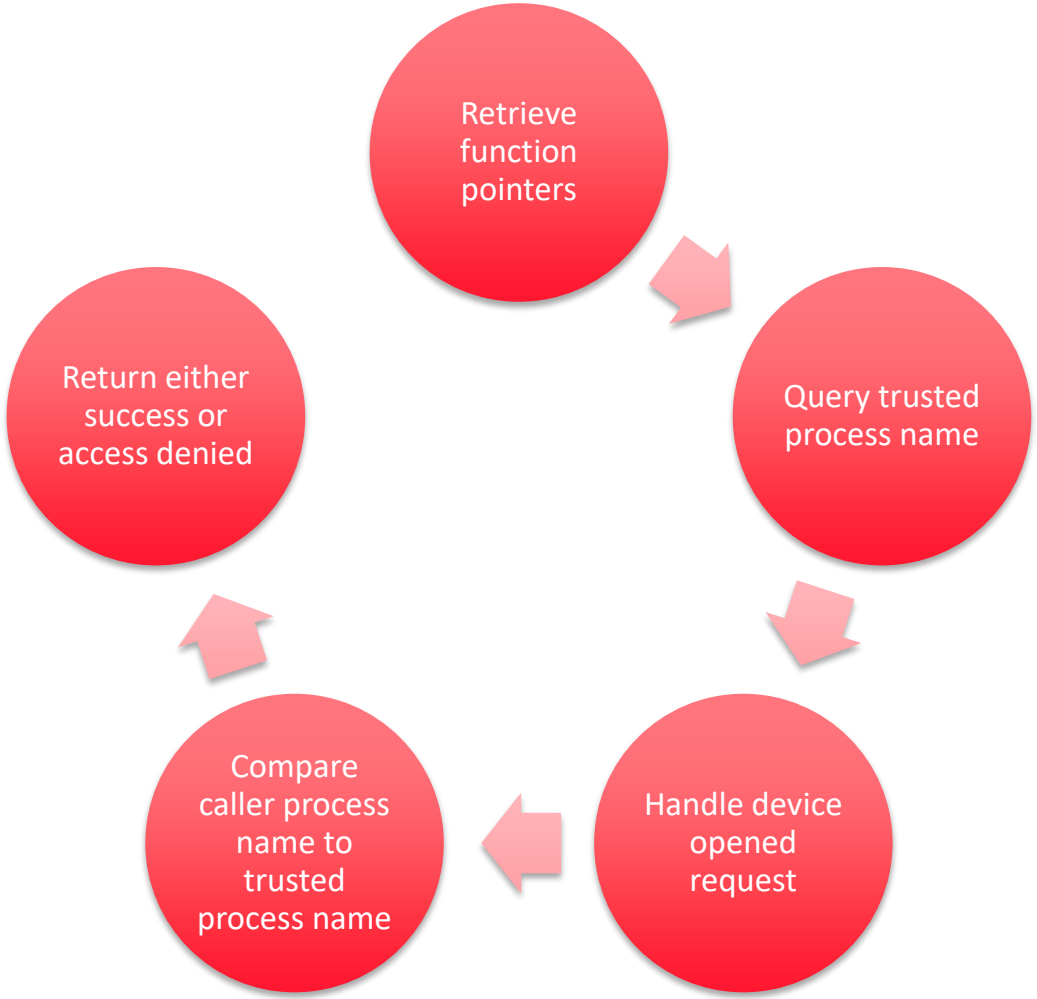
Digging into KSLD

- Extracted alongside definition update files
  - Extracted from *mpengine.dll*
    - Also contains antivirus process binary (*MsMpEng(CP).exe*)
- Driver is a KMDF (WDF) driver
  - Harder to analyze
  - Most handling performed by WDF
  - *mpengine.dll* has an equivalent WDM version available ☺
    - Called *KSL* instead of *KSLD*



Analyzing KSL

- Startup behavior
  - Retrieves pointers to *MmCopyMemory* based on OS version
    - Retrieves other function pointers
  - Retrieves registry parameters
    - Allowed process name, version, etc.
  - Securely creates device object
- Upon device open, get process name
  - If name != registry allowed name value, return *STATUS_ACCESS_DENIED*



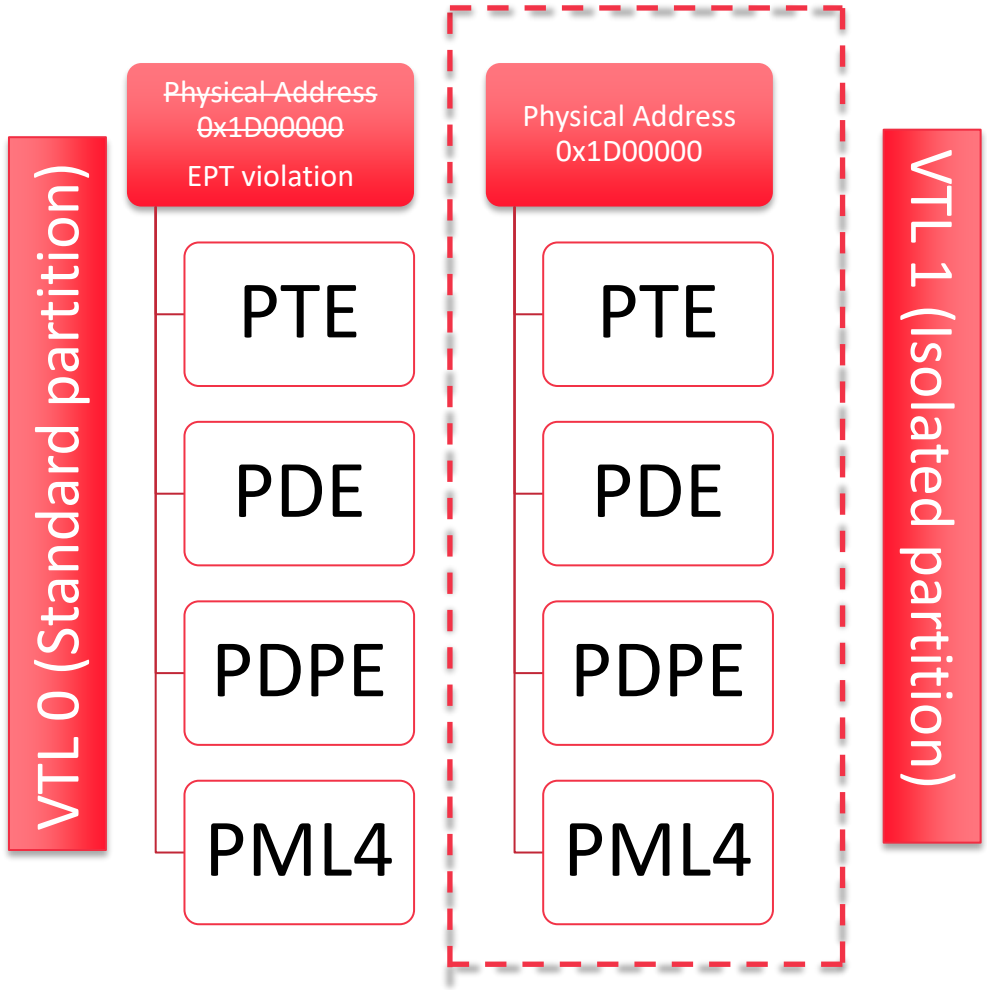
Analyzing KSL — cont.

- **PLENTY** of IOCTL functionality
  - Read/Write physical memory **MmMapIoSpace**
  - Read physical memory **ZwMapViewOfSection**
    - Possible to map as Read+Write
    - Not a hardened routine!
  - Read virtual & physical memory **MmCopyMemory**
    - Can probe arbitrary virtual & physical addresses
    - Read lots of memory at once
    - Only available on versions >= 8.1
  - Retrieve address of kernel functions **MmGetSystemRoutineAddress**
  - Open file HANDLES with kernelspace-level permissions **IoCreateFile**
  - Read specific CPU registers **including system registers**
  - And more

```
v6[1] = *(v5 + 1);
v6[2] = *(v5 + 2);
v6[3] = *(v5 + 3);
v10 = __readcr0();
v4[29] = v10;
v11 = __readcr2();
v4[31] = v11;
v12 = __readcr3();
v4[33] = v12;
v13 = __readcr4();
v4[35] = v13;
__sidt(v4 + 39);
v4[39] = v4[39] >> 16;
v4[51] = __readmsr(0x176u);
```

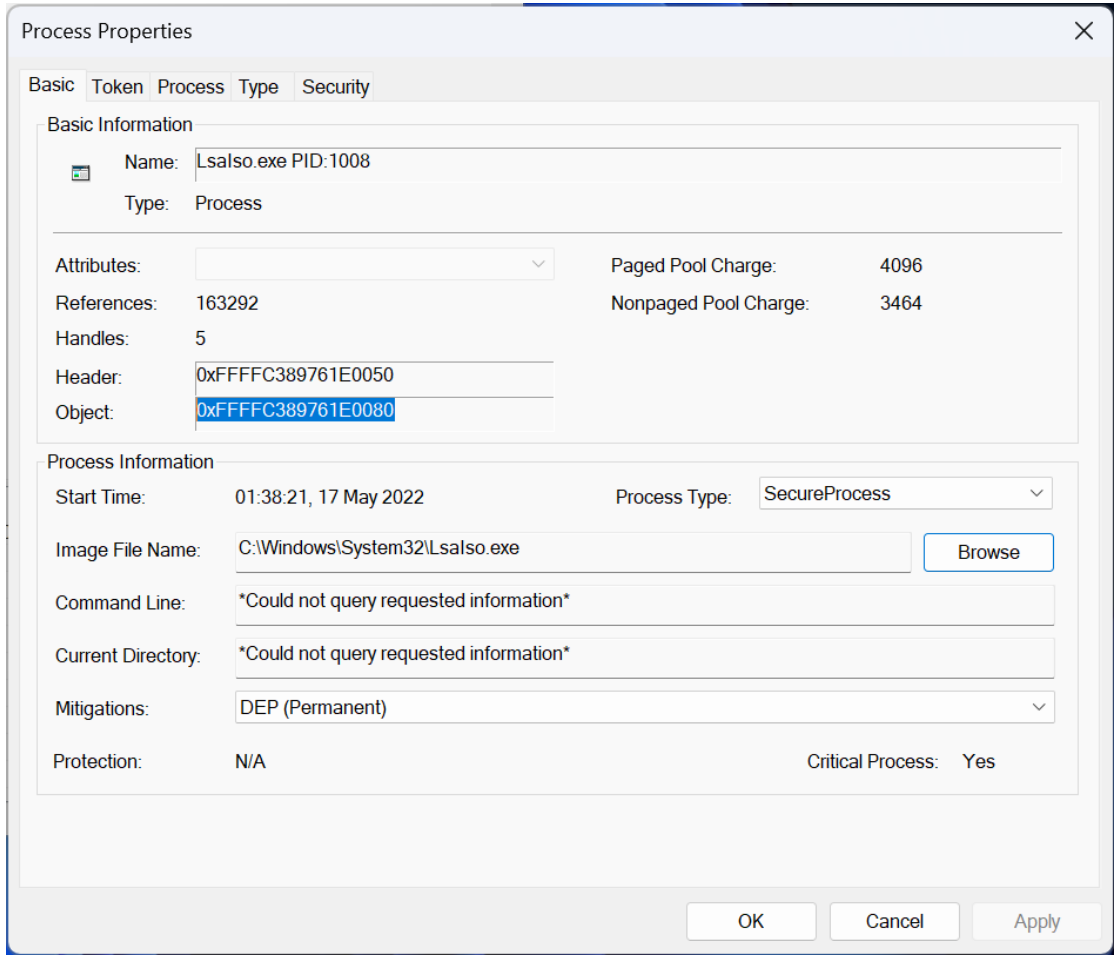
Designing an exploit

- ~~Execute custom unsigned code~~
  - VBS is on which causes problems
- Read isolated memory from an IUM process
  - Maybe
  - Need to develop a method
- IUM memory isolation
  - Hypervisor prevents reads
  - Not even kernel debugger can read memory
  - Only IUM process with selected PML4 base can read



Designing an exploit — cont.

- Use IUM process PML4
  - Stored in kernel KPROCESS object
  - Use **NtQuerySystemInformation** with **SystemHandleInformation** to get VA of object
- Parse EPROCESS in kernelspace using KSLD
  - Win 8.1 & Greater: use **MmCopyMemory** with VA
  - Otherwise, read current CPU's CR3 register
    - CPU's CR3 equals the current process's PML4 base address
    - Kernel is mapped into every process' page tables
    - Use **ZwMapViewOfSection** to convert VA to PA and extract information

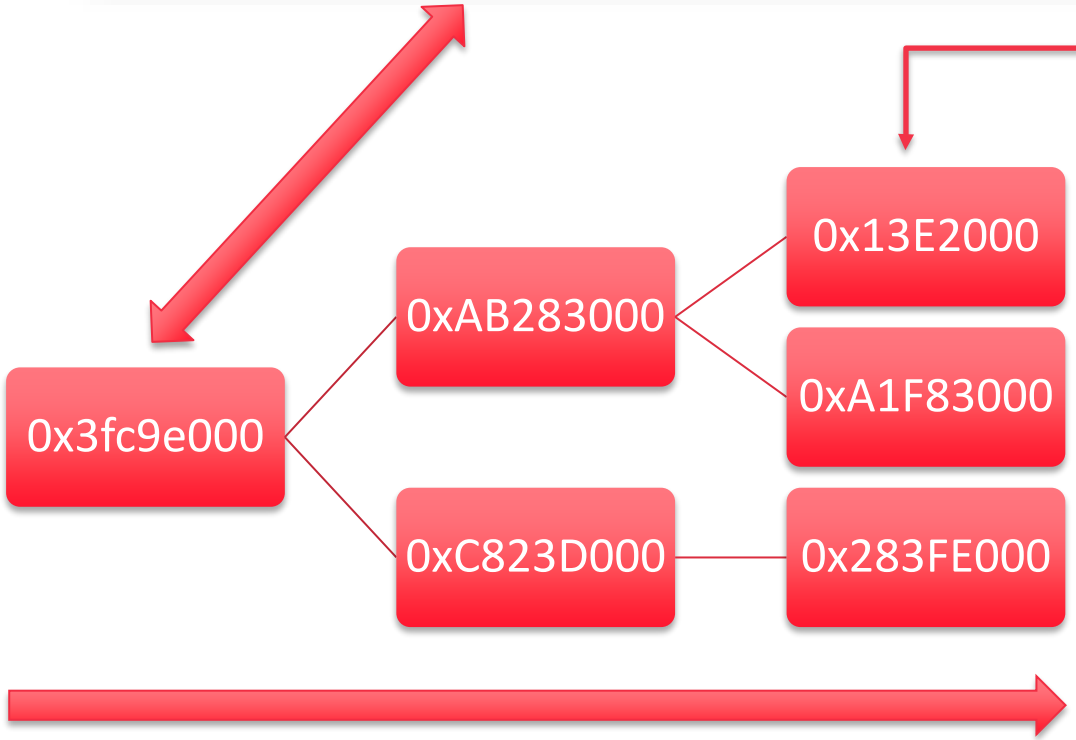


Reading IUM memory

```
0: kd> dx -id 0,0,ffffc389761e0080 -r1 (*((ntkrnlmp!_KPROCESS *)0xffffc
(*((ntkrnlmp!_KPROCESS *)0xffffc389761e0080))                    [Type: _K
    [+0x000] Header           [Type: _DISPATCHER_HEADER]
    [+0x018] ProfileListHead  [Type: _LIST_ENTRY]
    [+0x028] DirectoryTableBase : 0x3fc9e000 [Type: unsigned __int64]
```

Use IUM process PML4 to walk page tables **MmCopyMemory** with PA or **ZwMapViewOfSection** to read physical memory

Use **VirtualQuery** to enumerate private | commit memory regions and dump them using this read method



Lock the pages in memory with **NtLockVirtualMemory** to ensure the PTEs are valid

Base address	Type	Size	Protection	Private
0x7ffe0000	Private	4 kB	R	4 kB
0x7ffe0000	Private: Commit	4 kB	R	4 kB
0x7ffe9000	Private	4 kB	R	4 kB
0x7ffe9000	Private: Commit	4 kB	R	4 kB
0x81ea200000	Private	2 MB	RW	4 kB
0x81ea229000	Private: Commit	4 kB	RW	4 kB
0x1955d510000	Private	128 kB	RW	128 kB
0x1955d510000	Private: Commit	128 kB	RW	128 kB
0x1955d530000	Private	124 kB	RW	
0x1955d550000	Private	76 kB	RW	
0x1955d580000	Private	512 kB	RW	24 kB
0x1955d5fa000	Private: Commit	24 kB	RW	24 kB
0x1955d600000	Private	64 kB	RW	12 kB
0x1955d600000	Private: Commit	12 kB	RW	12 kB
0x1955d610000	Private	200 kB	RW	8 kB
0x1955d610000	Private: Commit	8 kB	RW	8 kB
0x1955d650000	Private	1 MB	RW	132 kB
0x1955d650000	Private: Commit	116 kB	RW	116 kB
0x1955d68c000	Private: Commit	12 kB	RW	12 kB
0x1955d690000	Private: Commit	4 kB	RW	4 kB

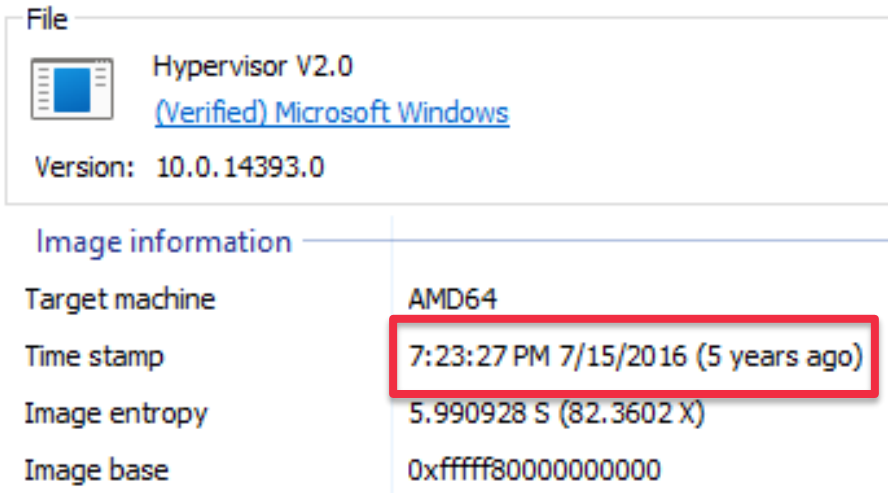
Reading IUM memory — cont.

PEB/TEB (has 0xFFs followed by image base address)

Other IUM data

KSLD exploit effectiveness

- High availability
  - Windows 10+ has Windows Defender included
  - *MpKslXxx* most likely already installed or running
    - If not, extract from latest definition update *mpengine.dll*
    - Otherwise, use existing driver with exploit
- Discrete memory access
  - Hypervisor's EPT does not prevent reads
  - *"Windows Defender"* is reading memory/installing driver
    - Already commonly does this
  - Nothing recorded in secure kernel's security log



Hypervisor from Windows 10 1607 (Anniversary Update)

Takeaways & next steps

- Treat signed artifacts as foreign
  - Even “trusted, quality signed code”
  - Static analysis and runtime behavior verification is a must if external code should be used in an enterprise
- Host systems need proper auditing
  - Built-in components can be abused
    - Even parts of Antivirus (as you saw)
  - If security is critical, all high-privilege components must undergo the same static analysis and runtime behavior verification that all external components go through
- Zero external dependencies is the best goal
  - Along with host system auditing
- Over the next several months, organizations should audit their high-security software dependencies
  - External dependencies and parts of the host system
- Over the next year, organizations can slowly move towards alternate solutions to ensure security across all privilege/integrity levels

RSAConference2022

Thank you for your time

It’s time for any questions!

GitHub/Twitter — @AzAgarampur/@axagarampur