ENSILO

TURNING (PAGE) TABLES

BYPASSING ADVANCED KERNEL MITIGATIONS USING PAGE TABLES MANIPULATIONS

BSidesLV 2018

ABOUT US

- Omri Misgav
  - Security Research Team Leader @ enSilo
  - Reverse Engineering, OS internals and malware research
  - Past speaker in BSidesLV
- Udi Yavo
  - CTO & Co-Founder @ enSilo
  - Former CTO, Rafael Cyber Security Division
  - Past speaker in Blackhat, RSA and BSidesLV
- Our technical blog: BreakingMalware.com

ENSILO

AGENDA

- Windows 10 Kernel Exploit Mitigations
- Memory Management Overview
- Virtualization, VBS & KMCI
- Turning Tables Technique
- Demo
- Mitigations

ENSILO

WINDOWS 10 KERNEL EXPLOIT MITIGATIONS

- Microsoft puts a lot of effort into kernel mitigations
- This is only partial list of improvements:

Mitigation/OS	Windows 7	Windows 8.1	Windows 10	Windows 10 November Update	Windows 10 Redstone 1	Windows 10 Redstone 2	Windows 10 Redstone 3	Windows 10 Redstone 4
Safe Unlinking	X	X	X	X	X	X	X	X
NULL Page Allocation		X	X	X	X	X	X	X
Disable Win32k Syscalls[3]		X	X	X	X	X	X	X
KASLR[3]		X	X	X	X	X	X	X
SMEP		X	X	X	X	X	X	X
Page Table Randomization					X	X	X	X
GDI Pointers Removal					X	X	X	X
NULL SecurityDescriptor					X	X	X	X
UserHandleTable Stripping						X	X	X
HAL Heap Randomization						X	X	X
KCFG[1]						X	X	X
Win32k Type Isolation							X	X
KMCI[1,2]			X	X	X	X	X	X

1. Not enabled by default
2. Require VBS
3. Mitigations that constantly improved

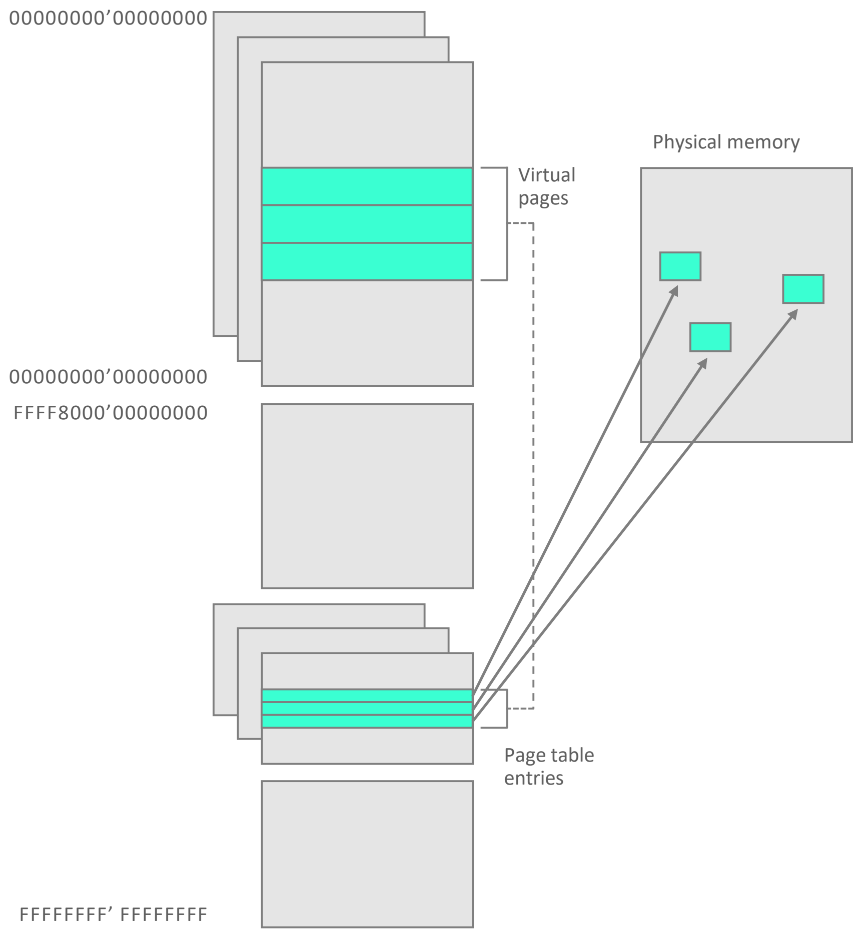
ENSILO

WINDOWS 10 KERNEL EXPLOIT MITIGATIONS

- Many new exploits techniques were developed to bypass these mitigations:
  - Taking Windows 10 Kernel Exploitation To The Next Level
  - Abusing GDI Objects for ring0 Primitives Revolution
  - Abusing GDI for ring0 exploit primitives
  - A New CVE-2015-0057 Exploit Technology (Vulnerability disclosed by us)
  - …
- Still in no generic exploitation methods with KMCI enabled
- Until now…

ENSILO

MEMORY MANAGEMENT OVERVIEW
Virtual memory



ENSILO

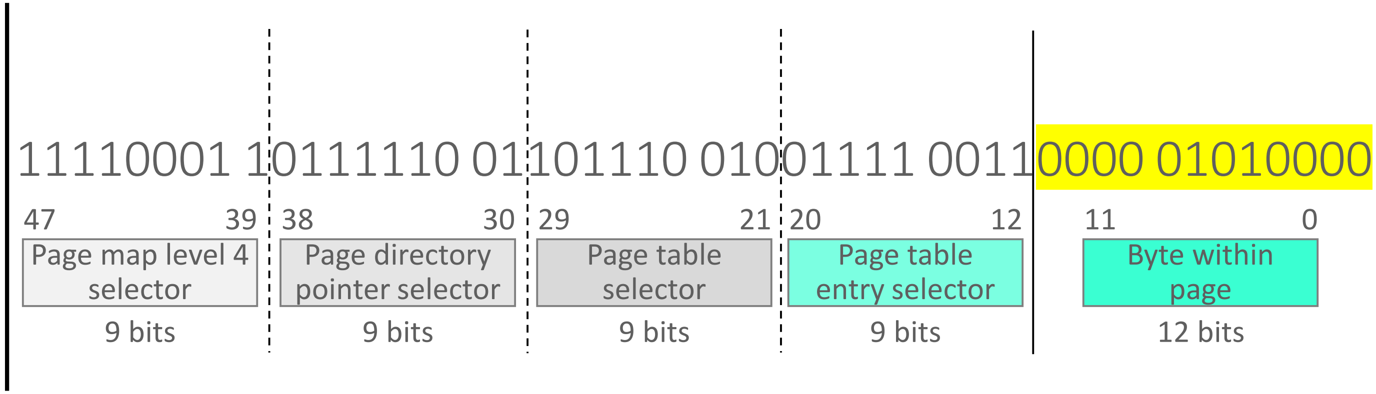
Source: Windows Internals 6th edition

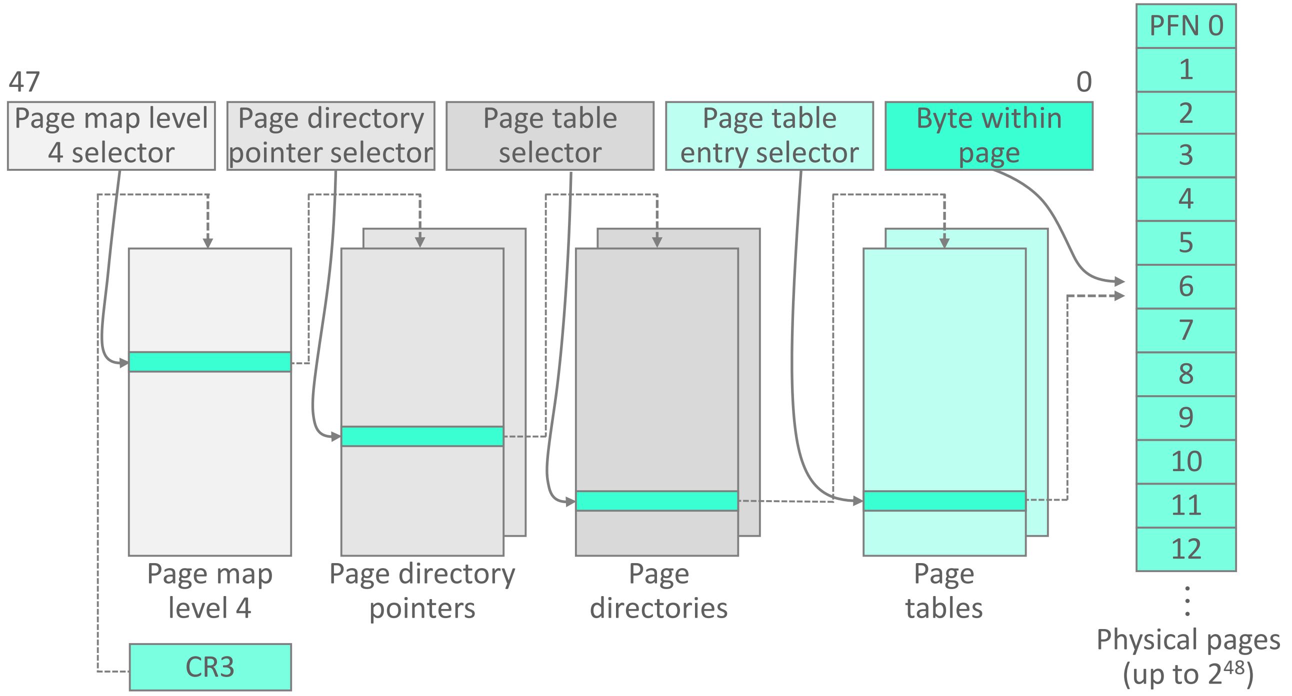
MEMORY MANAGEMENT OVERVIEW

Virtual memory address translation

VA: fffff1be`6e4f3050

Binary: *11111111 11111111* 11110001 10111110 01101110 01001111 0011**0000 01010000**

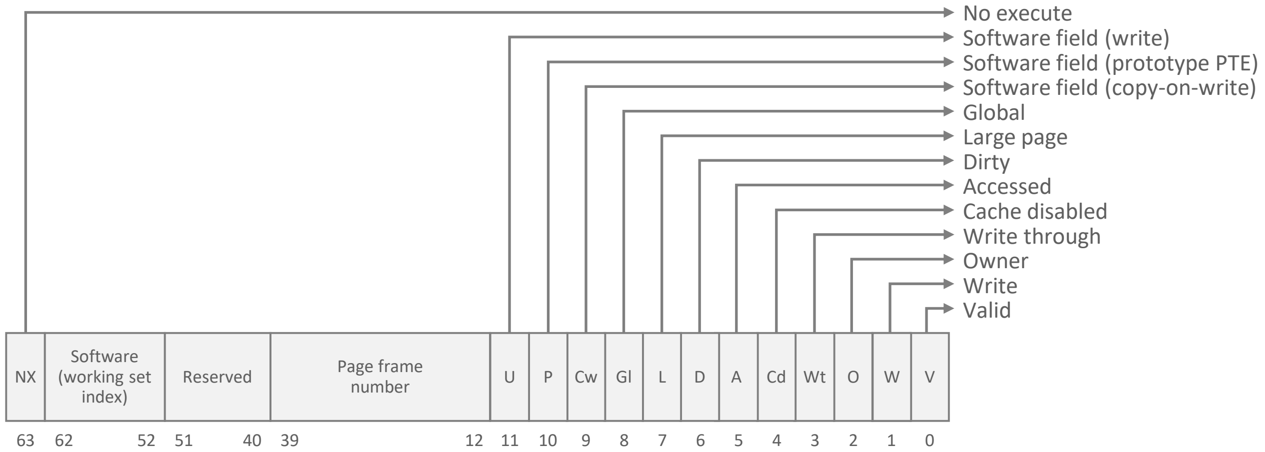




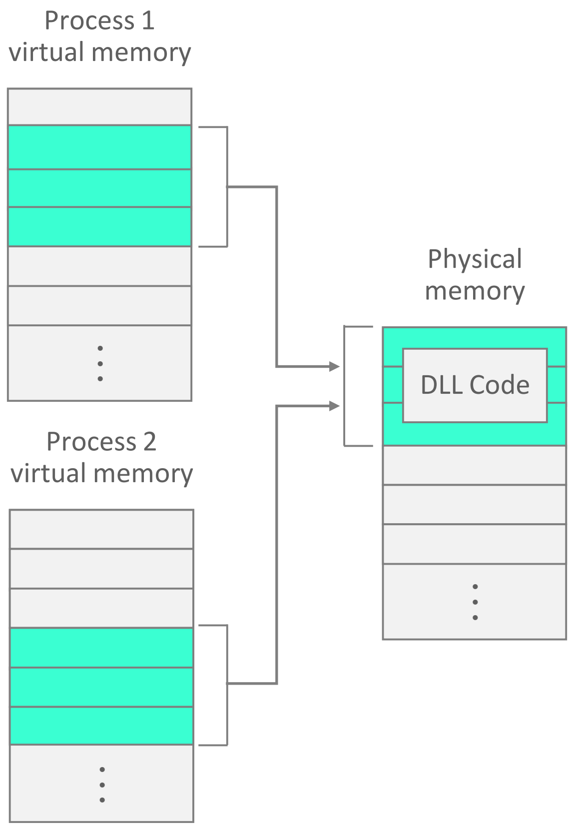
Source: Windows Internals 6th edition

MEMORY MANAGEMENT OVERVIEW

PTEs and shared memory



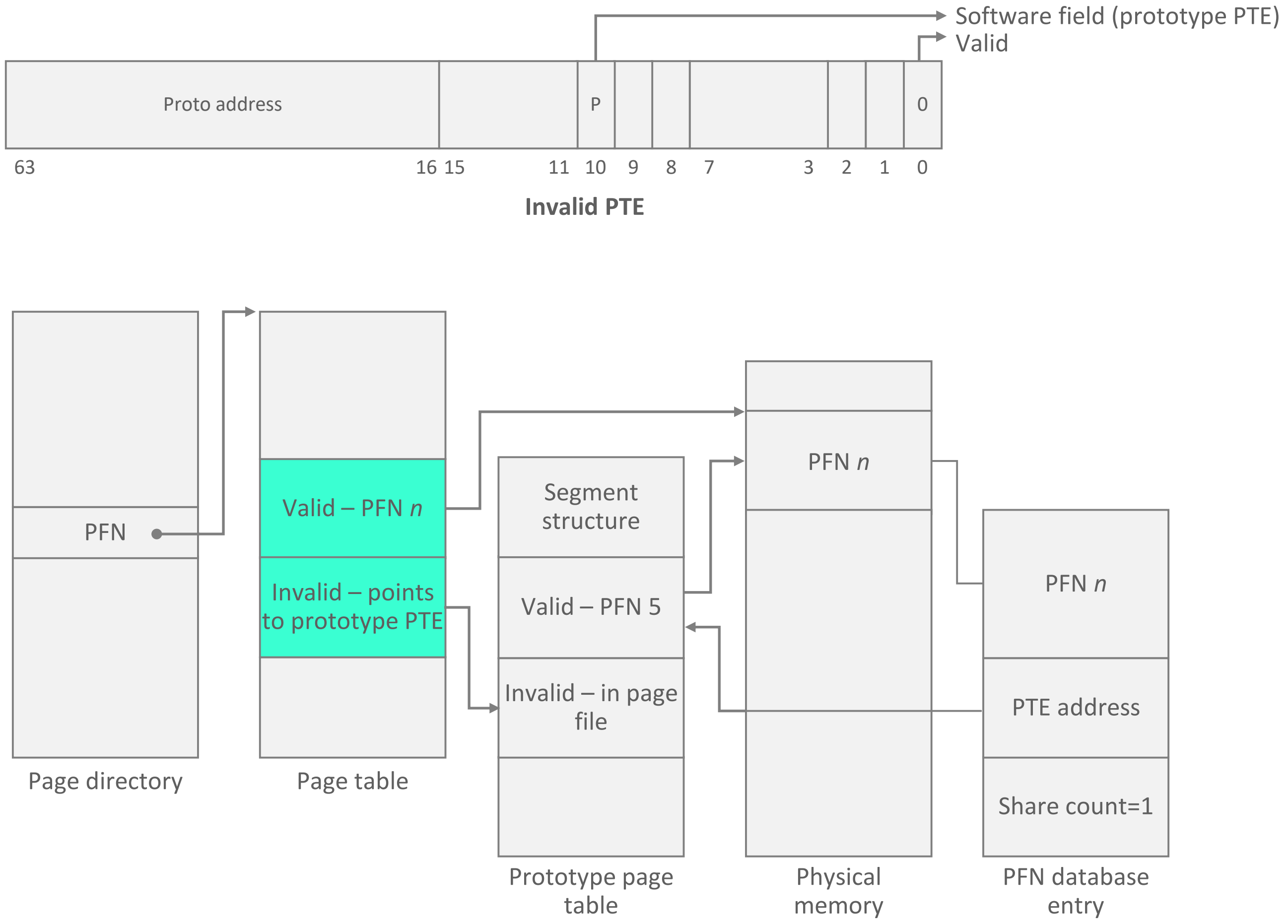
x64 PTE



ENSILO

Source: Windows Internals 6th edition

MEMORY MANAGEMENT OVERVIEW
Prototype PTEs

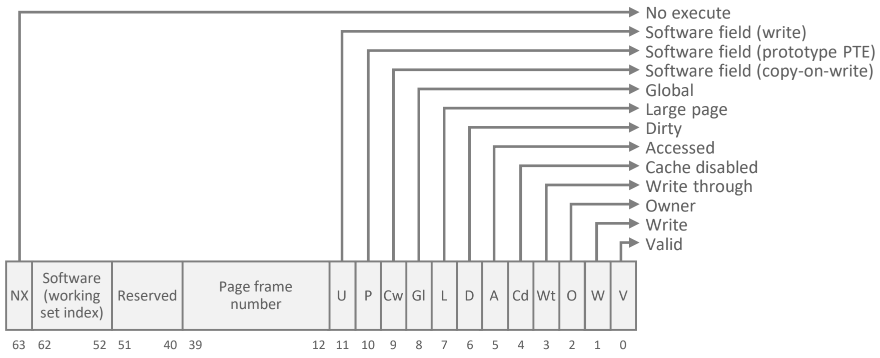


ENSILO

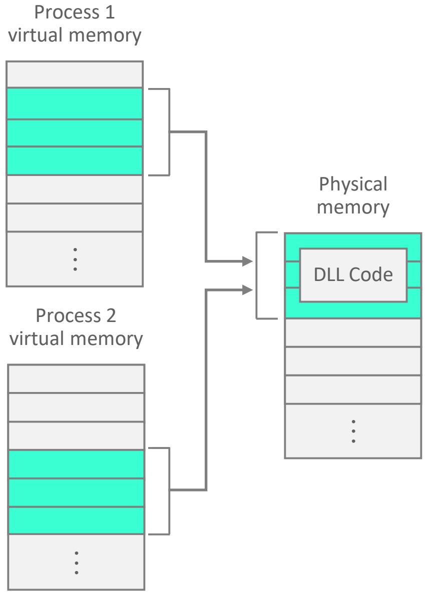
Source: Windows Internals 6th edition

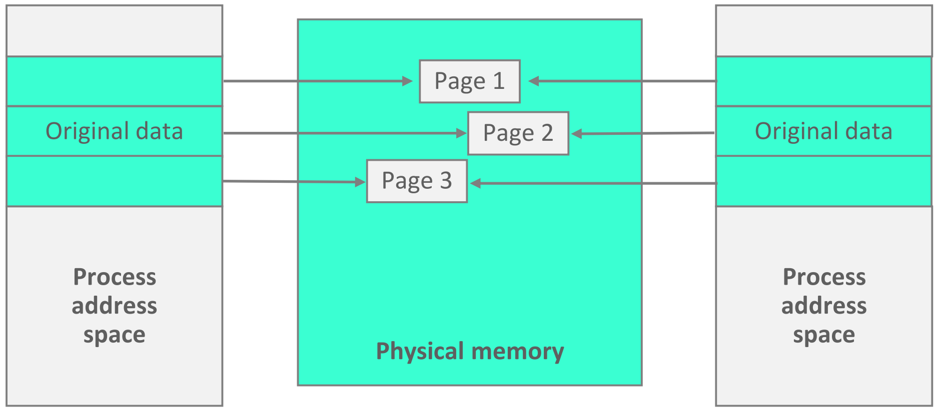
MEMORY MANAGEMENT OVERVIEW

Copy-on-Write

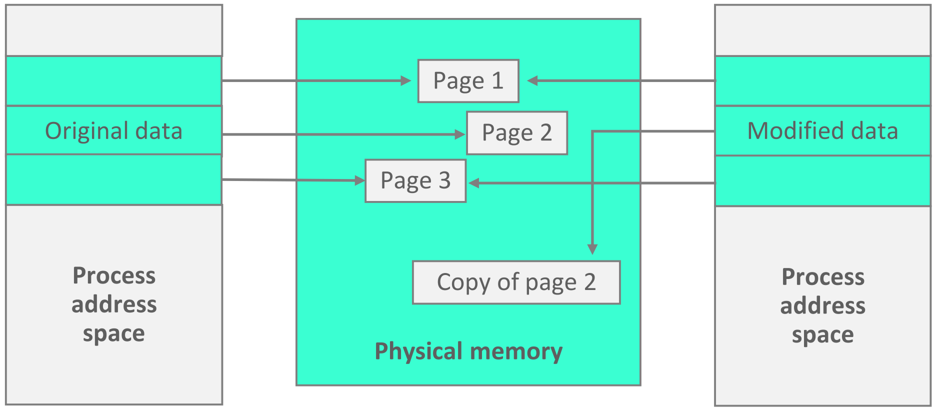


x64 PTE





Before



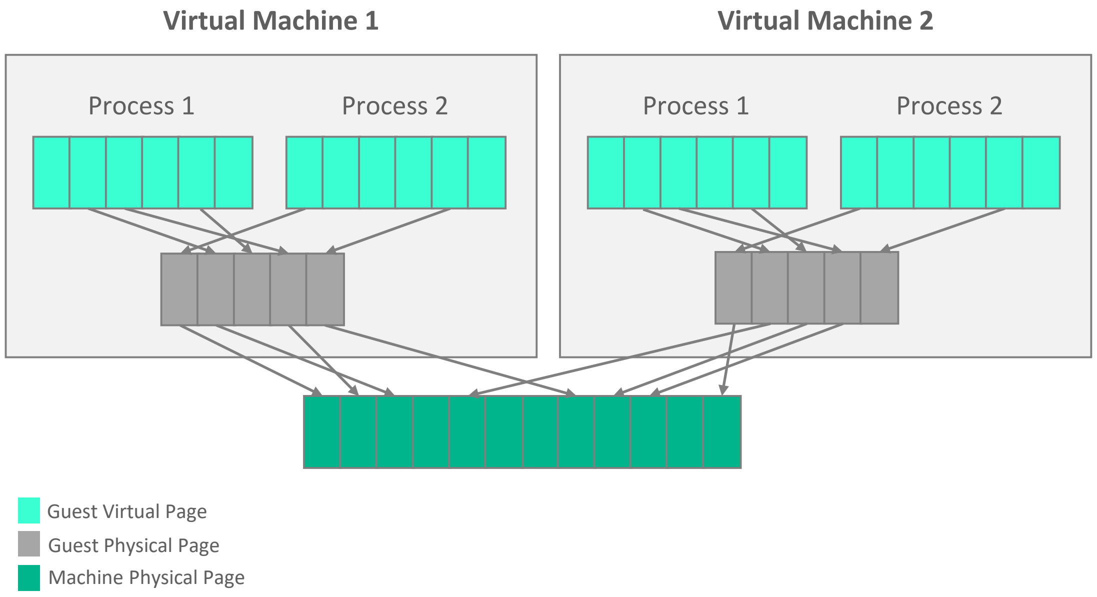
After

ENSILO

Source: Windows Internals 6th edition

MEMORY MANAGEMENT OVERVIEW
Virtualization

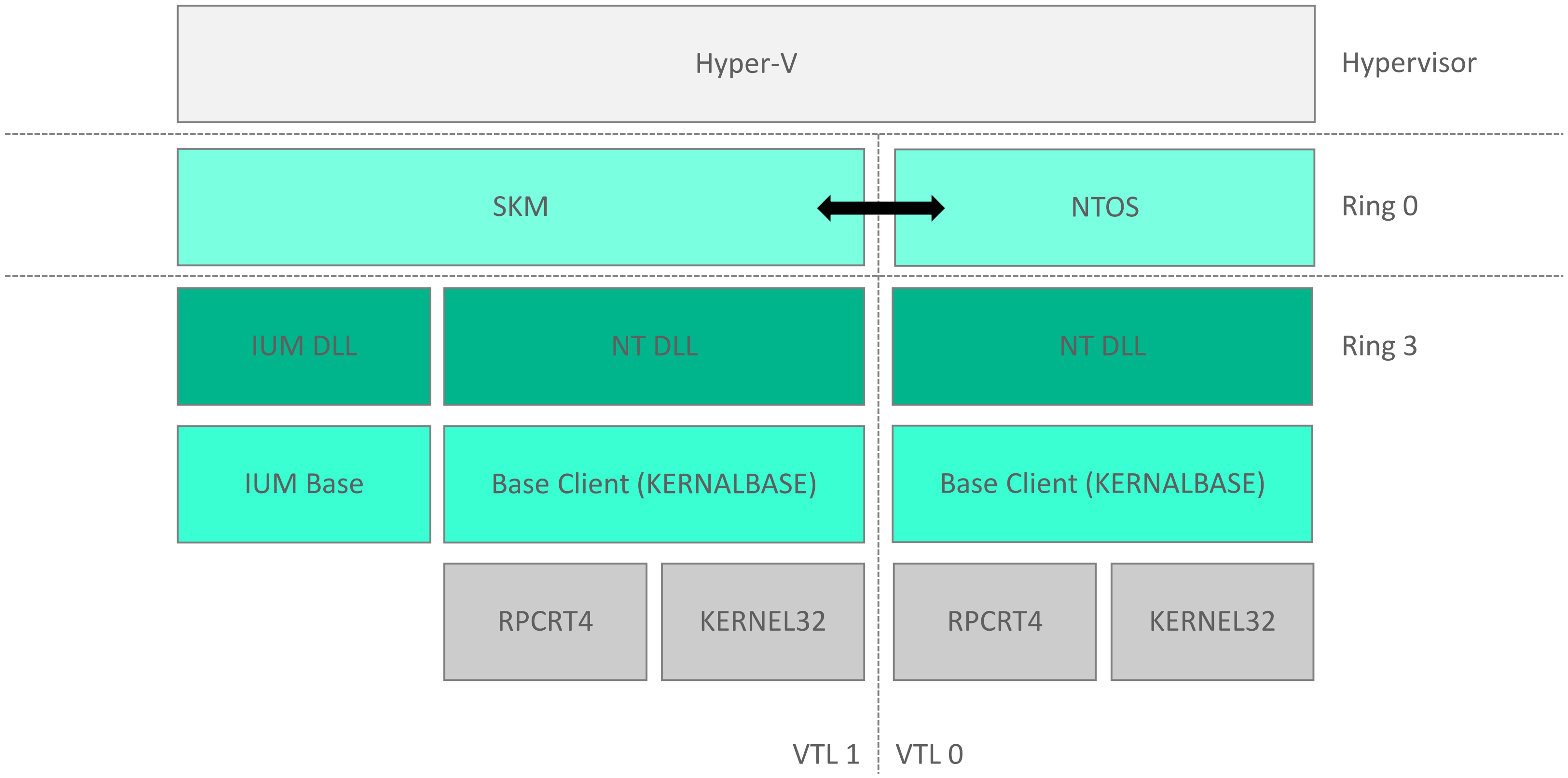
- Second Level Address Translation - SLAT
- Translation of Guest Physical Address (GPA) to Machine Physical Address (MPA)
- Same page table hierarchy: PML4 -> PDP -> PD -> PT




```
typedef struct _ept_pte {
        uint64 read               : 1;   // bits 2..0
        uint64 write              : 1;   // bits 2..0
        uint64 execute            : 1;   // bits 2..0
        uint64 ept_mt             : 3;   // bits 5..3
        uint64 ignore_pat_mt      : 1;   // bit 6
        uint64 is_large_page      : 1;   // bit 7
        uint64 accessed           : 1;   // bit 8
        uint64 dirty              : 1;   // bit 9
        uint64 user_execute       : 1;   // bit 10
        uint64 ignored1           : 1;   // bit 11
        uint64 pfn                : 40;  // bits 51..12
        uint64 ignored2           : 11;  // bits 62..52
        uint64 supress_ve         : 1;   // bit 6
} ept_pte;
```

VIRTUALIZATION-BASED SECURITY

Architectural overview



ENSILO

VIRTUALIZATION-BASED SECURITY

Kernel-Mode Code Integrity (KMCI)

- HVCI - HyperVisor Code Integrity
- Blocking +RX / +RWX
  - Preventing execution of code, or modification of code
- Blocking +W
  - Preventing modification of executable pages shared with VTL 1

- SKCI.DLL (Secure Kernel Code Integrity)
  - Same functionally of CI.DLL, the normal world Code Integrity library
- Upon loading a new driver the Secure Kernel is invoked in order to validate the digital signature and check it’s authorized within the current policy

ENSILO

QUICK RECAP

- Virtual memory management is a joint effort by hardware and software
- Virtual memory is the foundation for many important OS capabilities
  - Shared memory
  - Flexible physical memory management
  - …

- Microsoft leverages virtualization hardware capabilities to enhance security
  - HVCI: Raises the bar for exploitation
  - Credential Guard
  - Secure memory enclaves
  - …

GOALS AND MOTIVATION

- Most privilege escalation exploits runs a payload in kernel-mode in their course of action
- KMCI effectively prevents it
  - New kernel code can't be allocated if unsigned
  - Existing kernel code cannot be modified

- Previous publications assume KMCI is disabled (except data only attacks)
- The real goal of most kernel exploits is to run code with highest possible privileges

- Basically, we want to achieve arbitrary code execution with system privileges
  - *"…a place where architecturally, we do not currently define a defensible security boundary."*

TURNING TABLES TECHNIQUE

Prerequisites

- Essentially only read/write primitives are needed
- This is common in every modern exploit

- And that’s it ☺

ENSILO

TURNING TABLES TECHNIQUE
Outline

- Make a user-mode shared code page PTE writable in our process
  - Which typically runs also in system processes
  - Simply flipping a bit, remember?
- Change the code
- Wait…
- …And run as SYSTEM

ENSILO

TURNING TABLES BUILDING BLOCKS

Bypassing page-table randomization

- Assuming you already leaked NTOSKRNL.exe base address
- MmGetVirtualForPhysical
  - Exported and contains the PTE base address
  - The constant value is different in memory
- Additional method can be through MiGetPteAddress
  - Presented in Blackhat 2017

```
; Exported entry 1306. MmGetVirtualForPhysical



public MmGetVirtualForPhysical
MmGetVirtualForPhysical proc near
mov     rax, rcx
shr     rax, 0Ch
lea     rdx, [rax+rax*2]
add     rdx, rdx
mov     rax, 0FFFFFA8000000008h
mov     rax, [rax+rdx*8]
shl     rax, 19h
mov     rdx, 0FFFFF68000000000h
shl     rdx, 19h
and     ecx, 0FFFh
sub     rax, rdx
sar     rax, 10h
add     rax, rcx
retn
MmGetVirtualForPhysical endp
```

TURNING TABLES BUILDING BLOCKS

Finding targets

- Quite a few processes runs as user SYSTEM
  - svchost.exe
  - winlogon.exe, lsass.exe
  - MsMpEng.exe (Windows Defender) and most AVs…
- We can also use non-SYSTEM process with higher privileges
- Running in such processes may allow to avoid detection by some security products as they are excluded from monitoring due to performance/stability issues

winlogon.exe	720	NT AUTHORITY\SYSTEM
wininit.exe	672	NT AUTHORITY\SYSTEM
vmtoolsd.exe	2292	NT AUTHORITY\SYSTEM
vmacthlp.exe	1584	NT AUTHORITY\SYSTEM
VGAuthService.exe	2328	NT AUTHORITY\SYSTEM
TrustedInstaller.exe	220	NT AUTHORITY\SYSTEM
System Idle Process	0	NT AUTHORITY\SYSTEM
System	4	NT AUTHORITY\SYSTEM
svchost.exe	916	NT AUTHORITY\SYSTEM
svchost.exe	1096	NT AUTHORITY\SYSTEM
svchost.exe	1160	NT AUTHORITY\SYSTEM
svchost.exe	1668	NT AUTHORITY\SYSTEM
svchost.exe	2196	NT AUTHORITY\SYSTEM
svchost.exe	788	NT AUTHORITY\SYSTEM
spoolsv.exe	1964	NT AUTHORITY\SYSTEM
smss.exe	504	NT AUTHORITY\SYSTEM
SgrmBroker.exe	3148	NT AUTHORITY\SYSTEM
services.exe	800	NT AUTHORITY\SYSTEM
SecurityHealthService.exe	2256	NT AUTHORITY\SYSTEM
SearchIndexer.exe	5144	NT AUTHORITY\SYSTEM
Registry	68	NT AUTHORITY\SYSTEM
MsMpEng.exe	2316	NT AUTHORITY\SYSTEM
Memory Compression	1680	NT AUTHORITY\SYSTEM
ManagementAgentHost.exe	2300	NT AUTHORITY\SYSTEM
lsass.exe	808	NT AUTHORITY\SYSTEM
dllhost.exe	2936	NT AUTHORITY\SYSTEM

ENSILO

TURNING TABLES BUILDING BLOCKS

Finding targets

- The targeted modules can't be used by VTL1 components
  - UI DLLs are prime candidates
  - Parsers and network libraries also provide good options
- Preferably the module should be a one which is already loaded in the origin process
- The following DLLs fit the description:
  - ole32.dll
  - oleaut32.dll
  - imm32.dll
  - user32.dll

ENSILO

TURNING TABLES BUILDING BLOCKS

Finding targets

- A place that is shared but unused
  - So it won't lead to a crash
- Code caves in PEs are very common
  - At the end of .text section, so it's shared (and executable)
  - Thus, placing the payload is quite straightforward
- On RS4 build 17134:
  - ole32.dll: 0x939 bytes
  - oleaut32.dll: 0x3ef bytes
  - user32.dll: 0xcf7 bytes
  - Imm32.dll: 0x119 bytes

TURNING TABLES BUILDING BLOCKS

Triggering the payload

- The selected module needs to be used quite often in the target process
  - But not too often so overhead won't becomes an issue
  - May also be code that can be triggered from the origin process, for instance via RPC
- DLL entrypoints are very appealing
  - Invoked on every thread start and exit
  - Services on Windows 10 constantly create new threads
  - MSVC CRT main can be easily altered to reach the code cave

```
.text:0000000180023BA0 ; BOOL __stdcall DllMainCRTStartup(HINSTANCE hinstDLL
.text:0000000180023BA0                 public _DllMainCRTStartup
.text:0000000180023BA0 _DllMainCRTStartup proc near                ; DATA XREF:
.text:0000000180023BA0                                             ; .pdata:0000
.text:0000000180023BA0
.text:0000000180023BA0 arg_0           = qword ptr  8
.text:0000000180023BA0 arg_8           = qword ptr  10h
.text:0000000180023BA0
.text:0000000180023BA0                 mov     [rsp+arg_0], rbx
.text:0000000180023BA5                 mov     [rsp+arg_8], rsi
.text:0000000180023BAA                 push    rdi
.text:0000000180023BAB                 sub     rsp, 20h
.text:0000000180023BAF                 mov     rdi, r8
.text:0000000180023BB2                 mov     ebx, edx
.text:0000000180023BB4                 mov     rsi, rcx
.text:0000000180023BB7                 cmp     edx, 1
.text:0000000180023BBA                 jnz     short loc_180023BC1
.text:0000000180023BBC                 call    __security_init_cookie
.text:0000000180023BC1
.text:0000000180023BC1 loc_180023BC1:                              ; CODE XREF:
.text:0000000180023BC1                 mov     r8, rdi         ; reserved
.text:0000000180023BC4                 mov     edx, ebx        ; reason
.text:0000000180023BC6                 mov     rcx, rsi        ; instance
.text:0000000180023BC9                 mov     rbx, [rsp+28h+arg_0]
.text:0000000180023BCE                 mov     rsi, [rsp+28h+arg_8]
.text:0000000180023BD3                 add     rsp, 20h
.text:0000000180023BD7                 pop     rdi
.text:0000000180023BD8                 jmp     dllmain_dispatch
.text:0000000180023BD8 _DllMainCRTStartup endp
```

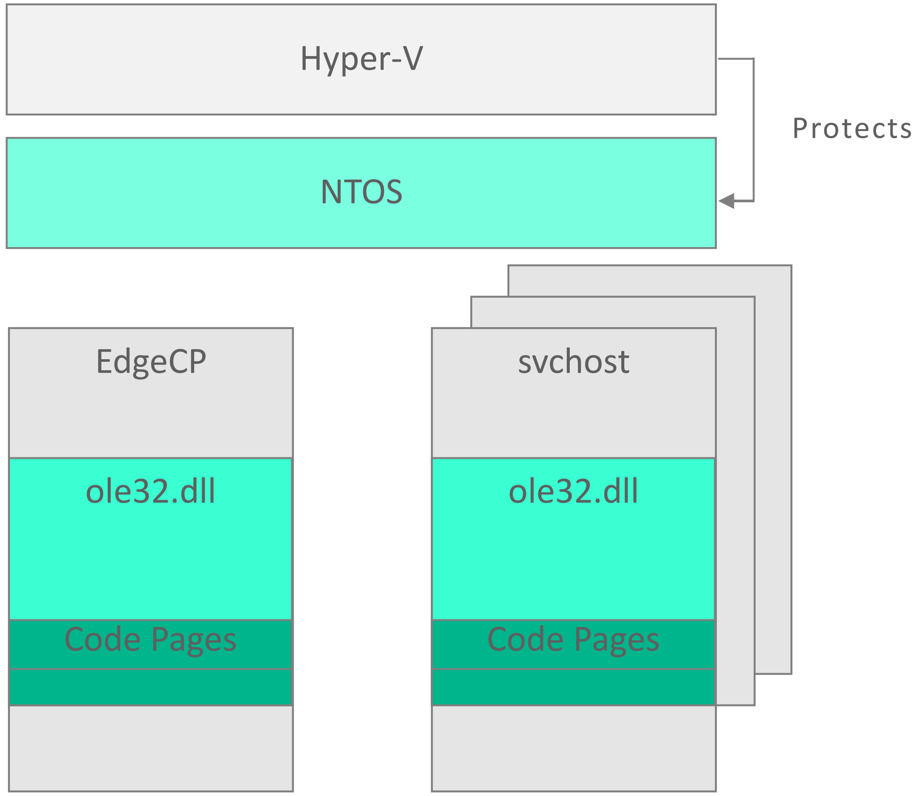

ENSILO

TURNING TABLES BUILDING BLOCKS

Crafting the payload

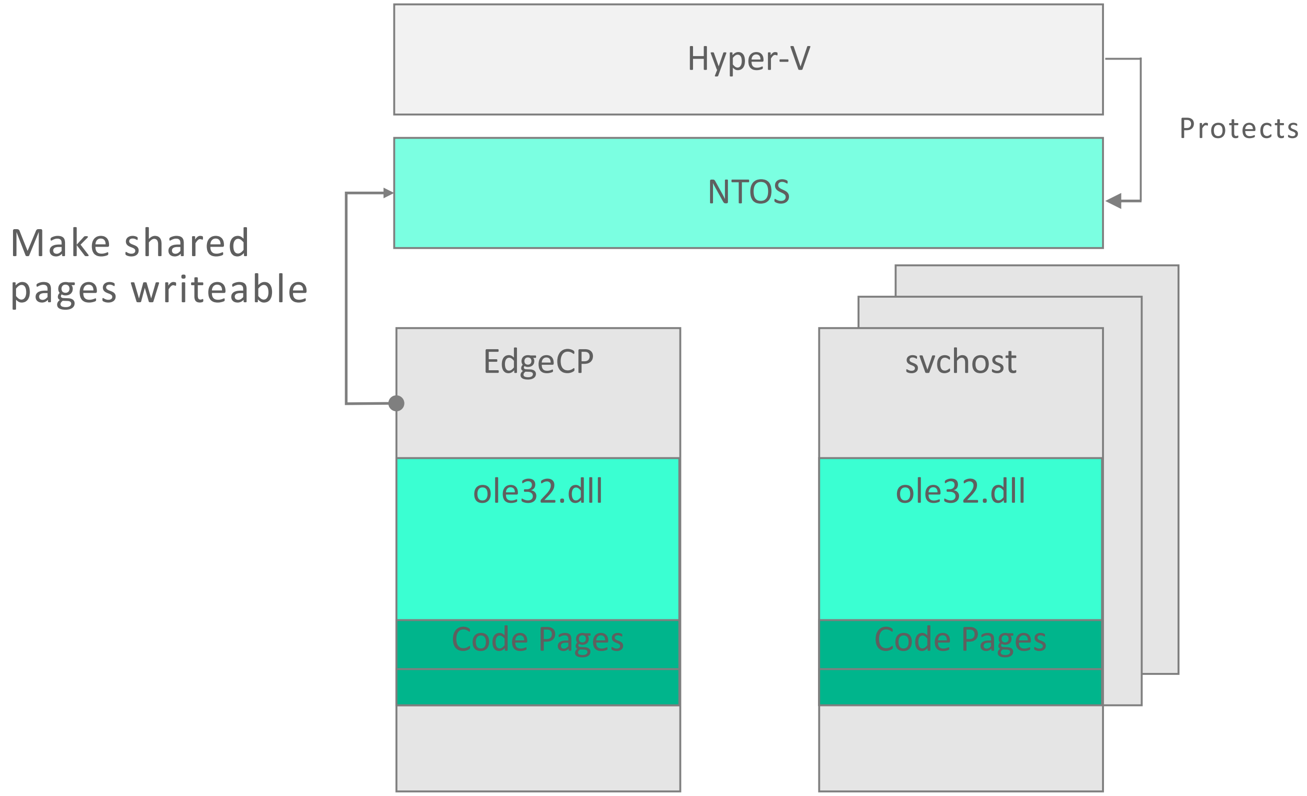
- Make sure we are in the target process
  - We don't know the specific target process ID
  - Check the process name and username
- Synchronize the execution between multiple processes so it will execute only once
  - Obtain a named mutex on start
- Continue to the main payload
  - Map a data section from the origin process
  - Read it directly from the origin process memory
  - Download it from a remote machine

TURNING TABLES TECHNIQUE WALKTHROUGH



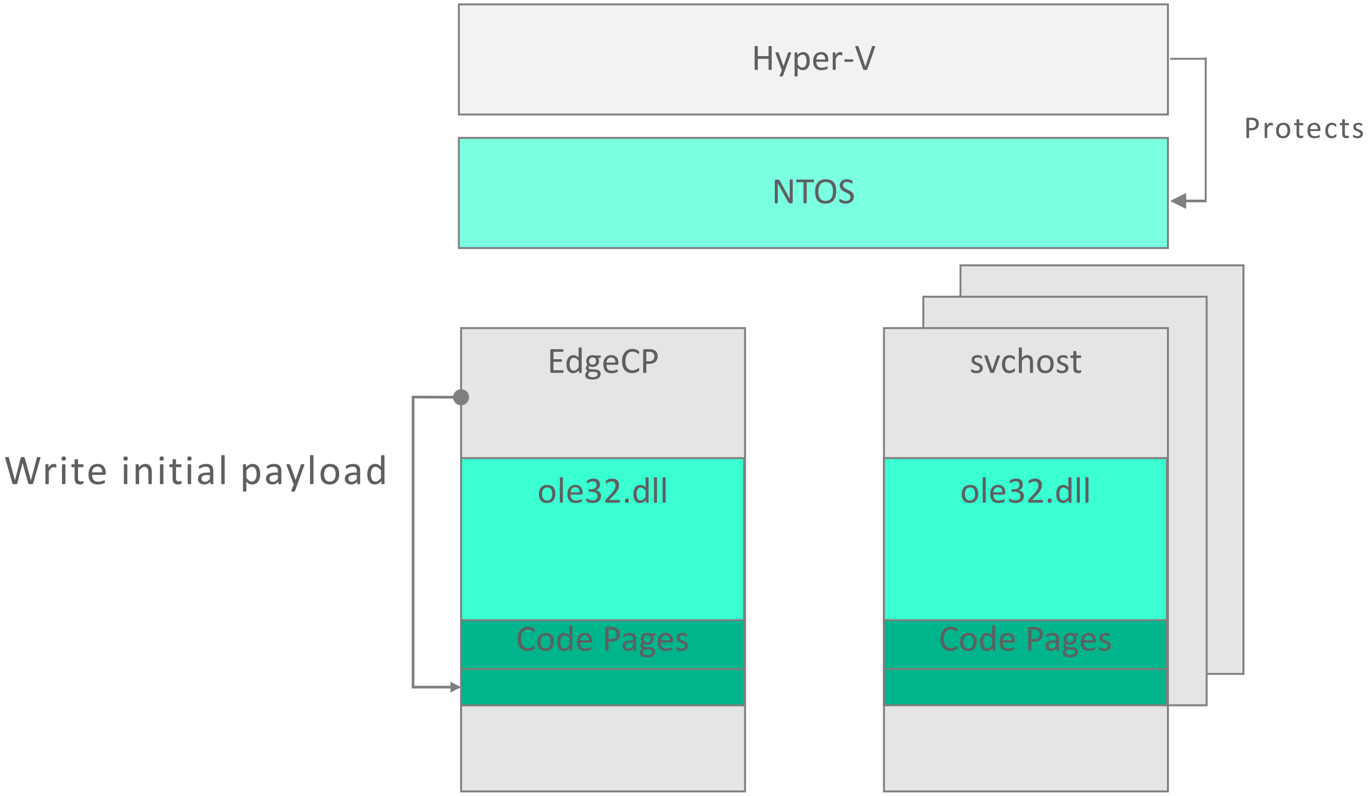
ENSILO

TURNING TABLES TECHNIQUE WALKTHROUGH



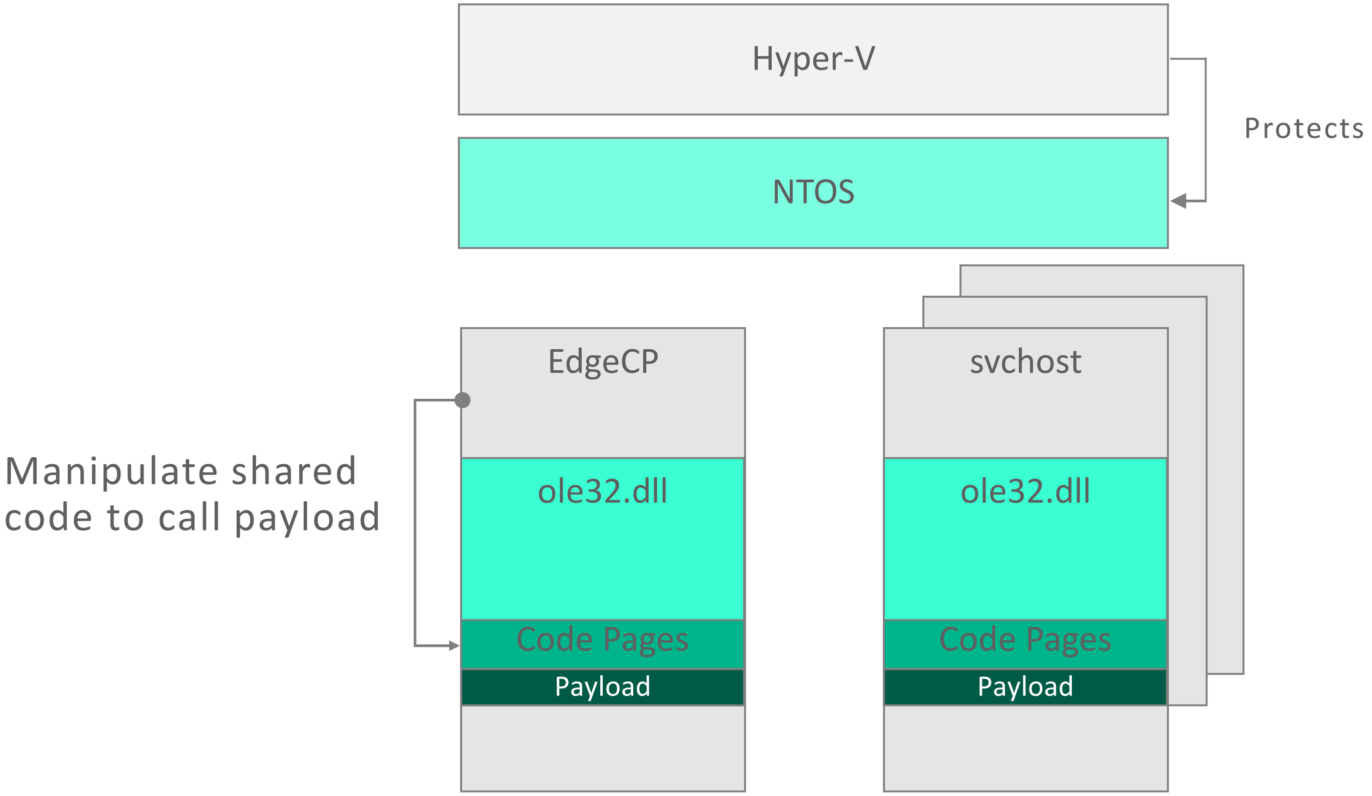
ENSILO

TURNING TABLES TECHNIQUE WALKTHROUGH



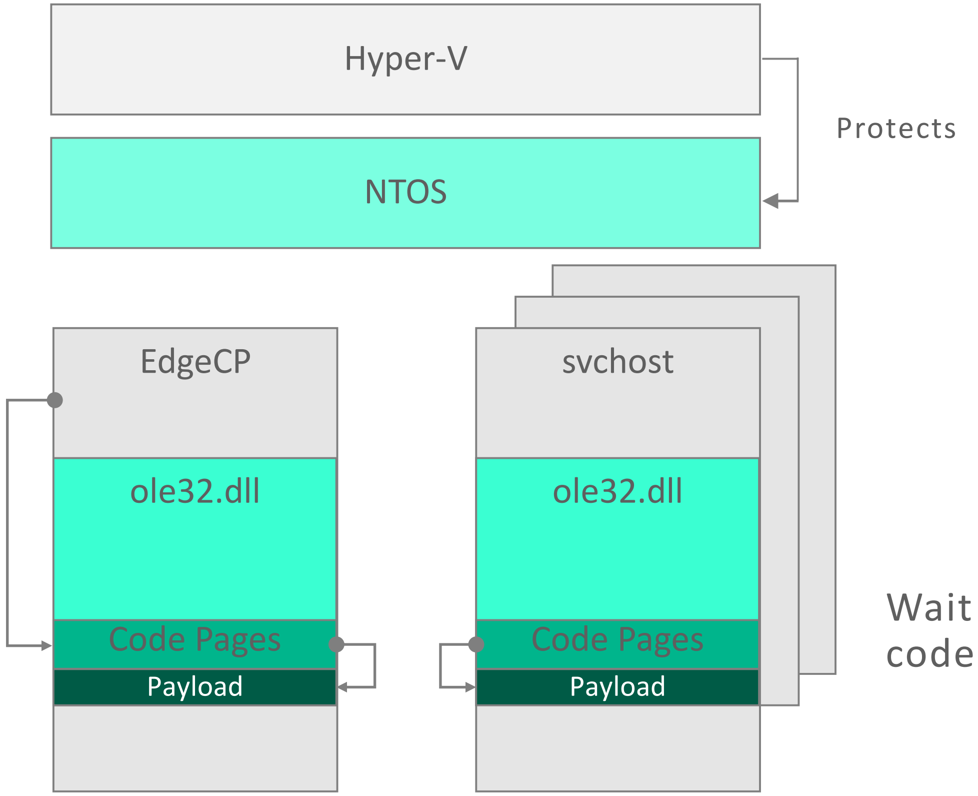
ENSILO

TURNING TABLES TECHNIQUE WALKTHROUGH



ENSILO

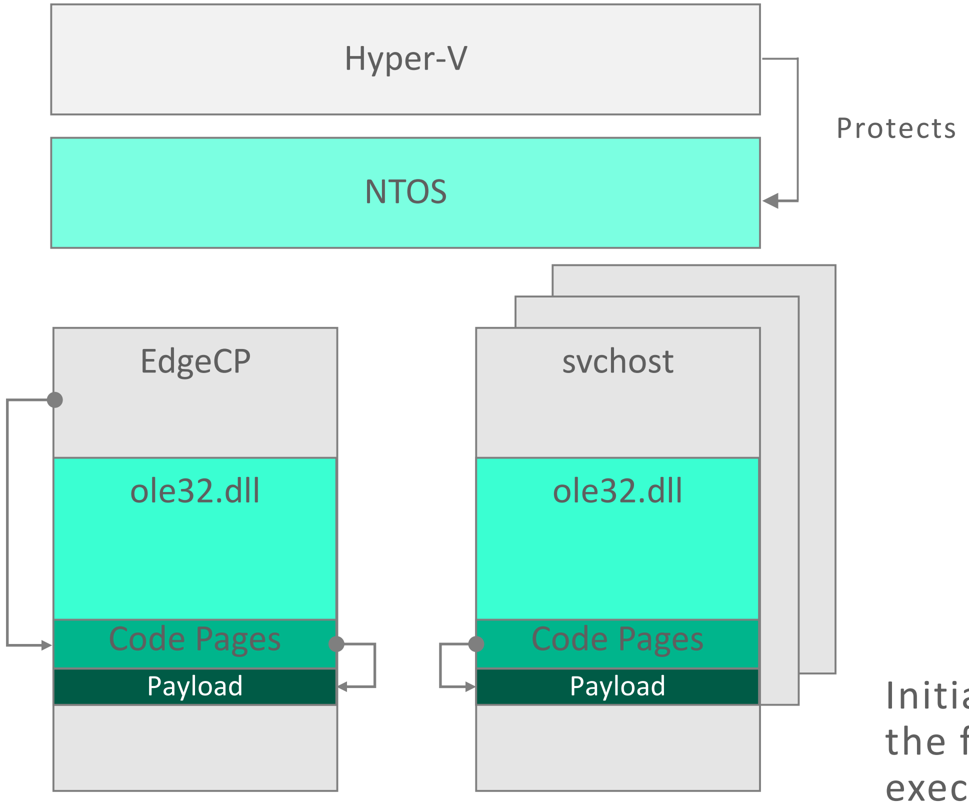
TURNING TABLES TECHNIQUE WALKTHROUGH



Wait for shared code to execute

ENSILO

TURNING TABLES TECHNIQUE WALKTHROUGH



Initial payload loads the full payload and executes it

ENSILO

DEMO

TURNING TABLES VS KERNEL MITIGATIONS

- Page table randomization
  - Easy to bypass using read primitive
- Kernel CFG is bypassed by design
  - No code runs in kernel-mode
- Bypassing KMCI
  - Again, no code runs in kernel-mode
  - No need to bypass the allowed drivers policy

TURNING TABLES VS OTHER TECHNIQUES

- Doesn’t change the process token
  - Which can be monitored and detected
  - Windows Defender System Guard
- Based on simple operations
  - Does not run shellcode in kernel-mode
  - Read operations are of simple, well-defined data structures
- Following a successful privilege escalation we already run in a different process
  - Usually exploited processes, like browsers, has a relatively short life span
- Can also target protected processes

ENSILO

MITIGATIONS

- UMCI (User-Mode Code Integrity)
  - Though not really feasible for general purpose scenarios
- Block +WX with SLAT on every prototype page
  - Already done for shared code with VTL1

ENSILO

CLOSING REMARKS

- Even with latest Windows 10 mitigations generic exploitation methods still work
  - Relevant for current insider build too (RS5)
  - With RS5 VBS and KMCI is planned to be enabled by default
  - Suggested mitigations sent to Microsoft
- Relevant without KMCI as well
- Control flow integrity mitigations are not an issue
  - No need to manipulate function pointers
  - Will work even with protections like CET (hardware enforced CFI)
- Not limited to Windows
  - Copy-On-Write/Shared Memory is used on every modern OS

ENSILO

REFERENCES

- Intel Software Developer's Manual
- AMD-V Nested Paging
- Windows Internals 6th edition
- Battle Of SKM And IUM - How Windows 10 Rewrite OS Architecture
- Taking Windows 10 Kernel Exploitation To the Next Level – Leveraging Write-What-Where Vulnerabilities In Creators Update

ENSILO

QUESTIONS?

THANK YOU

www.breakingmalware.com

omri@ensilo.com in/omri-misgav

udi@ensilo.com in/udiyavo @UdiYavo