Windows SMEP Bypass

U=S

Nicolas A. Economou

Enrique E. Nissim

CORE SECURITY

Schedule

- Reviewing Modern Kernel Protections
- Introducing SMEP
- Windows SMEP bypass techniques – Part 1
- Windows Paging Mechanism
- Windows SMEP bypass techniques – Part 2
- DEMO
- Conclusions

Reviewing Modern Protections

- **DEP/NX:** is a security feature included in modern operating systems. It marks areas of memory as either "executable" or "nonexecutable".

- **NonPagedPoolNX:** new type of pool introduced in Windows 8

- **KASLR:** Address-space layout randomization (ASLR) is a well-known technique to make exploits harder by placing various objects at random, rather than fixed, memory addresses.

- **NULL Dereference Protection:** cannot alloc the null page.

Reviewing Modern Protections

- **Integrity Levels:** call restrictions for applications running in low integrity level – since Windows 8.1.

- **KMCS:** Kernel-mode software must be digitally signed to be loaded on x64-based versions of Windows Vista and later versions of the Windows family of operating systems.

- **KPP:** Kernel Patch Protection (informally known as PatchGuard): is a feature of x64 editions of Windows that prevents patching common structures of the kernel.(Hooking IDT, SSDT, GDT, LDT is out of the table).

Reviewing Modern Protections

- **SMAP:** allows pages to be protected from supervisor-mode data accesses. If SMAP = 1, software operating in supervisor mode cannot access data at linear addresses that are accessible in user mode.

- **SMEP:** Supervisor Mode Execution Prevention allows pages to be protected from supervisor-mode instruction fetches. If SMEP = 1, software operating in supervisor mode cannot fetch instructions from linear addresses that are accessible in user mode.

SMEP

What is SMEP?

- Aka: **"Supervisor Mode Execution Prevention"**

- Detects **RING-0** code running in **USER SPACE**

- Introduced at Intel processors based on the **Ivy Bridge** architecture

- **Security feature** launched in **2011**

What is SMEP on Windows?

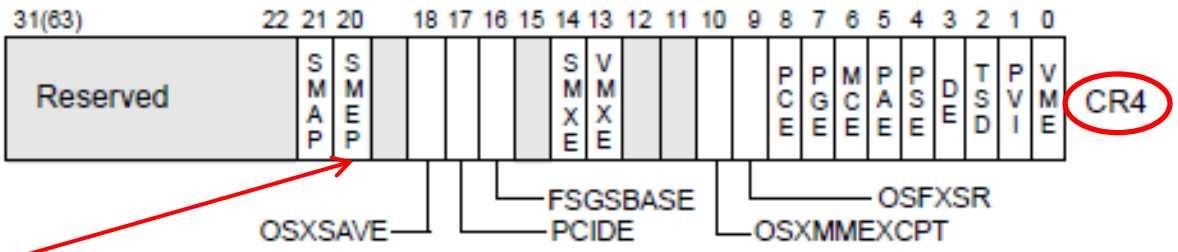
- **Enabled by default since Windows 8.0** (32/64 bits)

- **Kernel exploit mitigation**

- Specially "**Local privilege escalation**" exploits must now **consider** this feature.

How does it work?

- Feature enabled by the OS



- Detects **ring-0** code running in **user space**

- User space = Memory space used by applications programs (stack, heap, code, etc).

- **Ring-0** code is used by **kernel OSs**
- **Ring-3** code is used by **applications**

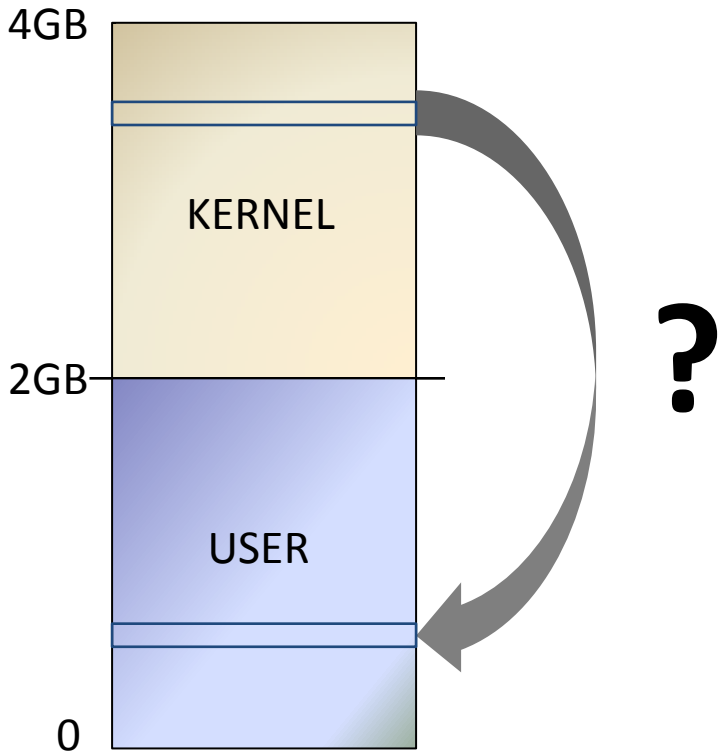
SMEP CPU support

- Desktop processors
  - **Intel Core**: Lastest models of i3, i5, i7
  - **Intel Pentium**: G20X0(T) and G21X0(T)
  - **Intel Celeron**: G1610(T), G1620(T) and G1630

- Server processors
  - **Intel Xeon**: Lastest models of E3, E5, E7
  - **Intel Pentium**: 1403v3 and 1405v2

SMEP Protection

- We control EIP/RIP = **0x41414141** in Ring-0
  - So, we can **jump** where we want to …



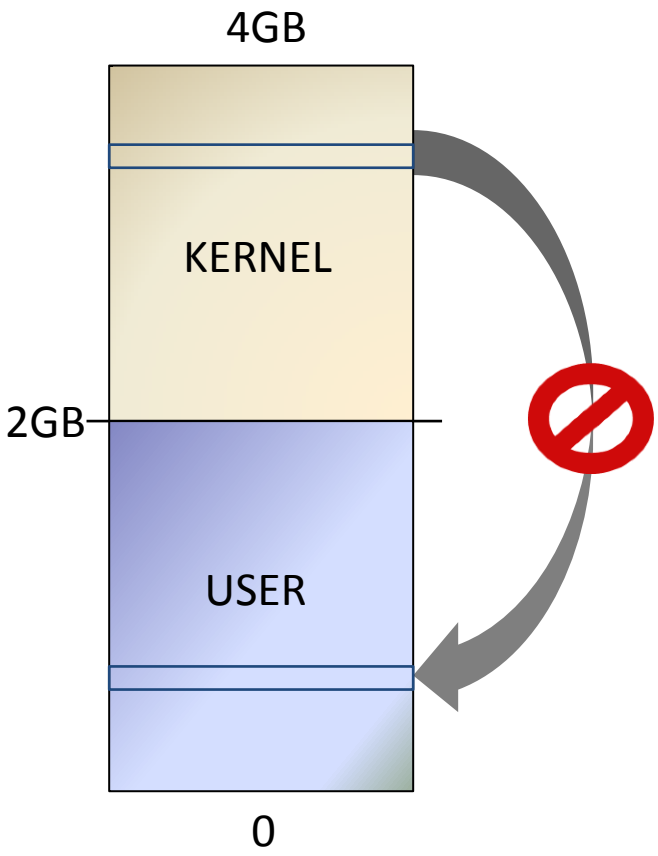
Windows SMEP bypass techniques – Part 1

CORE SECURITY

Option 0: Jumping to user space

- Jump to **user space** (to my “code”)

- If (**EIP/RIP** == **USER** memory pages)
  - **Page Fault → BSoD**



  - **Error: ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY**

Option 1 - x86: Jumping to kernel heap

- Jump to kernel data (Heap):
  - E.g: “Windows 8.1” **32 bits**
  - **Data** allocated in **POOL TYPE** = 0x21
  - 0x21 = NonPagedPoolSession + NonPagedPool = **executable!**
  - **SMEP bypass** ☺ ☺ ☺

http://blog.ptsecurity.com/2012/09/intel-smep-overview-and-partial-bypass.html

Option 1 - x64: Jumping to kernel heap

- Jump to kernel data (Heap):
  - E.g: “Windows 8.1” **64 bits**
  - **Data** allocated in **POOL TYPE** = 0x21
  - 0x21 = NonPagedPoolSession + NonPagedPool = **NO executable?**

  - **No longer an option ☹**

Option 2: ROPing in kernel space

- Jump to kernel code (win32k.sys):
  - Get modules addresses with **NtQuerySystemInformation()** (only in Medium Integrity since Windows 8.1)
  - If running in **Low Integrity** we need **memory leaks**
  - You need to write a **ROP chain** to **bypass SMEP**

Option 2: ROPing to Turn off SMEP

- Turn off the **bit 20th** of the **CR4** register
  - E.g "*mov rax,0xFFF**E**FFFFF*"/ "*mov cr4,rax*"/"*ret*"
- Jump to USER SPACE ☺
- **Problem:** Restore the CR4 register (**PatchGuard!**)

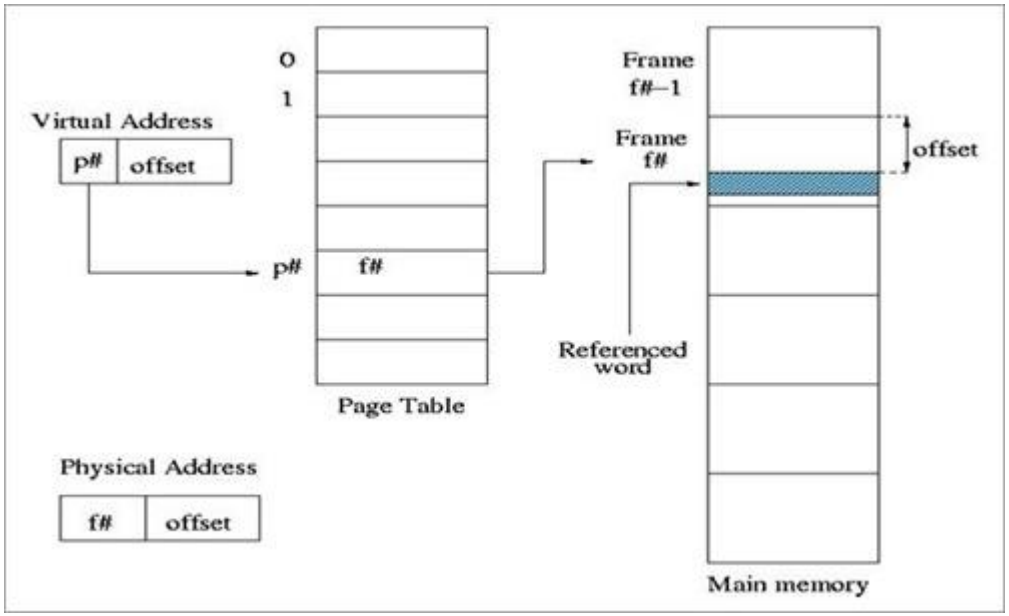
- The most **well-known** technique !

http://blog.ptsecurity.com/2012/09/bypassing-intel-smep-on-windows-8-x64.html

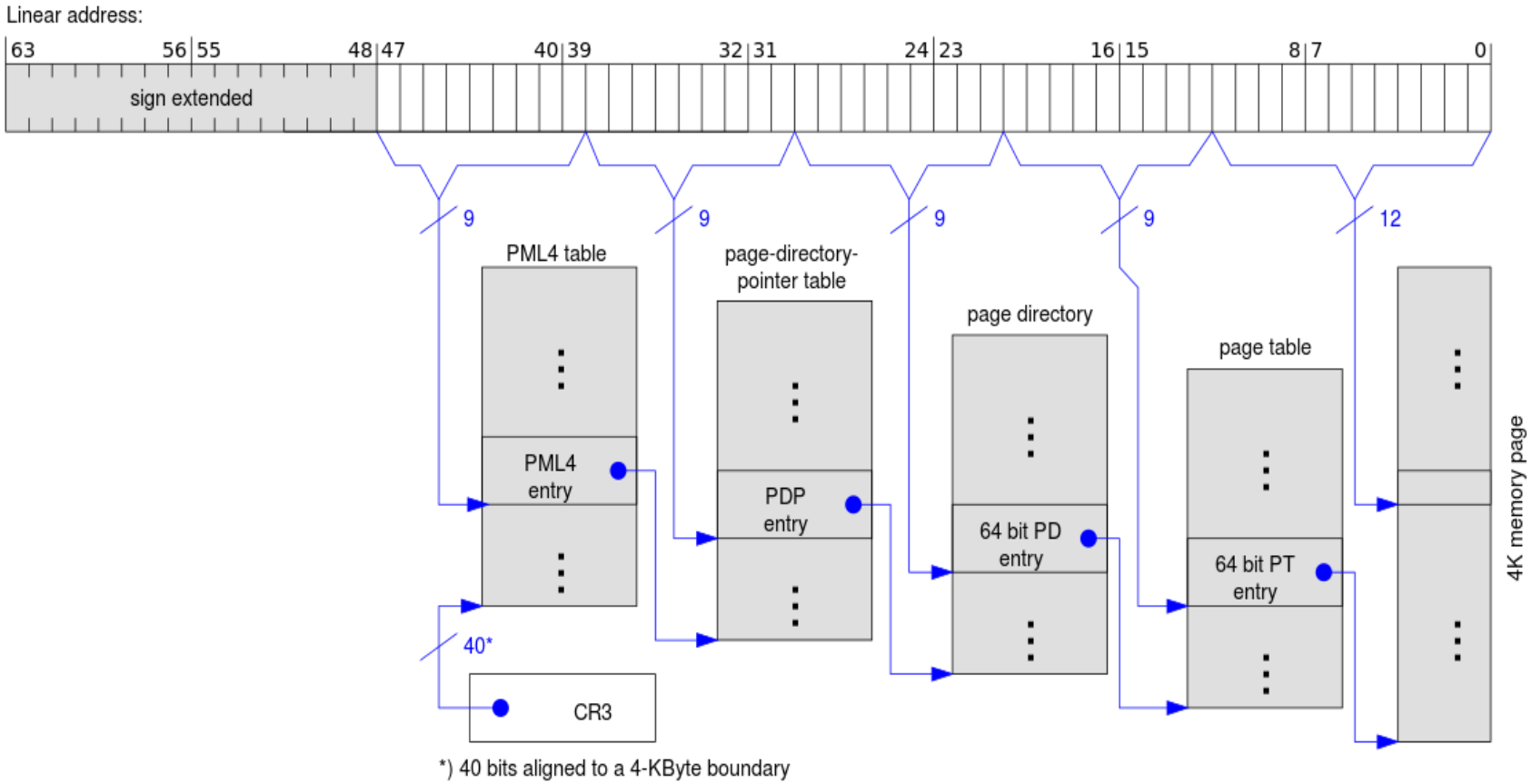
Windows Paging Mechanism

Paging 101

- Paging is a functionality provided by the MMU and used by the processor to implement virtual memory.
- A virtual address is the one used in processor instructions; this must be translated into a physical address to actually refer a memory location.



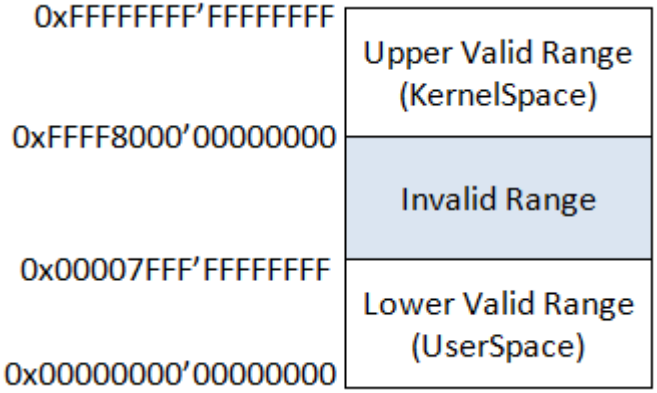
Windows Paging x64



CORE SECURITY

Canonical Addresses

- With 64bits we can address 2^{64} bytes of memory (16 Exabytes). Current x64 processors however, limit the number of bits to 48, but instead of simply disallowing bits 48-63, they set them to be equal to bit 47.



- Attempting to use a non-canonical address causes a Page Fault exception.

PxE Structure

63	62:52	51:12	11	10	9	8	7	6	5	4	3	2	1	0
XD	I	PFN	I	I	I	G	PAT	D	A	PCD	PWT	U/S	R/W	P

Interesting fields to know for our purposes:

- **R/W:** readonly/readwrite
- **U/S:** if set, the range mapped by the entry is accessible at CPL3. Otherwise it is only accessible at CPL0.
- **XD:** if set, instruction fetching is not allowed for the region mapped by the entry.

Self-ref Entry

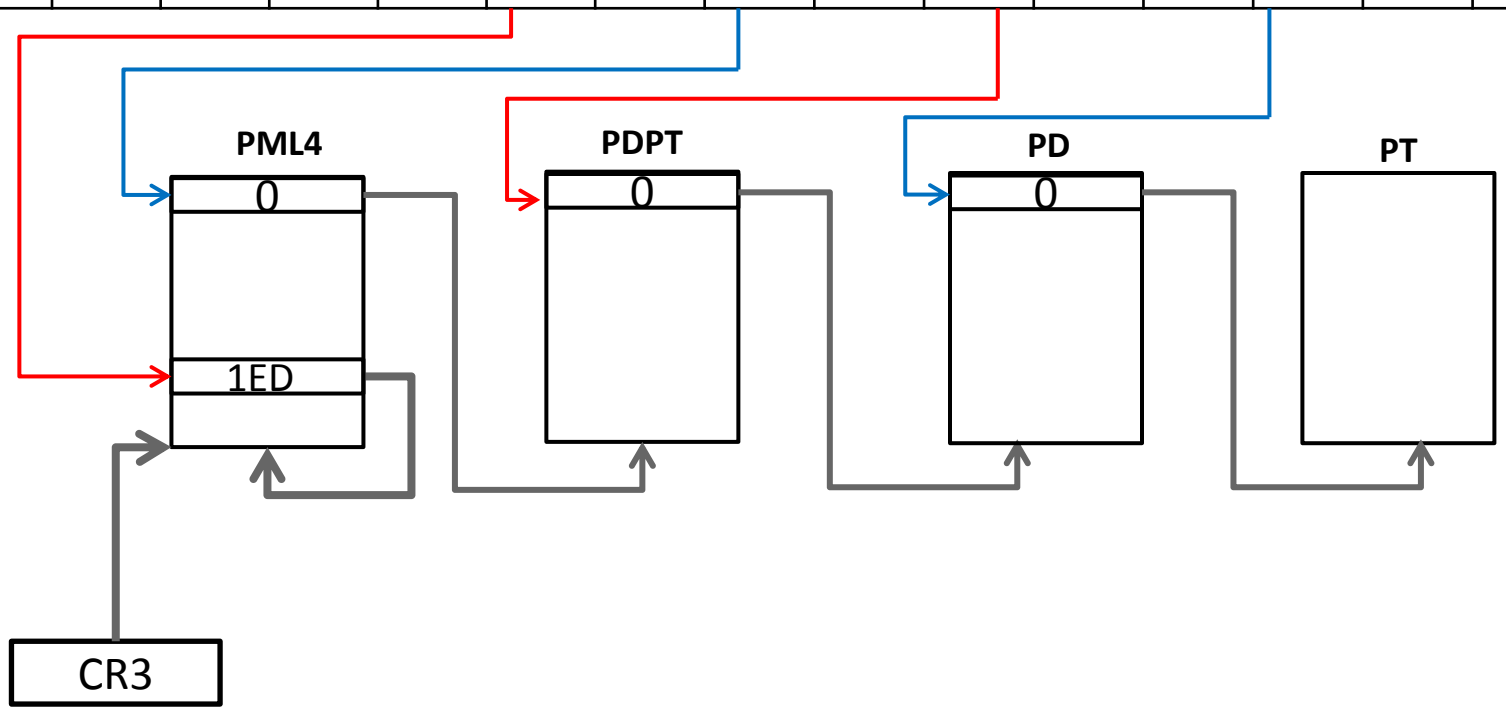
- **Entry 0x1ED = 1 1110 1101**
- Since bit 47 is 1, all the bits 48-64 must be 1 to be a valid canonical address

1111	1111	1111	1111	1111	0101	1XXX	XXXX
F	F	F	F	F	6	8-F	0-F

- **Range: 0xFFFFF680'00000000 – 0xFFFFF6FF'FFFFFFFF**

Self-ref Entry

F	F	F	F	F	6	8	0	0	0	0	0	0	0	0	0
1111	1111	1111	1111	1111	0101	1000	0000	0000	0000	0000	0000	0000	0000	0000	0000



Quick Formula

```
kd> !pte 0x0000000000000000
                                           VA 0000000000000000
PXE at FFFFF6FB7DBED000    PPE at FFFFF6FB7DA00000    PDE at FFFFF6FB40000000    PTE at FFFFF68000000000
contains 0110000003A5A867  contains 0000000000000000
pfn 3a5a      ---DA--UWEV  not valid

kd>
```

```
_int64 get_pxe_address(_int64 address)
{
 _int64 result = address>>9;
 result = result | 0xFFFFF68000000000;
 result = result & 0xFFFFF6FFFFFFFFF8;
 return result;
}
```

Quick Formula

```
kd> !pte ffd00000
                    VA ffd00000
PDE at C0603FF0            PTE at C07FE800
contains 000000000034C163  contains 000000005DAF0123
pfn 34c        -G-DA--KWEV  pfn 5daf0      -G--A--KWEV

kd>
```

```
int get_pxe_32(int address) {
    int result = address>>9;
    result = result | 0xC0000000;
    result = result & 0xC07FFFF8;
    return result;
}
```

Windows SMEP bypass techniques – Part 2

CORE SECURITY

Option 3: Unprotecting HAL.DLL heap

- Using **multiple** arbitrary writes (**ROPing or not**)

- Write shellcode in this area:
  - Address - 32 bits = **0xffd00000** (**no ASLR**)
  - Address - 64 bits = **0xffffffff'ffd00000** (**no ASLR**)

- Turn off the **NX bit** HAL's heap
- Overwrite a HAL's heap function pointer
- **Jump** to HAL's heap ☺

https://drive.google.com/file/d/0B3P18M-shbwrNWZTa181ZWRCclk/edit?pli=1

U = S

Option 4: Deceiving SMEP

- If **SMEP** detects ring-0 code running in USER SPACE (USER PAGES)

- If **PTE tables** are in **fixed addresses**

- What about changing our **USER PAGE** to **SUPERVISOR PAGE**? … ☺

Flipping U/S

- Option 4: “First time somebody mentioned this”
  - Conference: **NSA - Trusted Computing (2011)**
  - Speaker: **Stephen Fischer**
  - https://www.ncsi.com/nsatc11/presentations/wednesday/emerging_technologies/fischer.pdf
  - Slide: 9

Flipping U/S

- Option 4: "… and then …"
  - Blog: **Windows 8 Kernel Memory Protections Bypass**
  - Author: **MWR LABS - Jérémy Fetiveau**
  - https://labs.mwrinfosecurity.com/blog/2014/08/15/windows-8-kernel-memory-protections-bypass
  - Section: "Modifying Paging Structures"

Flipping U/S

- Option 4: "… and finally"
  - Conference: **Infiltrate 2015**
  - Speaker: **Alex Ionescu**
  - http://www.alex-ionescu.com/infiltrate2015.pdf
  - Slides: 69 and 71 …

A. Ionescu at Infiltrate 2015

What MWR & Others Got Wrong

"When checking the rights of a page, the kernel will check every PXE involved in the address translation. That means that if we want to check if the U/S flag is set, we will check all entries relating to that page. ***If any of the entries do not have the supervisor flag set****, any attempt to use this page from kernel mode will trigger a fault.* ***If all of the entries have the supervisor flag set****, the page will be considered a kernel-mode page."*

We also found the same behavior on our own

so, it's FALSE !

A. Ionescu at Infiltrate 2015

Intel Doublespeak

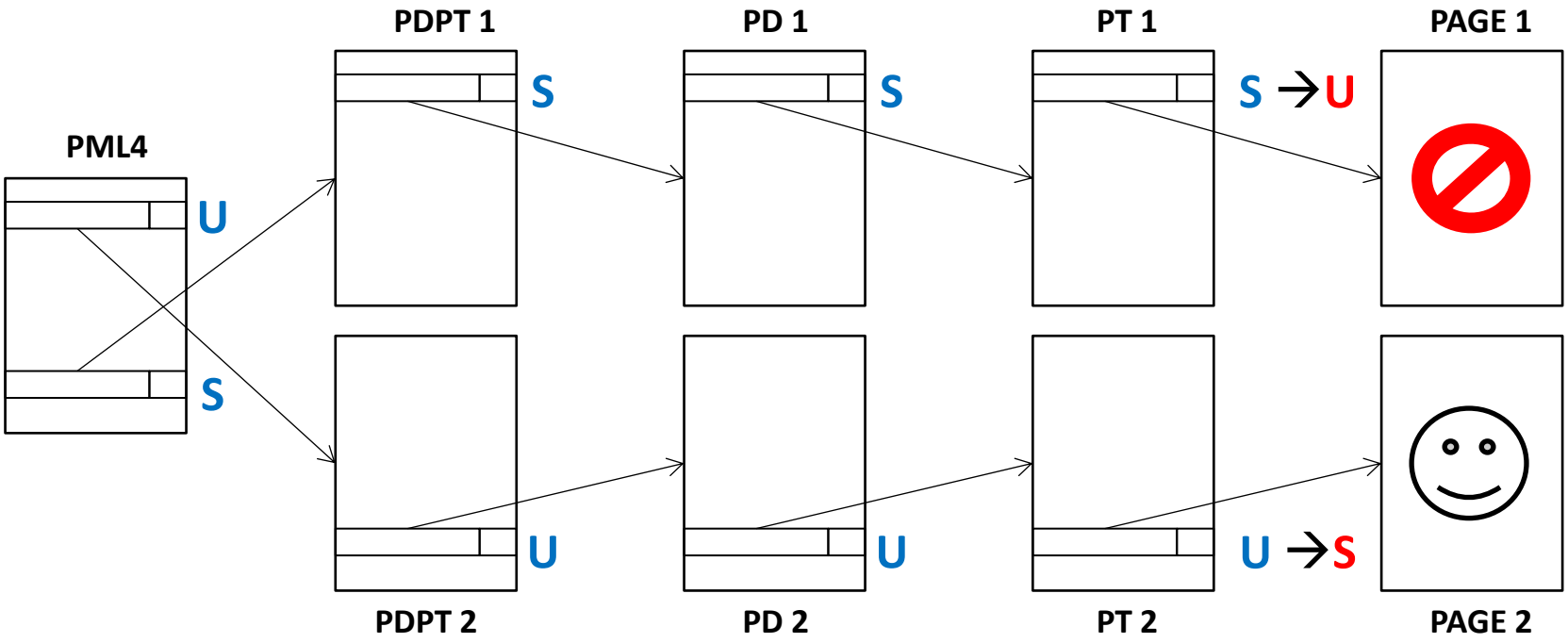
SMEP architectural control details

- CR4.SMEP – If 1 and in supervisor mode (CPL<3), instructions may not be executed from a linear address for which the U/S flag is 1 (user mode) in every paging-structure entry controlling the translation for the linear address
  - SMEP U/S paging attribute precedence:
    - Any page level marked as supervisor (U/S=0) will result in treatment as supervisor for SMEP enforcement
      - Existing user/supervisor privileging checking continues to require the more conservative mapping (i.e. execution in user mode (CPL=3) requires all levels to be mapped as U/S=1 (user)

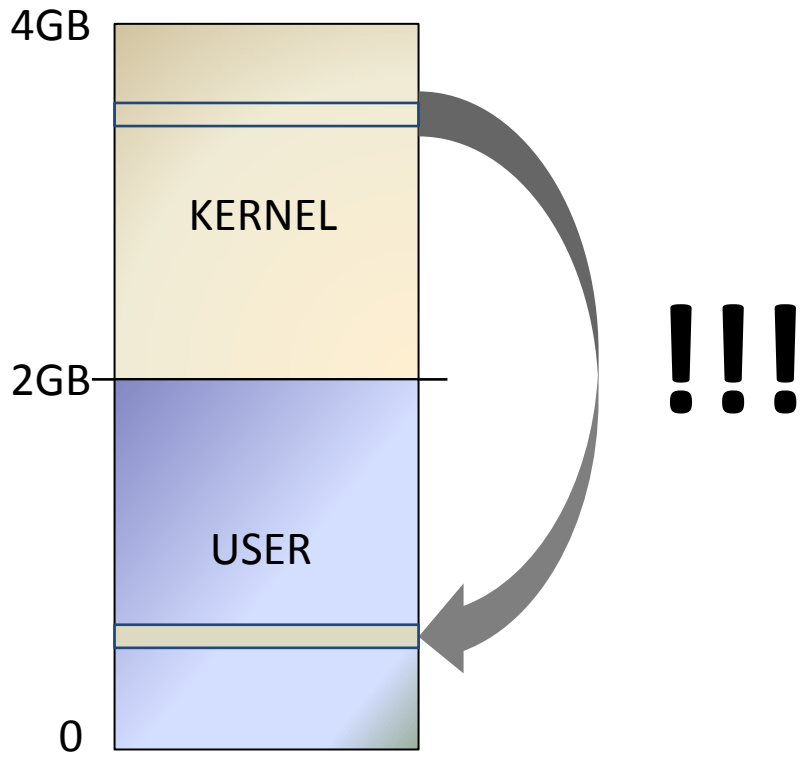
CORE SECURITY

Flipping U/S

- **Breaking Rules**



Mapping a Kernel Page in User Space



The processor will not generate an exception!

Demo time

CVE-2015-5736

- Exploit:
  - **“Fortinet Antivirus Multiple Vulnerabilities”** (CVE-2015-5736)
  - **Arbitrary** function callback feature?
  - Local Privilege escalation
  - http://www.coresecurity.com/advisories/forticlient-antivirus-multiple-vulnerabilities

Exploit for CVE-2015-5736

- Target:
  - **"Windows 10"** 64 bits + **"Forticlient <= 5.2.3"** installed
- Scenario:
  - We **can't** jump directly to USER SPACE (**SMEP!**)
  - **No registers** poiting to **our DATA**!
  - The only way, **Stack Pivoting** to **USER SPACE**
- Objective:
  - **Write** a **ROP chain** to **avoid SMEP!**
  - **Run** our **RING-0** code in **USER SPACE**

Exploit for CVE-2015-5736

- **Vulnerable Driver:** FortiShield.sys
  - A filesystem filter driver that hooks several operations -> IRP_MJ_SET_INFORMATION

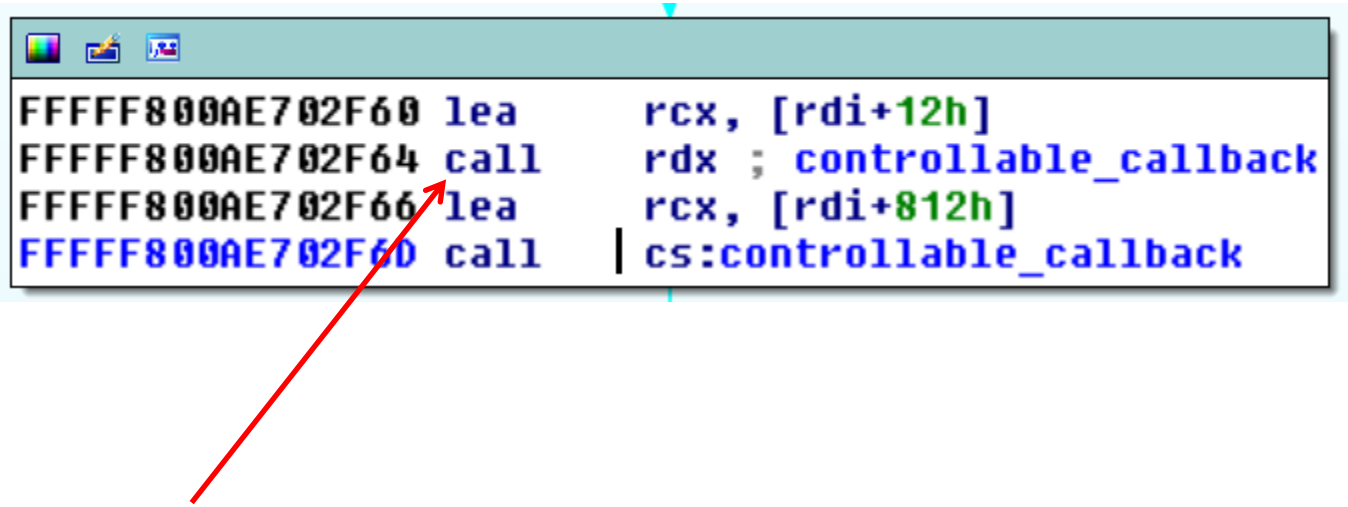
```
FFFFF800AE7045DD lea     rcx, SpinLock    ; SpinLock
FFFFF800AE7045E4 call    cs:KeAcquireSpinLockRaiseToDpc
FFFFF800AE7045EA mov     rcx, [rbp+0]
FFFFF800AE7045EE mov     cs:controllable_callback, rcx
```

RCX Controllable

  - IOCTL: 220028h

Exploit for CVE-2015-5736

- Arbitrary Callback: Invoked via **MoveFileEx()**



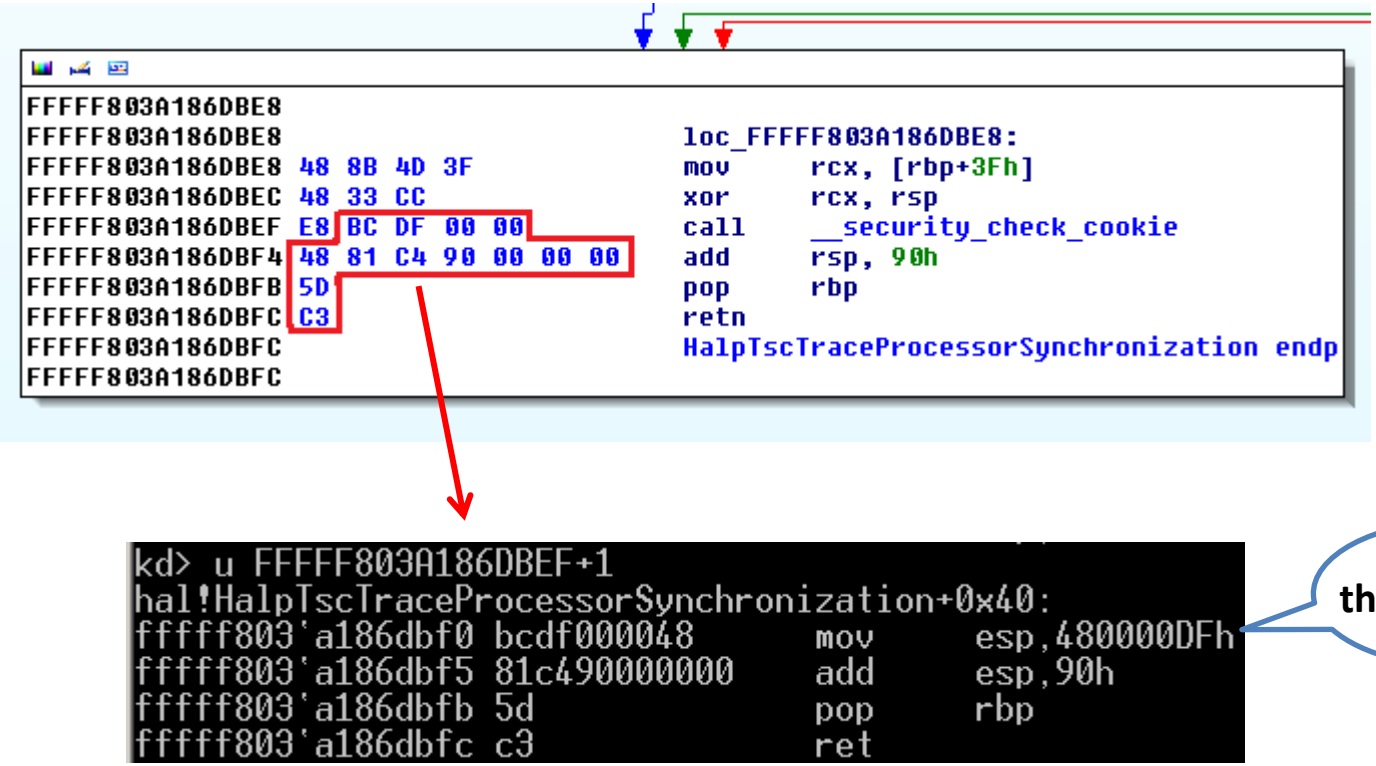
We control this call

Exploit for CVE-2015-5736

- Gadget finding:
  - Tool: **Agafi** - https://github.com/CoreSecurity/Agafi
  - Trick: Many **64 bit** instr. are **equal** to **32 bit** instr.
  - Manual search: We found the rest!
  - Result: **ALL** gadgets located in **HAL.DLL** … ☺

ROP in KernelSpace

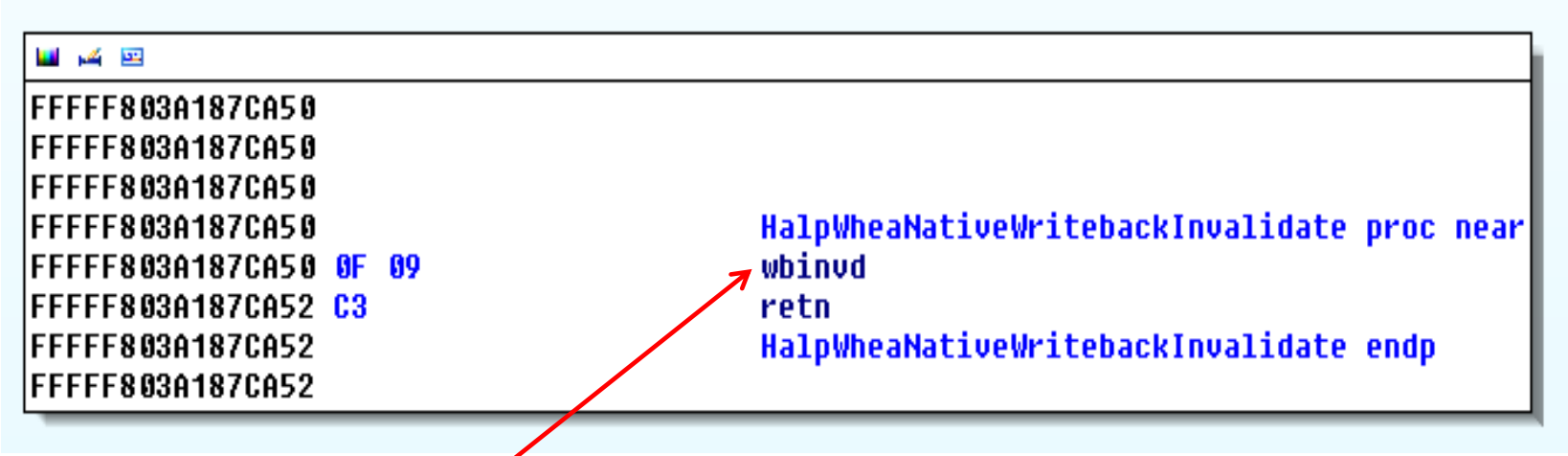
- Special gadget: **Stack Pivoting** to **user space**



CORE SECURITY

ROP in KernelSpace

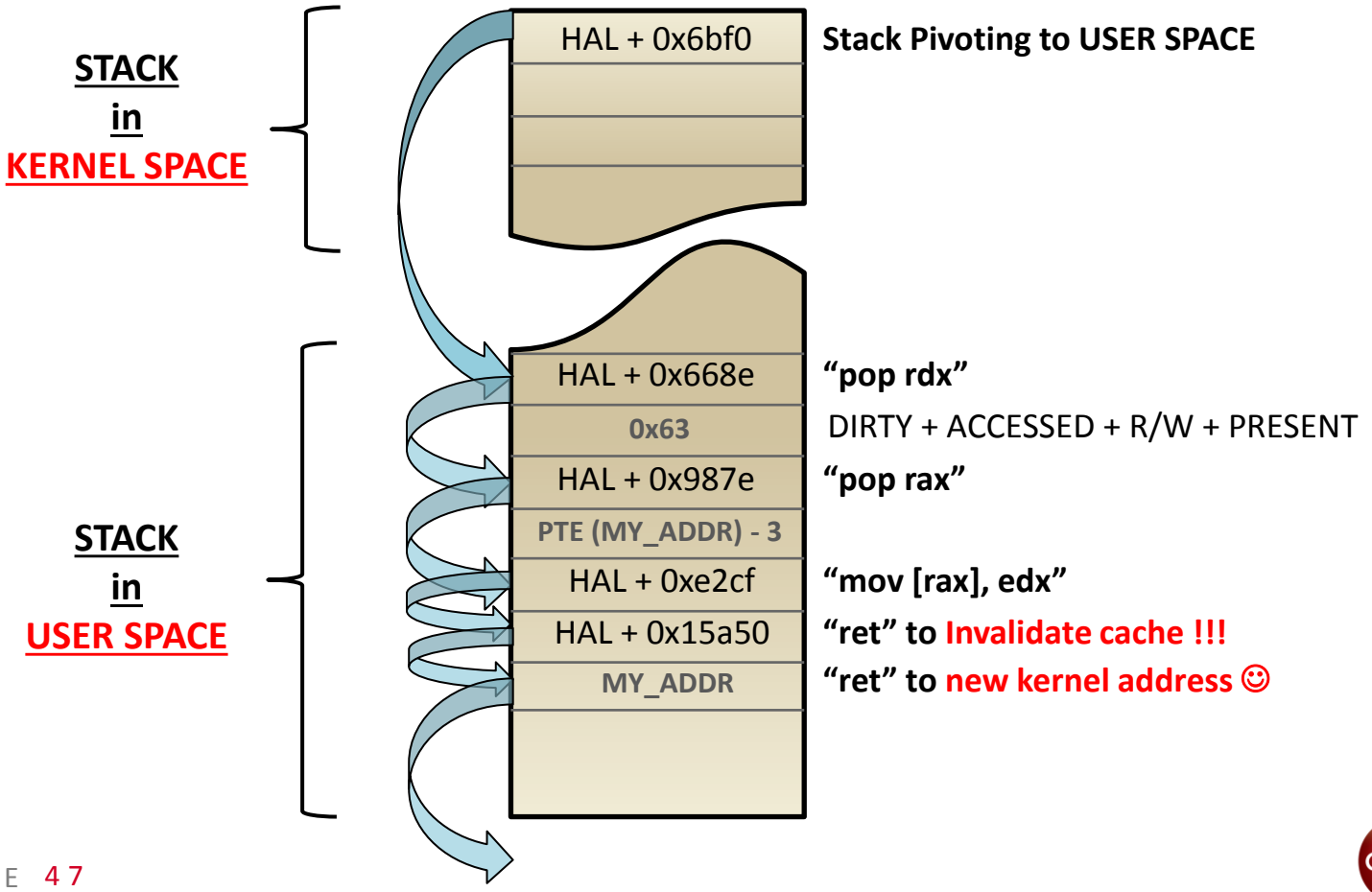
- Special gadget: **Disabling** the **CPU TLB cache**



It refreshes the TLB cache !

ROP in KernelSpace

- ROPing to "**hal.dll**" - "Windows 10" 64 bits



CORE SECURITY

Demo time now

Conclusions

- The **PML entry** (**0x1ed**) should be **RANDOMIZED**
  - **256** entries are available for the OS kernel
  - **Only ~20 entries** are used by **Windows**

- **Paging tables (PTs)** shouldn't be in **PREDICTABLE VAs**
  - It can be **abused** by **LOCAL** and **REMOTE** kernel exploits

- **Virtualization ?**
  - Enabled by **VSM** in **Windows 10**
  - Multiples **EPTs** (Extended Pages Tables - **SLAT**) could be a solution

Conclusions

- This **bypass technique** is **useful** when **EIP/RIP** is controllable (directly or via ARB. WRITE)

- **Windows SMEP kernel exploit mitigation**
  - **Easily bypassable**
  - **Only useful** when we are in **Low Integrity Level**

Questions?

Thank You

Enrique Nissim
@kiqueNissim
enissim@coresecurity.com

Nicolas Economou
@NicoEconomou
neconomou@coresecurity.com

CORE SECURITY