IOMMU et attaques DMA

Jérémie Boutoille[1] et Jean-Christophe Delaunay[2]

[1] Synacktiv - jeremie.boutoille@synacktiv.com
[2] Synacktiv - jean-christophe.delaunay@synacktiv.com

Résumé Ce papier présente un état de l'art des connaissances actuelles en matière d'attaques via *Direct Memory Access* visant à déverrouiller une session utilisateur. Une revue du fonctionnement et l'intégration de l'*Input Output Memory Management Unit* (IOMMU) [3] par les principaux systèmes d'exploitation modernes (Windows, macOS et Linux) est effectuée dans un premier temps. Puis, les attaques existantes via DMA par périphérique externe sur machine allumée sont présentées. Ces attaques permettent d'accéder à la session d'un utilisateur bien que l'ordinateur soit verrouillé. Ces travaux ont été réalisés en prévision d'un nouveau projet RAPID [4] par Synacktiv : `DMArvest`.

Keywords: DMA · IOMMU · ACLs · PCI Express · Attaques logicielles · Attaques matérielles · Windows · Linux · macOS · Pcileech · Thunderclap

3. Seule la technologie Intel VT-d est traitée dans ce document
4. https://www.defense.gouv.fr/aid/soutenir-vos-projets/subventions/rapid

1 Introduction

La technologie *Direct Memory Access* (DMA) permet à des périphériques d'accéder à la mémoire principale (*Random Access Memory* - RAM) sans passer par le processeur principal (CPU). Cette technologie existe depuis de nombreuses années mais pose de nouvelles problématiques avec l'utilisation généralisée des technologies de virtualisation et de *Cloud*. Le DMA permet l'amélioration des performances et est donc indispensable. Cependant, il peut aussi être utilisé pour contourner des mécanismes de sécurité implémentés par les composants logiciels et matériels. Afin de limiter et de contrôler les accès DMA, un nouveau composant appelé *Input Output Memory Management Unit* (IOMMU) a été intégré. Cette IOMMU détermine notamment les accès mémoire pouvant être réalisés par les périphériques situés sur le bus *PCI*.

1.1 Technologies

De multiples bus de communications supportant le DMA ont fait leur apparition au fil des années, afin d'offrir des débits toujours plus importants.

En particulier, PCI, AGP, `FireWire` et `PCI Express`, comptent parmi les technologies plus connues du grand public. Si `FireWire` tend à disparaitre de nos jours, PCIe évolue constamment avec de nouvelles spécifications permettant des débits accrus [5].

PCIe se démocratisant au-delà d'un simple usage graphique, diverses technologies en tirent parti pour garantir des performances optimales. Par exemple, les disques durs M.2 NVMe et certaines cartes réseau ou Bluetooth reposent sur PCIe. Thunderbolt permet lui de connecter des périphériques externes directement sur le bus PCI. La norme USB 4.0, fraichement dévoilée[28], apporte le support par défaut du *PCIe Tunneling*.

(a) Disque dur NVMe avec connectique M.2[22]

(b) Câble USB-C avec support Thunderbolt[23]

FIGURE 1: Technologies reposant sur PCI Express

Les attaques décrites dans ce document reposent presque toutes sur l'utilisation d'un port USB-C permettant l'accès au bus PCI.

5. La spécification 4.0 est sortie en 2017 et la version 5.0 est prévue pour 2019

2 Intel VT-d

2.1 Fonctionnement global

La technologie *Intel Virtualization Technology for Directed I/O* (VT-d) permet de mettre en place une IOMMU. VT-d établit du DMA *remapping* visant à restreindre l'accès DMA à certaines zones mémoires. Ce DMA *remapping* fonctionne à la manière d'une MMU classique, la traduction d'adresse s'effectue via différents niveaux de tables de pages (d'où le terme IO-MMU), avec un *mapping* par pages de 4ko, 2Mo ou 1Go.

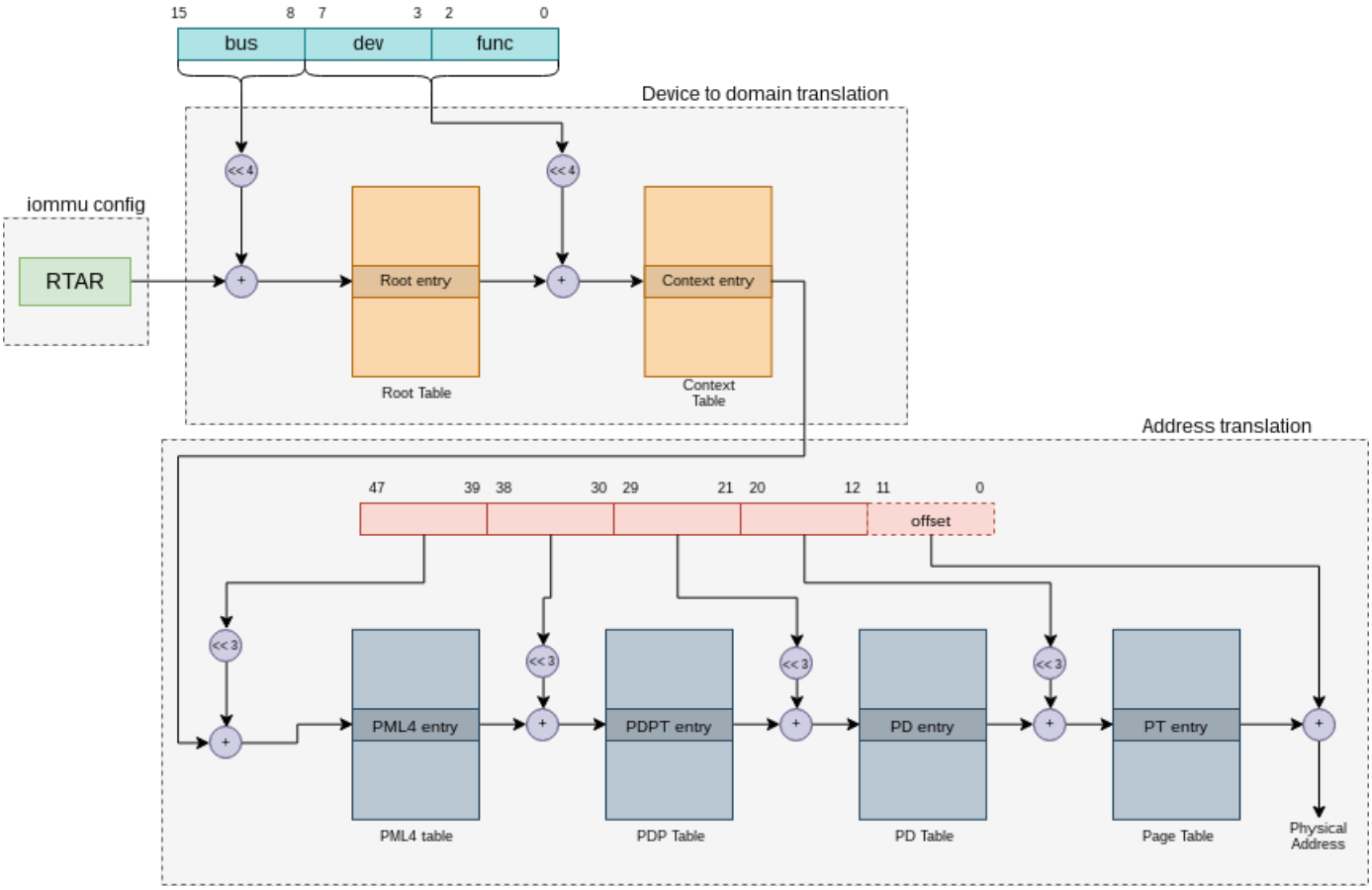

FIGURE 2: DMA *remapping* d'une adresse de 48 bits, avec des pages de 4096 octets

Ainsi, les adresses manipulées par les périphériques peuvent être vues comme des adresses virtuelles traduites en adresses physiques.

En plus de cela, VT-d propose de regrouper les périphériques par "domaines", chaque domaine se voyant associer une configuration de MMU spécifique. Plusieurs périphériques partagent alors le même espace d'adressage s'ils appartiennent au même domaine.

Chaque périphérique est identifié par le triplet PCI `bus:dev:fun`, duquel est dérivé le domaine auquel il appartient. Celui-ci permet de traduire l'adresse virtuelle en adresse physique.

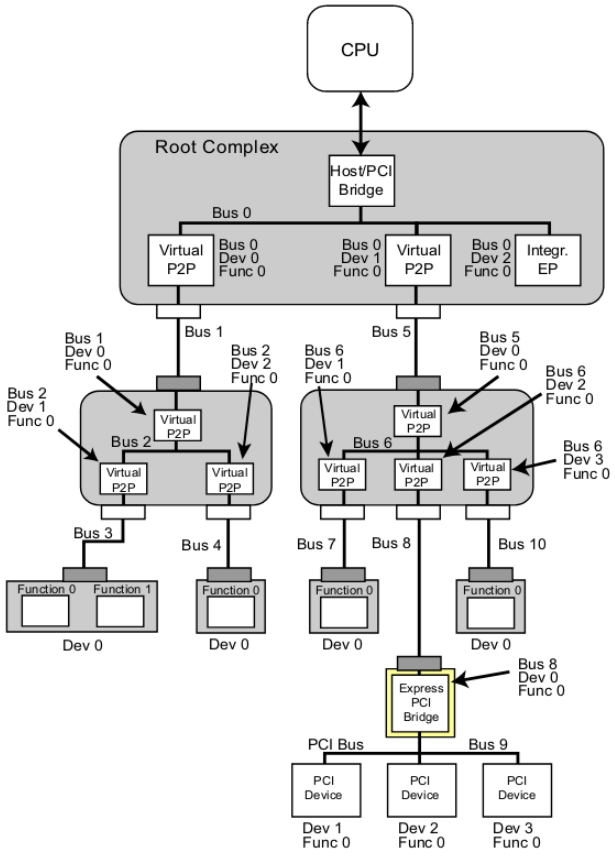

FIGURE 3: Topologie PCIe comprenant IOMMU, *switches* et périphériques PCIe[20]

Intel propose plusieurs cas d'usage de l'IOMMU grâce à VT-d :

— **contexte hyperviseur** : les solutions de virtualisation permettent de partager les ressources matérielles disponibles à des machines virtuelles. À titre d'exemple, l'hyperviseur est en mesure d'assigner une carte graphique à une machine virtuelle en particulier. Cette machine virtuelle

pourra alors configurer cette carte graphique qui, étant capable de faire du DMA, pourra accéder à des zones mémoires arbitraires et potentiellement réaliser un *vm-escape*. Si l'espace mémoire accessible via DMA est restreint et correctement configuré par une IOMMU, alors un périphérique assigné à une machine virtuelle ne pourra accéder qu'à de la mémoire déjà attribuée à cette machine virtuelle. L'isolation des machines virtuelles est alors effective.

— **système d'exploitation** : un système d'exploitation utilisera principalement VT-d afin de protéger son intégrité vis-à-vis de périphériques externes malveillants (ce qui correspond à notre cas d'usage). Le principe est de ne permettre aux périphériques d'accéder qu'aux zones mémoire légitimes, comme les *buffers* de données dans le cas d'une carte réseau. Notons que la granularité la plus fine de VT-d étant la page (4096 octets), il parait complexe de s'assurer qu'aucune structure *kernel* ne soit accessible via DMA.

2.2 Implémentation sous Windows

L'IOMMU est sollicitée comme moyen de protection sous Microsoft Windows [6] lors de l'utilisation des technologies suivantes :

— la solution de virtualisation Hyper-V.
— la solution de sécurisation *Virtualization Based Security* (VBS) basée sur Hyper-V.

Contrairement aux systèmes *NIX et macOS, les détails d'implémentation du support de l'IOMMU par Windows ne sont pas accessibles publiquement. La suite de ce document se repose donc sur la documentation fournie par les équipes de Microsoft ainsi que sur divers travaux de rétro-ingénierie accomplis par d'autres chercheurs.

Les auteurs de ce papier n'ayant pas encore étudié en détail l'utilisation de l'IOMMU par Microsoft Windows, les éléments suivants sont fournis à titre informatif.

Hyper-V Hyper-V est la solution de virtualisation mise en place par Microsoft. Elle permet, entre autres, la création de machines virtuelles au même titre que les logiciels VirtualBox [7] et VMWare Workstation [8]. La différence entre Hyper-V et ces autres solutions se situe dans la technique de virtualisation.

Dans le cas d'une installation basique d'Hyper-V, une partition principale est créée "au-dessus" de l'hyperviseur. Cette partition racine (*root partition*) est celle dans laquelle évoluera l'utilisateur final.

Virtualization Based Security En plus du cas d'utilisation basique, Microsoft Windows offre la possibilité de protéger certaines ressources critiques du système d'exploitation via Hyper-V :

— secrets d'authentifications

6. au moins jusqu'à la version 1803
7. https://www.virtualbox.org/
8. https://www.vmware.com/fr/products/workstation-pro.html

— intégrité du système en bloquant les applications identifiées comme n'étant pas de confiance
— etc.

Cette nouvelle couche de sécurité n'est possible que via l'activation de la *Virtualization Based Security* (VBS) [9], elle même reposant sur Hyper-V. Dans ce cas spécifique, deux environnements de niveaux de sécurité différents sont créés par défaut :

— VTL0, représentant l'environnement "normal" [10] et comprenant le noyau du système d'exploitation[1]
— VTL1, représentant l'environnement "sécurisé" ou *Virtual Secure Mode* (VSM) et contenant un *micro-kernel* ainsi que des *trustlets* (contextes d'exécution sécurisés, type HVCI, KMCI, *CredentialGuard*, etc.)[1].

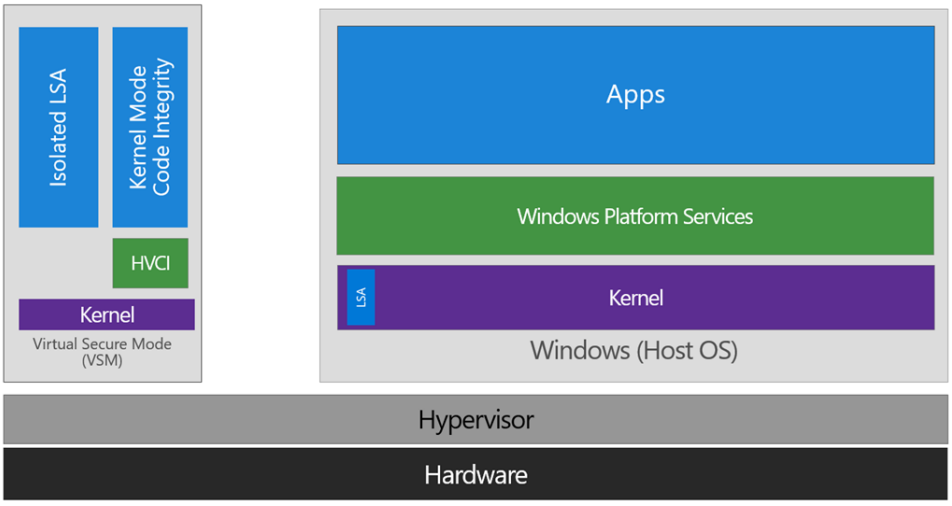

FIGURE 4: Système avec Hyper-V et VBS ainsi que quelques *trustlets*[7]

Mécanismes anti-DMA Comme indiqué en introduction de cette partie, peu d'informations sont disponibles concernant l'utilisation de l'IOMMU par les systèmes Windows.

Microsoft déclare que VBS repose sur l'utilisation de l'IOMMU afin de garantir une ségrégation entre environnements sécurisés. Néanmoins, cette affirmation ne peut être vérifiée que via rétro-ingénierie, ce travail n'ayant pas encore été réalisé.

9. L'activation de ce mécanisme de sécurité requiert du matériel compatible[4].
10. L'utilisateur final évolue dans cet environnement

Parmi les contre-mesures offertes pour se prémunir des attaques DMA, Microsoft Windows [11] désactive le chargement du pilote de périphériques *FireWire* à la page de *login* du système.

Un administrateur peut également spécifier une liste de GUID[16] identifiant la classe des périphériques de type *FireWire* afin que ceux-ci ne soient plus utilisables par le système.

De façon analogue, il semble possible de configurer le système afin que les ports permettant un accès au BUS PCI soient désactivés lorsque le système est verrouillé[18]. Néanmoins, la documentation détaillant cette option est très floue et n'explicite pas les mécanismes sous-jacents.

Enfin, Microsoft annonce en 2019[9][19] que la version 1809 de son système d'exploitation utilise l'IOMMU pour se prémunir des attaques DMA. Ainsi Microsoft affirme que l'IOMMU serait utilisée afin d'empêcher l'utilisation de périphériques *Thunderbolt 3* pour faire du DMA lorsque le système est verrouillé. Cependant, une fois encore, peu de détails sont fournis sur l'implémentation d'une telle contre-mesure.

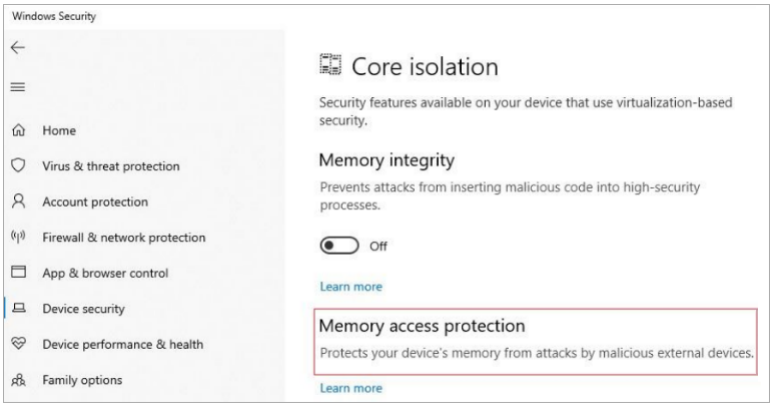

FIGURE 5: Capture d'écran montrant la fonctionnalité de protection contre les attaques DMA basées sur *Thunderbolt 3* sur un système *Windows Insiders Preview*

2.3 Implémentation sous Linux

Le code chargé de configurer l'IOMMU pour les différentes plateformes se trouve dans le dossier `linux/drivers/iommu/`. Ce dossier contient aussi du code permettant d'interagir avec des hyperviseurs, comme Hyper-V.

Une grande diversité de plateforme est prise en compte :

11. pour les systèmes à partir de Windows 8.1

— Intel ;
— AMD ;
— Exynos ;
— MicroTik ;
— etc.

L'IOMMU n'est pas activée par défaut sur Linux. Afin d'en bénéficier, il est nécessaire de spécifier l'argument de démarrage `intel_iommu=on`[14] (respectivement `amd_iommu=on` pour les processeurs AMD[2]). Chaque type d'IOMMU, correspondant à une plateforme différente, définit une structure `struct iommu_ops` servant de couche d'abstraction et d'interface pour interagir avec le matériel[3].

```
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,
};
```

Listing 1: Structure `iommu_ops`[3]

Les primitives de *mapping* d'adresse sont relativement simples. Elles permettent d'associer à une adresse physique (`paddr`) une adresse virtuelle vue depuis le périphérique (`iova`) en y associant des droits de lecture et/ou écriture. Ces *mappings* se font par domaine et non pas par périphérique. Un appel à ces fonctions peut donc servir pour plusieurs périphériques.

```
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
);
```

Listing 2: Fonctions `map` et `unmap`

Notons que chaque périphérique détient son propre domaine. L'adressage est donc différent pour chaque périphérique.

2.4 Implémentation sous macOS

macOS est probablement le système d'exploitation utilisant le plus les fonctionnalités physiques de sécurité. Ceci s'explique probablement car Apple maîtrise entièrement la chaîne de construction de ses produits, du matériel au logiciel, et n'a donc aucun problème de compatibilité.

À l'opposé, Linux et Windows se doivent d'être le plus générique possible, et ne peuvent donc pas imposer de fonctionnalités trop dépendantes d'un matériel en particulier.

Ainsi, macOS active par défaut l'IOMMU dès le démarrage, au sein de l'UEFI. Ce composant n'étant pas *open source*, un travail de rétro-ingénierie a été commencé mais reste à terminer.

Les premiers résultats montrent que l'implémentation est conforme à ce que préconise Intel dans son *white paper* [15]. Apple expose un protocole (au sens UEFI) personnalisé permettant aux *drivers* de configurer les *mappings* de l'IOMMU par périphérique. Ce protocole est notamment utilisé par le pilote exposant le *Target Disk Mode*[29], pour rendre possible la communication avec le contrôleur Thunderbolt 3 et le disque NVMe.

Lorsque l'UEFI transfère l'exécution au système d'exploitation macOS, le *driver* `IOPCIFamily` se charge de ré-initialiser l'IOMMU pour qu'elle soit utilisable dans le nouveau contexte[6].

Ce pilote déclare alors une classe `AppleVTDDeviceMapper` surchargeant `IOMapper`. Cette classe redéfinit les deux méthodes `iovmMapMemory` et `iovmUnmapMemory` permettant respectivement d'ajouter ou de supprimer des *mappings* mémoire dans l'IOMMU.

Contrairement à Linux, macOS n'utilise qu'un seul domaine pour tous les périphériques.

3 Attaques existantes

Cette partie porte sur les différentes attaques reposant sur un accès DMA. Seules les attaques logicielles sont présentées et n'est uniquement considéré que le cas d'un accès physique à un poste de travail verrouillé, sur lequel l'attaquant souhaite avoir un accès total.

S'il est possible de réaliser des attaques via le protocole *FireWire*, seules les attaques basées sur la technologie *PCI Express* sont exposées. En effet, les constructeurs d'ordinateurs n'offrent plus par défaut de connectique *FireWire* contrairement à la connectique basée sur norme M.2 et le Thunderbolt via USB-C.

3.1 pcileech sous Windows

Déverrouillage d'un poste n'utilisant pas l'IOMMU

Dans le but de déverrouiller un poste sous Windows, l'approche du logiciel *pcileech* consiste à altérer la routine de vérification d'identité d'un utilisateur. Pour cela, cette application parcourt la mémoire physique de l'ordinateur ciblé en intégralité à la recherche de motifs permettant d'identifier la routine. Une fois les motifs identifiés, le code responsable de la vérification d'identité est modifié en mémoire afin d'autoriser l'authentification de l'utilisateur, quel que soit le mot de passe fourni.

`MsvpPasswordValidate` est la fonction en charge de valider l'identité d'un utilisateur. Elle est présente dans la bibliothèque partagée `msv1_0.dll` jusqu'à Windows 8.1 inclus et dans `NtlmShared.dll` à partir de Windows 10.

L'extrait de code assembleur [12] en charge de comparer l'entrée utilisateur aux secrets stockés par le système est le suivant :

```
.text:0000000180003730 41 BE 10 00 00 00                    mov     r14d, 10h
.text:0000000180003736 48 8D 56 50                          lea     rdx, [rsi+50h]  ; Source2
.text:000000018000373A 45 8B C6                             mov     r8d, r14d       ; Length
.text:000000018000373D 48 8B CB                             mov     rcx, rbx        ; Source1
.text:0000000180003740 FF 15 B2 1B 00 00                    call    cs:RtlCompareMemory
.text:0000000180003746 49 3B C6                             cmp     rax, r14
.text:0000000180003749 0F 84 0B FB FF FF                    jz      loc_18000325A
.text:000000018000374F
.text:000000018000374F                          loc_18000374F:
.text:000000018000374F
.text:000000018000374F 32 C0                                xor     al, al
.text:0000000180003751 E9 06 FB FF FF                       jmp     loc_18000325C
```

Listing 3: NtlmShared.dll, version 10.0.17763.1, Windows 10 x64

12. Code désassemblé par le logiciel IDA Pro

Cet extrait est décompilé [13] de la sorte :

```
// [...]
   else if ( RtlCompareMemory(v8, (const void *)(v9 + 80), 0x10ui64) == 16 )
    {
      return 1;
    }
    return 0;
```

Listing 4: Vérification de l'entrée utilisateur

Aussi, la méthodologie de *pcileech* consiste à trouver en mémoire physique les motifs correspondants aux *opcodes* recherchés et les altérer de sorte que le programme retourne toujours 1. La manière dont le code doit être modifié peut varier en fonction de la version de la DLL. Dans certains cas, il peut être suffisant de changer la condition d'un saut conditionnel, ou alors de simplement retirer un ensemble d'*opcodes* comme dans cet exemple :

```
.text:0000000180003730 41 BE 10 00 00 00                    mov     r14d, 10h
.text:0000000180003736 48 8D 56 50                          lea     rdx, [rsi+50h]  ; Source2
.text:000000018000373A 45 8B C6                             mov     r8d, r14d       ; Length
.text:000000018000373D 48 8B CB                             mov     rcx, rbx        ; Source1
.text:0000000180003740 FF 15 B2 1B 00 00                    call    cs:RtlCompareMemory
.text:0000000180003746 49 3B C6                             cmp     rax, r14
.text:0000000180003749 90                                   nop
.text:000000018000374A 90                                   nop
.text:000000018000374B 90                                   nop
.text:000000018000374C 90                                   nop
.text:000000018000374D 90                                   nop
.text:000000018000374E 90                                   nop
.text:000000018000374F
.text:000000018000374F                          loc_18000374F:
.text:000000018000374F
.text:000000018000374F B0 01                                mov     al, 1
.text:0000000180003751 E9 06 FB FF FF                       jmp     loc_18000325C
```

Listing 5: Routine de vérification modifiée pour retourner 1

13. Code obtenu par le décompilateur de Hex-Rays[13]

```
// [...]
else
{
  RtlCompareMemory(v8, (const void *)(v9 + 80), 0x10ui64);
}
return 1;
```

Listing 6: La vérification retourne 1 quelle que soit l'entrée utilisateur

Pour pouvoir patcher de façon adéquate, *pcileech* nécessite que lui soit fourni un fichier contenant une ou plusieurs signatures susceptibles de correspondre aux motifs recherchés. Ces signatures sont de la forme `"offset1, opcodes1, offset2, opcodes2, offset3, opcodes3"` :

- **offset1 :** décalage du premier motif à rechercher par rapport au début de la page mémoire courante.
- **opcodes1 :** premier motif à rechercher en mémoire, sous forme d'*opcodes*.
- **offset2 (optionnel) :** décalage du second motif à rechercher par rapport au début de la page mémoire courante.
- **opcodes2 (optionnel) :** second motif à rechercher en mémoire, sous forme d'*opcodes*.
- **offset3 :** décalage du patch à appliquer par rapport au début de la page mémoire courante.
- **opcodes3 :** patch à appliquer à l'*offset3*, sous forme d'*opcodes*.

Dans l'exemple abordé précédemment, la signature permettant de produire un tel patch est la suivante [14] :

```
748,C60F840BFBFFFF32C0E906,0,-,749,909090909090B001
```

Après avoir modifié la mémoire physique de l'ordinateur attaqué, l'attaquant peut avoir accès à un compte donné, quel que soit le mot de passe qu'il saisit.

Déverrouillage d'un poste utilisant l'IOMMU

Nous avons vu que Windows tire parti de l'IOMMU lorsque la *Virtualization Based Security* (VBS) est activée (voir Implémentation sous Windows page 4).

Néanmoins, lorsque les prérequis matériels ne sont pas satisfaits lors de la séquence de *boot*, le système d'exploitation désactive VBS afin de garantir un démarrage réussi. Ce comportement spécifique de Windows est exploitable par un attaquant pour s'abstraire des mécanismes de protection issus de l'IOMMU[24].

En effet, lors de la phase de démarrage le système d'exploitation consulte la table ACPI DMAR [15]. Cette table est une structure en mémoire décrivant,

14. Le script "search_offsets_DMA.py"[17] a été utilisé pour déterminer ces *offsets*
15. "DMAR" est l'abréviation de "DMA Remapping"

entres autres, l'adresse à laquelle se trouvent les registres de configuration de l'IOMMU. Dès lors, si l'OS ne parvient pas à trouver cette table ACPI, il n'est pas capable de configurer l'IOMMU.

L'attaque consiste donc à identifier en mémoire cette table puis la corrompre, en altérant simplement le premier champ [16]. Cette phase d'identification est triviale car la table commence par la signature `DMAR` et l'adresse en mémoire est fixe d'un redémarrage à un autre.

Le système d'exploitation, ne trouvant pas d'IOMMU, démarre en désactivant VBS.

3.2 pcileech sous Linux

La méthodologie pour déverrouiller un ordinateur sous Linux via *pcileech* étant identique à celle pour Windows, elle ne sera pas explicitée dans cet article. Simplement, la fonction *verify_pwd_hash* de la bibliothèque *pam_unix.so* est altérée afin de déverrouiller la session utilisateur.

3.3 pcileech sous macOS

Déverrouillage d'un poste n'utilisant pas l'IOMMU

pcileech supporte macOS jusqu'à la version Sierra (10.12.x)[27]. Par défaut, macOS se protège contre les périphériques malveillants en utilisant une IOMMU. Néanmoins, il est possible de désactiver cette protection à l'aide du mode *recovery*. Ce mode est accessible librement, excepté si un mot de passe EFI a été défini.

Le mode *recovery* est accessible en appuyant sur les touches *Command+R* lors du démarrage. Il suffit ensuite de positionner la variable *nvram* `boot-args` à `dart=0x0` [17]. Cet argument de *boot* permet de spécifier au *driver* `IOPCIFamily` de ne pas activer l'IOMMU. Il est alors possible, au prochain démarrage, d'accéder à toute la mémoire physique depuis un périphérique externe.

Cette attaque reste relativement contraignante dans la mesure où elle nécessite un redémarrage de la machine cible. De plus, si cette dernière est protégée par *FileVault2*[5], il n'est alors plus possible d'accéder aux données. Enfin, si un mot de passe EFI a été défini par l'utilisateur il est également impossible d'accéder aux données.

Lorsque l'IOMMU n'est pas utilisée, le déverrouillage de l'ordinateur s'effectue suivant la même méthodologie que pour Windows en altérant la routine de vérification du mot de passe. La fonction à modifier en mémoire est *ODRecordVerifyPassword* de la bibliothèque *OpenDirectory.framework* (Listing 7). Cette fonction est utilisée par le module *pam* `pam_opendirectory.so.2` afin de vérifier le mot de passe spécifié par l'utilisateur. Une fois cette fonction patchée, il est alors possible de se connecter avec n'importe quel mot de passe.

16. Cette attaque n'est réalisable qu'en cas d'absence de mécanismes de pré-authentification lors de la phase de démarrage

17. À partir de la version 10.13.x, l'option `dart` n'est plus disponible.

```
SIGNATURE oSigs[NUMBER_OF_SIGNATURES] = {
        { .chunk = { // CFOpenDirectory!ODRecordVerifyPassword (El Capitan | 466064 bytes)
                { .cbOffset = 0xfce,.cb = 6,.pb = { 0xe8, 0x69, 0xc4, 0x00, 0x00, 0xeb, 0x02, 0x31 } },
                { .cbOffset = 0xfd3,.cb = 6,.pb = { 0xeb, 0x02, 0x31, 0xdb, 0x88, 0xd8, 0x48, 0x83 } },
                { .cbOffset = 0xfd7,.cb = 2,.pb = { 0xb0, 0x01 } } }
        },
        { .chunk = { // CFOpenDirectory!ODRecordVerifyPassword (El Capitan | 466064 bytes)
                { .cbOffset = 0x134,.cb = 8,.pb = { 0x08, 0x00, 0x00, 0x00, 0x4c, 0x89, 0xf7, 0xe8 } },
                { .cbOffset = 0x13c,.cb = 8,.pb = { 0x3e, 0xc4, 0x00, 0x00, 0xeb, 0x02, 0x31, 0xdb } },
                { .cbOffset = 0x144,.cb = 2,.pb = { 0xb0, 0x01 } } }
        },
        { .chunk = { // CFOpenDirectory!ODRecordVerifyPassword (Sierra 10.12.3)
                { .cbOffset = 0x130,.cb = 8,.pb = { 0x08, 0x00, 0x00, 0x00, 0x4c, 0x89, 0xf7, 0xe8 } },
                { .cbOffset = 0x138,.cb = 8,.pb = { 0x3e, 0xc4, 0x00, 0x00, 0xeb, 0x02, 0x31, 0xdb } },
                { .cbOffset = 0x140,.cb = 2,.pb = { 0xb0, 0x01 } } }
        },
        { .chunk = { // CFOpenDirectory!ODRecordVerifyPassword (Sierra 10.12.4)
                { .cbOffset = 0x130,.cb = 8,.pb = { 0x08, 0x00, 0x00, 0x00, 0x4c, 0x89, 0xf7, 0xe8 } },
                { .cbOffset = 0x138,.cb = 8,.pb = { 0x1a, 0xc4, 0x00, 0x00, 0xeb, 0x02, 0x31, 0xdb } },
                { .cbOffset = 0x140,.cb = 2,.pb = { 0xb0, 0x01 } } }
        },
};
```

Listing 7: Signature macOS de pcileech

Filevault2 de macOS

Filevault2 est la solution de Apple permettant de chiffrer entièrement le disque dur des ordinateurs sous macOS. *pcileech* supporte l'extraction du mot de passe *Filevault2* de la mémoire avant la version 10.12.2[26][25].

Cette extraction est possible pour plusieurs raisons :

— l'EFI de Apple n'active pas l'IOMMU avant que le système d'exploitation principal ne soit chargé. Or, le contrôleur Thunderbolt est configuré pour pouvoir supporter le mode TDM[29]. Un périphérique Thunderbolt peut donc faire des accès DMA lors de la phase de démarrage.
— le mot de passe *Filevault2* est stocké en clair dans la mémoire et n'est pas effacé lors d'un redémarrage. De plus, l'adresse à laquelle se trouve cette chaîne de caractères diffère peu d'un redémarrage à l'autre.

Dans le cas d'un mac allumé et chiffré avec *Filevault2*, l'attaque consiste donc à brancher un périphérique Thunderbolt puis redémarrer. L'attaquant dispose alors d'une fenêtre de quelques secondes au redémarrage pour lire le mot de passe *Filevault2* en clair avant que la zone mémoire ne soit réutilisée.

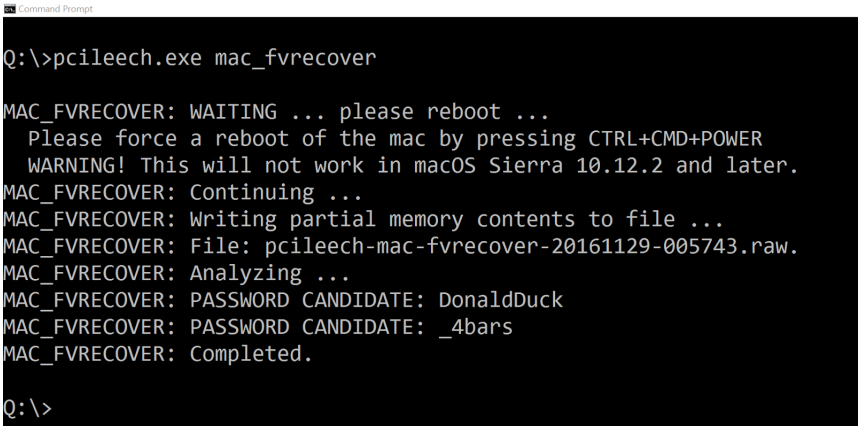

FIGURE 6: Extraction du mot de passe *Filevault2* avec *pcileech*

Cette vulnérabilité a été corrigée par Apple dans la version 10.12.3 en activant l'IOMMU (VT-d) dès l'EFI :

EFI:
— **Available for**: macOS Sierra 10.12.3
— **Impact**: A malicious Thunderbolt adapter may be able to recover the FileVault 2 encryption password
— **Description**: An issue existed in the handling of DMA. This issue was addressed by enabling VT-d in EFI.
— **CVE-2016-7585**: Ulf Frisk (@UlfFrisk)

3.4 Thunderclap

Thunderclap est le nom donné par ses auteurs au papier traitant de vulnérabilités dans les systèmes d'exploitation au niveau de l'implémentation des protections par IOMMU. Ces vulnérabilités sont relatives aux accès DMA via périphériques externes n'étant pas de confiance [18]. Ces travaux ont été réalisés dans le cadre de la thèse de *Colin Rothwell*[21].

La partie innovante de ces recherches concerne la description d'une nouvelle attaque sur macOS. Cette attaque permet d'exécuter du code arbitraire avec les privilèges les plus élevés (*root*) alors que l'IOMMU est activée.

Étant donné que tous les périphériques PCIe sont dans le même domaine, ceux-ci ont tous accès au même espace d'adressage. Il est ainsi possible d'accéder aux pages utilisées par la carte réseau pour lire et écrire les données devant transiter. macOS utilise la structure `mbuf` pour décrire les paquets devant être émis sur le réseau. Ces structures sont donc visibles dans la mémoire depuis un périphérique externe.

18. "Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals"[8]

```
/* header at beginning of each mbuf: */
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 */
};

/*
 * 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;
};

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;
};
```

Listing 8: Structure `struct mbuf`

Des macros sont définies afin d'accéder aux champs de façon claire. Celles-ci seront employées dans le reste du document :

— m_next : m_hdr.mh_next
— m_len : m_hdr.mh_len
— m_data : m_hdr.mh_data
— m_type : m_hdr.mh_type
— m_flags : m_hdr.mh_flags
— m_nextpkt : m_hdr.mh_nextpkt
— m_pkthdr : M_dat.MH.MH_pkthdr
— m_ext : M_dat.MH.MH_dat.MH_ext
— m_pktdat : M_dat.MH.MH_dat.MH_databuf
— m_dat : M_dat.M_databuf

Lorsque le flag M_EXT est spécifié dans le champ m_flags, le champ m_ext est utilisé. De base, un mbuf permet de stocker au maximum MLEN octets. Le champ m_ext permet de gérer un buffer externe pour le stockage des données du mbuf. Le champ m_ext.ext_free est utilisé pour stocker un pointeur permettant de libérer le buffer externe.

Lorsque la fonction ext_free est appelée, les champs ext_buf, ext_size et ext_arg sont passés en paramètres. Dans la mesure où toutes ces données sont accessibles en lecture et écriture via DMA, il est possible de modifier le pointeur de fonction et de détourner le flux d'exécution du noyau vers une fonction arbitraire, tout en contrôlant les trois premiers arguments.

La fonction du noyau KUNCExecute [19] permet de lancer un exécutable en tant que *root* dans le *userland*. Il suffit donc de modifier un m_ext.ext_free et d'attendre que le mbuf modifié soit libéré pour, par exemple, lancer Terminal.app en tant que *root*.

Apple a corrigé ce problème en masquant les pointeurs m_ext.ext_free et m_ext.ext_refflags avec deux valeurs aléatoires définies lors du démarrage du système.

```
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));
```

Listing 9: mb_obscure_extfree et mb_obscure_extref

Il n'est donc plus possible de modifier ces pointeurs sans connaître les valeurs de ces masques. Cette contre-mesure rend ainsi caduque l'attaque à partir de macOS 10.12.4.

4 Conclusions

Comme escompté, macOS demeure le système d'exploitation le plus avancé en matière de protections DMA. Bien que les autres systèmes d'exploitation grand

19. https://developer.apple.com/documentation/kernel/1434510-kuncexecute

public tentent de combler leur retard, celui d'Apple reste le seul à proposer l'utilisation de l'IOMMU par défaut.

Les différentes attaques présentées au sein de ce papier visent en particulier les systèmes d'exploitation dans leur configuration de base. Aussi, il serait intéresserant d'étudier les contre-mesures existantes nécessitant un durcissement du socle système. Par ailleurs, peu de ressources[10][11][12] existent concernant l'implémentation de l'IOMMU en elle-même et son niveau de sécurité selon les plateformes. Les auteurs de ce papier suivront ces axes de recherche dans le cadre du projet RAPID `DMArvest`.

Bibliographie

[1] Adrien Chevalier - Amossys. *Virtualization Based Security - Part 1: The boot process*. [Online; accessed September 03, 2019]. 2017. URL : https://blog.amossys.fr/virtualization-based-security-part1.html.

[2] Advanced Micro Devices, Inc. *amd_iommu_init.c*. [Online; accessed September 03, 2019]. 2010. URL : https://elixir.bootlin.com/linux/latest/source/drivers/iommu/amd%5C_iommu%5C_init.c#L2943.

[3] Advanced Micro Devices, Inc. *iommu.h*. [Online; accessed September 03, 2019]. 2008. URL : https://elixir.bootlin.com/linux/v5.2.11/source/include/linux/iommu.h#L180.

[4] ANSSI. *Mise en œuvre des fonctionnalités de sécurité de Windows 10 reposant sur la virtualisation*. [Online; accessed September 03, 2019]. 2017. URL : https://www.ssi.gouv.fr/uploads/2017/11/np_securisation_windows10_securite_reposant_sur_la_virtualisation_v1.pdf.

[5] Apple. *Chiffrement du disque de démarrage d'un Mac à l'aide de FileVault*. [Online; accessed September 03, 2019]. 2018. URL : https://support.apple.com/fr-fr/HT204837.

[6] Apple Inc. *IOPCIFamily-330.250.10*. [Online; accessed September 03, 2019]. 2019. URL : https://opensource.apple.com/source/IOPCIFamily/IOPCIFamily-330.250.10/.

[7] Ash de Zylva. *Windows 10 Device Guard and Credential Guard Demystified*. [Online; accessed September 03, 2019]. 2016. URL : https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/.

[8] Colin Rothwell. *Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals*. [Online; accessed September 03, 2019]. 2019. URL : http://thunderclap.io/thunderclap-paper-ndss2019.pdf.

[9] David Weston - Microsoft. *Advancing Windows Security - Bluehat Shanghai*. [Online; accessed September 03, 2019]. 2019. URL : https://github.com/dwizzzle/Presentations/blob/master/Bluehat%20Shanghai%20-%20Advancing%20Windows%20Security.pdf.

[10] Eric Lacombe, Fernand Lone Sang, Vincent Nicomette, Yves Deswarte. *Analyse de l'efficacité du service fourni par une IOMMU*. [Online; accessed November 04, 2019]. 2010. URL : https://www.sstic.org/media/SSTIC2010/SSTIC-actes/Analyse_de_l_efficacite_du_service_fourni_par_une_/SSTIC2010-Article-Analyse_de_l_efficacite_du_service_fourni_par_une_IOMMU-lacombe_lone-sang_nicomette_deswarte.pdf.

[11] Fernand Lone Sang. *Protection des systèmes informatiques contre les attaques par entrées-sorties*. [Online; accessed November 04, 2019]. 2012. URL : https://www.theses.fr/2012ISAT0051.

[12] Fernand Lone Sang, Loic Duflot, Vincent Nicomette, Yves Deswarte. *Attaques DMA peer-to-peer et contremesures*. [Online; acces-

sed November 04, 2019]. 2011. URL : https://www.sstic.org/media/SSTIC2011/SSTIC-actes/attaques_dma_peer-to-peer_et_contremesures/SSTIC2011-Article-attaques_dma_peer-to-peer_et_contremesures-lone-sang_duflot_nicomette_deswarte.pdf.

[13] HEX-RAYS. *IDA: About*. [Online; accessed September 17, 2019]. 2019. URL : https://www.hex-rays.com/products/ida/.

[14] INTEL CORPORATION. *intel-iommu.c*. [Online; accessed September 03, 2019]. 2014. URL : https://elixir.bootlin.com/linux/latest/source/drivers/iommu/intel-iommu.c#L428.

[15] INTEL CORPORATION. *Using IOMMU for DMA Protection in UEFI Firmware*. [Online; accessed September 03, 2019]. 2017. URL : https://firmware.intel.com/sites/default/files/Intel_WhitePaper_Using_IOMMU_for_DMA_Protection_in_UEFI.pdf.

[16] JEAN-CHRISTOPHE DELAUNAY - SYNACKTIV. *Practical DMA attack on Windows 10*. [Online; accessed September 03, 2019]. 2018. URL : https://www.synacktiv.com/posts/pentest/practical-dma-attack-on-windows-10.html.

[17] JEAN-CHRISTOPHE DELAUNAY - SYNACKTIV. *search_offsets_DMA.py*. [Online; accessed September 26, 2019]. 2017. URL : https://github.com/Synacktiv-contrib/stuffz/blob/master/search_offsets_DMA.py.

[18] MICROSOFT. *BitLocker Countermeasures*. [Online; accessed September 03, 2019]. 2019. URL : https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-countermeasures.

[19] MICROSOFT. *Kernel DMA Protection for Thunderbolt™ 3*. [Online; accessed September 03, 2019]. 2019. URL : https://docs.microsoft.com/en-us/windows/security/information-protection/kernel-dma-protection-for-thunderbolt.

[20] MIKE JACKSON, RAVI BUDRUK. *PCI Express Technology, Comprehensive Guide to Generations 1.x, 2.x and 3.0*. [Ebook; available September 16, 2019]. 2012. URL : https://www.mindshare.com/Books/Titles/PCI_Express_Technology_3.0.

[21] Colin Lewis ROTHWELL. "EXPLOITATION FROM MALICIOUS PCI EXPRESS PERIPHERALS". Thèse de doct. Churchill College, sept. 2017.

[22] SAMSUNG. *Samsung 970 EVO NVMe M.2*. [Online; accessed September 16, 2019]. 2019. URL : https://static.macway.com/images/p/g/originalid_915000/300/915352/zoom/915352_7fdad14.jpg.

[23] STÉPHANE MOUSSIE. *Thunderbolt 3, USB-C, USB 3.1 : ce qu'il faut savoir*. [Online; accessed September 16, 2019]. 2015. URL : http://img.igen.fr/2015/6/macgpic-1433421562-12011488884021-sc-op.jpg.

[24] ULF FRISK. *Disable Virtualization Based Security (VBS) on auto-booting systems*. [Online; accessed September 13, 2019]. 2016. URL : http://blog.frizk.net/2016/11/disable-virtualization-based-security.html.

[25] ULF FRISK. *macOS FileVault2 Password Retrieval*. [Online; accessed September 19, 2019]. 2019. URL : http://blog.frizk.net/2016/12/filevault-password-retrieval.html.

[26] Ulf Frisk. *Target operating system: mac FileVault2 (EFI)*. [Online; accessed September 03, 2019]. 2019. URL : https://github.com/ufrisk/pcileech/wiki/Target-Mac-EFI-FileVault2.

[27] Ulf Frisk. *Target operating system: macOS*. [Online; accessed September 03, 2019]. 2019. URL : https://github.com/ufrisk/pcileech/wiki/Target-macOS.

[28] USB Implementers Forum, Inc. *USB4™ Specification*. [Online; accessed September 19, 2019]. 2019. URL : https://www.usb.org/document-library/usb4tm-specification.

[29] Wikipedia. *Target Disk Mode*. [Online; accessed September 03, 2019]. 2018. URL : https://en.wikipedia.org/wiki/Target_Disk_Mode.