Analyse de l'efficacité du service fourni par une IOMMU

Fernand Lone Sang[1,2], Éric Lacombe[1,2], Vincent Nicomette[1,2], and Yves Deswarte[1,2]
fernand.lone-sang(@)laas.fr,
eric.lacombe(@){laas.fr,security-labs.org},
vincent.nicomette(@)laas.fr, yves.deswarte(@)laas.fr

[1] CNRS ; LAAS ; 7 Avenue du colonel Roche, F-31077 Toulouse, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France

Résumé Aujourd'hui, il est difficile d'assurer la protection d'un noyau de système d'exploitation. Les attaques visant à le corrompre proviennent de sources diverses. La plupart utilisent le processeur comme vecteur d'accès à la mémoire du noyau. Cependant, depuis un certain nombre d'années, nous voyons apparaître des attaques impliquant des périphériques, en particulier depuis le bus FireWire. Pour se protéger de ces derniers, une IOMMU est nécessaire. C'est un composant matériel permettant à un système d'exploitation de contrôler l'accès des périphériques à la mémoire principale. Cet article s'intéresse au service fourni par une IOMMU vis-à-vis des attaques DMA. Nous présentons un cas d'étude axé sur la technologie Intel VT-d et le noyau Linux. L'analyse que nous en avons faite a révélé l'existence de vulnérabilités dans cette technologie. Nous détaillons l'une d'elles et nous l'illustrons par une preuve de concept. Finalement, nous formulons quelques recommandations pour empêcher son exploitation.

Mots-clés: IOMMU, Intel VT-d, sécurité, contrôle d'accès à la mémoire, vulnérabilité matérielle, pont PCI-PCI Express.

1 Introduction

Pour réduire les coûts de développement, les systèmes informatiques utilisent de plus en plus de composants sur étagère à la fois logiciels (systèmes d'exploitation, applications, bibliothèques partagées, etc.) et matériels (microprocesseurs, *chipsets*, etc.). Or, ces composants sont conçus pour être génériques et sont par conséquent complexes. Cette complexité les rend vulnérables à des attaques ou à des fautes d'autres composants avec lesquels ils interagissent.

Dans cet article, nous nous intéressons aux attaques utilisant l'accès direct à la mémoire par les périphériques pour corrompre le noyau d'un système d'exploitation. L'accès direct à la mémoire ou DMA (*Direct Memory Access*) est un mécanisme où les transferts de données depuis un périphérique vers la mémoire principale, ou inversement, sont effectués par un contrôleur spécifique. Grâce à ce mécanisme, le processeur se trouve déchargé des transferts E/S et n'intervient que pour initier et conclure ces transferts.

Les attaques DMA peuvent être classées en deux catégories [14] suivant que l'accès est commandé par le processeur ou à l'initiative du périphérique.

La première catégorie d'attaque concerne les périphériques qui nécessitent l'intervention du processeur pour mettre en place un accès à la mémoire. Une action malveillante visant à modifier la mémoire principale en utilisant le DMA requiert alors l'exécution d'un code depuis le système d'exploitation. Un exemple d'accès initié par le processeur est publié dans [10]. Dans cette preuve de concept, un code malveillant programme des transferts DMA dans les contrôleurs USB de type UHCI [11] dans l'optique de changer la valeur courante du `securelevel` dans un système OpenBSD.

La seconde catégorie concerne les périphériques qui initient eux-mêmes leurs accès. Ceux-ci intègrent nécessairement un contrôleur DMA et sont connectés à un bus d'E/S qui autorise les périphériques à en prendre le contrôle. Nous parlons alors de périphériques DMA *bus-masters*. De nombreuses preuves de concept existent qui s'appuient sur différents périphériques pour effectuer des accès DMA malveillants. Certaines mettent en jeu des cartes PCI reprogrammées afin de compromettre la mémoire du noyau [7,5]. Les cartes FireWire peuvent également être utilisées pour lire ou modifier la mémoire physique [4,9,8,16,2]. Enfin, une preuve de concept utilisant des périphériques USB « *On-The-Go* » (OTG)[1] de type OHCI [18] a été publiée [17].

Sur les plate-formes matérielles récentes, il est possible de contrôler ces accès par une IOMMU (*Input/Output Memory Management Unit*) [1,12]. Il s'agit d'un composant matériel qui agit comme

1. L'USB OTG est une extension de l'USB standard. Elle permet aux périphériques de communiquer entre eux dans un mode pair-à-pair.

un pare-feu et filtre les accès en provenance des périphériques vers la mémoire principale.

Cet article évalue le service de contrôle d'accès à la mémoire fourni par ce composant matériel. Nous focalisons notre attention sur la technologie Intel *Virtualization Technology for Directed Input/Output* (VT-d) [12]. Son fonctionnement est décrit en section 2. En section 3, nous présentons les bénéfices d'une telle technologie pour un système d'exploitation au travers de la mise en œuvre qui en est faite sous Linux. Nous présentons ensuite en section 4 les vulnérabilités que nous avons pu identifier dans cette technologie. Une preuve de concept pour l'une d'elles est fournie en section 5. Enfin, nous donnons quelques recommandations en section 6 pour empêcher son exploitation avant de conclure et de présenter quelques perspectives à nos travaux en section 7.

2 Description de la technologie Intel VT-d

La technologie Intel *Virtualization Technology for Directed Input/Output* (VT-d) [12] implémente une IOMMU. Elle rend possible la virtualisation des périphériques d'entrée/sortie (E/S). Cette extension matérielle agit au niveau du *Memory Controller Hub* (MCH) du *chipset* et se place généralement dans le *northbridge*, en coupure entre les périphériques et le bloc processeur-mémoire. La figure 1 présente l'architecture d'un système qui contient Intel VT-d.

Cette section propose un aperçu rapide de cette technologie. Nous commençons par décrire les différents composants matériels qui la constituent. Puis, nous détaillons les mécanismes qui leur permettent de contrôler les accès à la mémoire.

2.1 Architecture matérielle d'Intel VT-d

L'extension matérielle Intel VT-d s'appuie sur un ensemble d'unités matérielles de traduction DMA ou *DMA Remapping Hardware Units* (DRHU). Ces unités sont les entités élémentaires chargées de traduire les accès effectués par les périphériques vers la mémoire principale. Elles vérifient également leurs droits pour l'accès mémoire demandé.

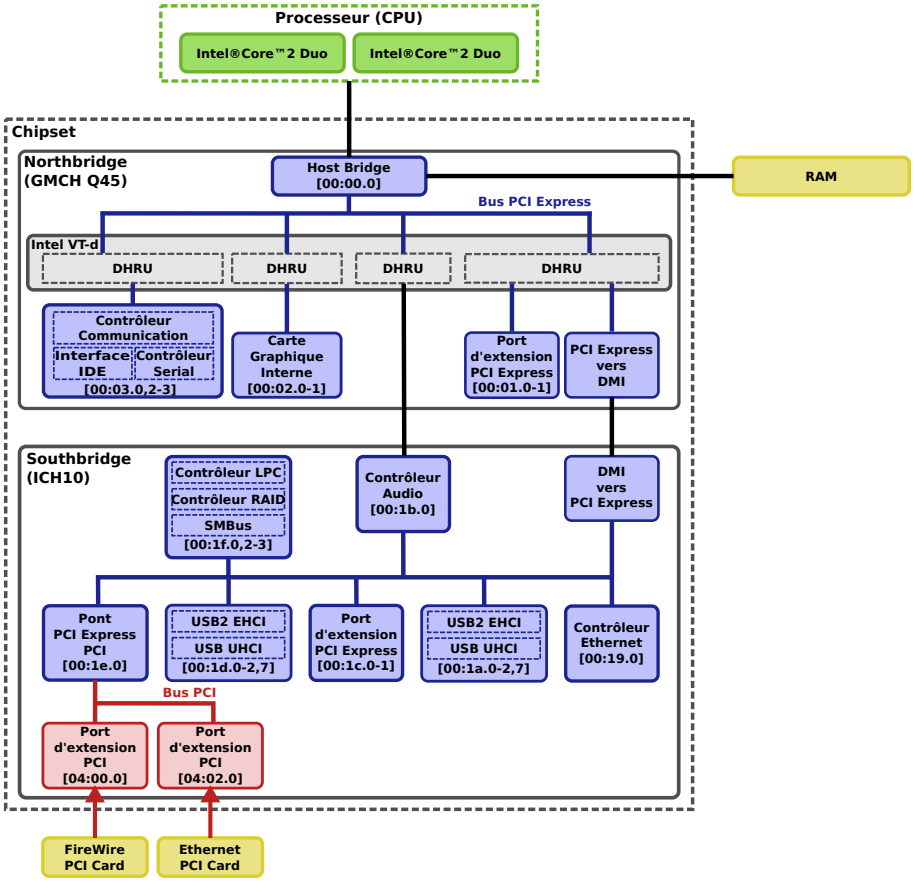

Figure 1. Architecture d'un *chipset* (Intel Q45) contenant la technologie VT-d

Les DRHU (*cf.* figure 1) sont localisées dans le *northbridge* du *chipset*. Plus précisément, nous les retrouvons sur le canal de communication qui relie la mémoire principale et les périphériques, c'est-à-dire le bus d'E/S 0. Durant la phase d'initialisation de la machine, le BIOS est en charge de les détecter et de les initialiser pour le système d'exploitation. Notamment, il définit pour chaque unité l'ensemble des périphériques dont elle doit contrôler les accès vers la mémoire. Le noyau configure ensuite la politique d'accès à la mémoire à appliquer pour chaque périphérique.

Dans la suite de cette section, nous nous plaçons au niveau d'une *DMA Remapping Hardware Unit*. Nous détaillons les différents mécanismes impliqués dans le contrôle d'accès des périphériques à la mémoire. Nous présentons également les structures de données mises en jeu.

2.2 Mécanismes de traduction d'adresses DMA et de contrôle d'accès à la mémoire pour les périphériques

La technologie Intel VT-d contrôle l'accès des périphériques d'E/S à la mémoire principale et virtualise l'espace d'adressage physique pour ces derniers. Par conséquent, ils n'accèdent plus directement à l'espace d'adressage physique mais à un espace d'adressage virtualisé par une DRHU. L'accès à la mémoire principale s'effectue par le biais d'une adresse virtuelle DMA (appelée DVA, pour *DMA Virtual Address*). Elle est traduite en adresse physique (HPA ou *Host Physical Address*) lors du transit de la requête d'accès direct à la mémoire dans une DRHU.

Cette section se focalise sur les mécanismes qui permettent à une DRHU de décider de relayer un accès au contrôleur mémoire ou de la bloquer. Ces mécanismes suivent un schéma d'identification et d'autorisation pour contrôler l'accès des périphériques d'E/S à la mémoire. Nous supposons qu'un accès à la mémoire depuis un périphérique a été intercepté par une DRHU. Voyons comment s'effectue l'identification du périphérique et la détermination de l'espace d'adressage virtuel pour le DMA qui en découle.

Identification des périphériques effectuant l'accès à la mémoire et détermination de l'espace d'adressage associé. Sur les architectures matérielles de type x86, les périphériques PCI sont identifiés de manière unique sur les bus d'E/S par le triplet (numéro de bus, numéro de périphérique, numéro de fonction). Ce triplet constitue l'identifiant BDF (*Bus/Device/Function*) du périphérique :

- **numéro de bus**. Il correspond au numéro de bus affecté par le BIOS au bus PCI ou PCI Express auquel le périphérique est physiquement connecté.

- **numéro de périphérique**. Pour un numéro de bus donné, il indique le connecteur d'extension (ou *slot*) auquel le périphérique est relié sur ce bus.
- **numéro de fonction**. Il identifie un sous-système du périphérique. Par exemple, sur une carte USB à plusieurs ports, ce numéro distingue les différents ports de la carte.

Les accès à la mémoire effectués par les périphériques que la DRHU intercepte contiennent l'identifiant BDF du périphérique à l'origine du transfert de données. La DRHU s'appuie sur cette métadonnée pour identifier le périphérique qui effectue l'accès. Cet identifiant est renseigné dans le champ `requester-id` de l'entête de la trame de la couche `transaction` du protocole PCI Express (figure 2). Dans les spécifications de la technologie Intel VT-d, ce champ est également nommé `source-id` (figure 3). Dans la suite de cet article, nous suivons la dénomination proposée par Intel.

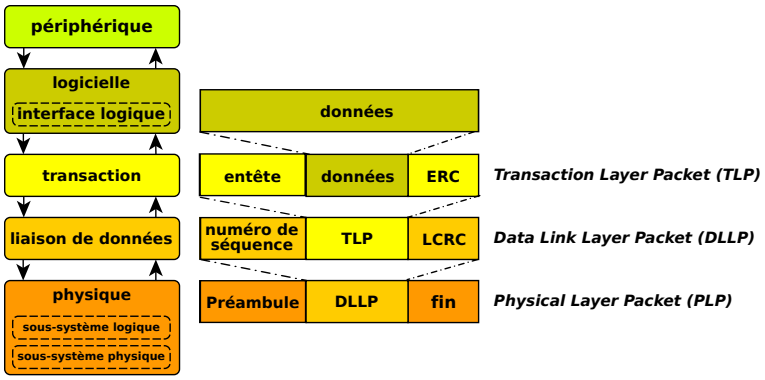

Figure 2. Aperçu des couches du standard PCI Express

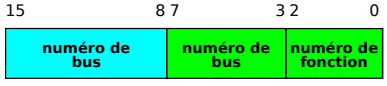

Figure 3. Format du champ `source-id`

La figure 4 montre le calcul effectué par la DRHU suite à l'identification du périphérique d'E/S. Les différentes parties qui composent le `source-id` indexent des entrées dans plusieurs niveaux de tables. Le numéro de bus sélectionne une entrée de la `root-entry table`. Cette entrée contient l'adresse de la `context-entry table` qui forme le second niveau d'indirection. Ensuite, le numéro de périphérique combiné au numéro de fonction indexent une entrée dans ce second niveau d'indirection, lequel indique le domaine de protection associé à ce périphérique. Un domaine de protection définit l'espace d'adressage DMA virtuel associé aux périphériques appartenant à ce domaine. Un ensemble de tables de pages établit la correspondance entre des plages d'adresses DVA (adresses virtuelles DMA) et des régions de la mémoire physique auxquelles ces périphériques sont autorisés à accéder. Signalons que le noyau de système d'exploitation indique à une DRHU l'adresse physique de la `root-entry table` par son registre de configuration `root-entry table address`.

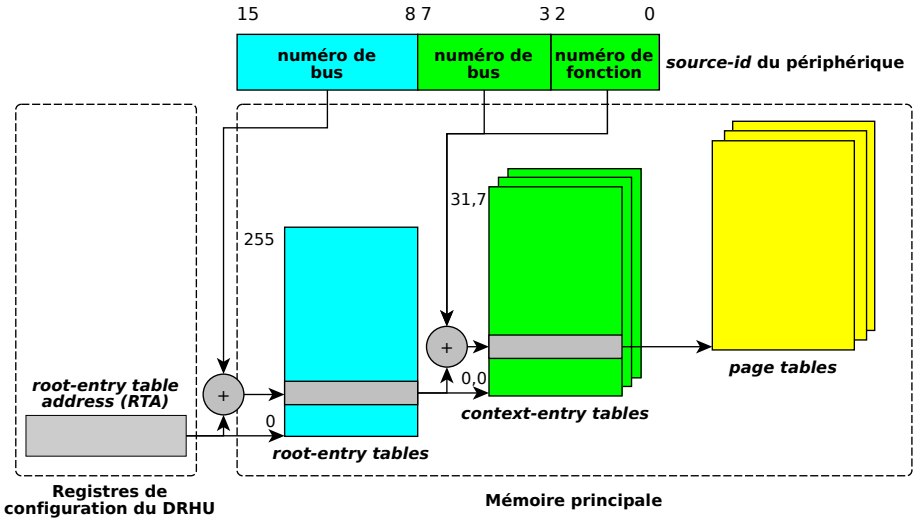

Figure 4. L'association d'un périphérique à un domaine de protection

Arrivé à ce stade, l'identité du périphérique effectuant l'accès à la mémoire et le domaine de protection auquel il est associé sont

connus. La DRHU entre dans la phase d'autorisation durant laquelle elle va convertir l'adresse DVA à laquelle accède le périphérique en une adresse physique correspondante et vérifier par la même occasion que l'accès effectué est autorisé.

Traduction de l'adresse DVA en adresse physique et contrôle d'accès à la mémoire. Tout comme une MMU traditionnelle, la correspondance entre les adresses DVA et les adresses physique HPA est déterminée à partir de tables de pages. Celles-ci sont allouées, initialisées et maintenues par le noyau de système d'exploitation.

La figure 5 présente la logique de parcours des tables de pages par la DRHU. Une adresse DVA est composée de différents blocs. Ces blocs indexent des entrées (ou PTE pour *Page Table Entry*) dans des tables de pages. L'entrée sélectionnée pour un niveau donné pointe sur la table de pages du niveau suivant. Cette logique d'indirection continue jusqu'à atteindre le dernier niveau des tables de pages. À partir de là, la valeur du PTE indexé constitue l'adresse de la page physique alignée sur 4 ko à laquelle accède le périphérique. L'adresse physique complète HPA est déduite en additionnant l'adresse de la page physique et le déplacement dans le dernier champ de l'adresse DVA.

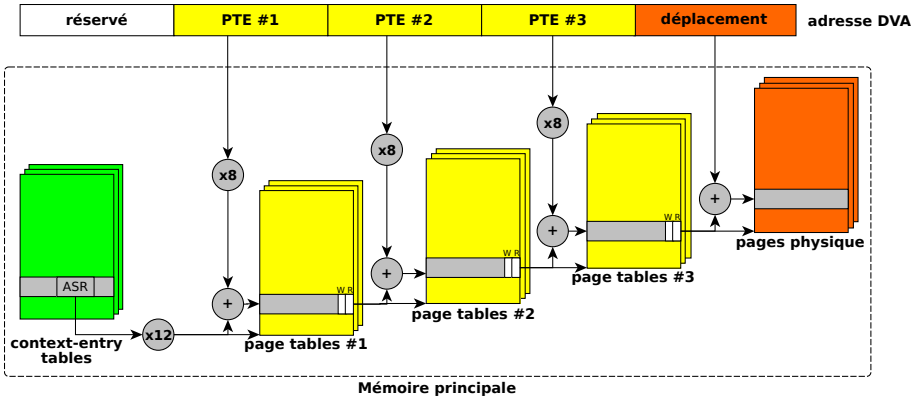

Figure 5. La traduction d'adresse DVA

La DRHU évalue également les permissions d'accès du périphérique à l'adresse mémoire demandée lors du parcours des tables de pages. Toutes les entrées des tables de pages contiennent des attributs de page qui définissent les droits d'accès du périphérique. À chaque niveau d'indirection, la DRHU s'appuie sur ces attributs de page pour évaluer si l'accès est autorisé. Par exemple, si le bit `Read (R)` est positionné, alors la page physique pointée par l'entrée est accessible en lecture pour le périphérique. De même, une page physique accessible en écriture est indiquée par le bit `Write (W)`.

Lorsque la traduction d'adresse aboutit et que le périphérique dispose de droits suffisants pour effectuer l'accès, l'adresse DVA est remplacée par l'adresse physique HPA correspondante puis l'accès mémoire est relayé au contrôleur mémoire. Dans le cas contraire, l'accès effectué par le périphérique est bloqué par la DRHU. Celle-ci avertit le périphérique (par une transaction PCI-Express `Unsupported Request`) que la requête n'a pas abouti, et déclenche une interruption matérielle pour alerter le système d'exploitation.

La section qui suit présente la mise en œuvre dans le noyau Linux de la technologie Intel VT-d. Nous exposons également les résultats que nous avons obtenus après l'avoir confrontée à une attaque DMA.

3 Intel VT-d en action sous Linux 2.6

Le support de la technologie Intel VT-d dans le noyau Linux a été ajouté depuis la version 2.6.24. Il est activé par les options de compilation `CONFIG_DMAR` et `CONFIG_DMAR_ON`.

La technologie Intel VT-d sous Linux est employée afin d'isoler du reste du noyau les régions de la mémoire principale associées aux différents périphériques. Les DRHU associent les périphériques d'E/S qu'ils identifient à des domaines de protection différents et en conséquence, à des régions distinctes de la mémoire physique. L'accès des périphériques se trouve ainsi restreint à ces régions de la mémoire.

Nous souhaitons vérifier, dans cette section, que cette extension matérielle empêche effectivement les périphériques d'E/S d'accéder à d'autres régions mémoire — telles que le segment de code du noyau — qui n'appartiennent pas au domaine de protection défini pour ces périphériques. Nous procédons pour cela par comparaison. Dans

un premier temps, nous allons mettre en place une attaque sur un système ne supportant pas la technologie Intel VT-d [2]. Notre attaque consiste à utiliser l'accès direct à la mémoire dans les périphériques d'E/S pour injecter du code malveillant dans la mémoire du noyau du système d'exploitation. Par la suite, nous rejouons l'attaque sur un système disposant d'une IOMMU. Nous nous assurons que l'injection de code est bien bloquée par Intel VT-d.

3.1 Escalade de privilèges par FireWire sur un système n'utilisant pas l'extension matérielle Intel VT-d

Notre machine d'expérimentation fonctionne sous le système d'exploitation Linux et utilise un noyau 2.6 [3]. Elle possède la technologie Intel vPro [4] [13]. Dans une première étape, nous supposons que la détection par le BIOS de l'extension matérielle Intel VT-d est désactivée. Par conséquent, du point de vue du système d'exploitation, la machine ne supporte pas l'extension matérielle. Enfin, elle dispose d'un périphérique d'E/S capable d'effectuer un accès direct à la mémoire principale — comme les cartes FireWire — que nous utilisons pour effectuer notre attaque DMA.

L'attaque que nous mettons en place vise à altérer le code du noyau du système d'exploitation. Elle s'inspire des techniques développées par Silvio Cesare [6], à la différence près que nous corrompons le noyau depuis l'espace d'adressage physique et non depuis l'espace d'adressage virtuel. Notre démarche s'apparente à celle publiée dans [15]. En nous servant de la capacité d'un périphérique à effectuer un accès direct à la mémoire, nous modifions l'appel système `sys_setuid()` dans le but d'obtenir les privilèges du super-utilisateur. Cet appel système associe l'identité d'un utilisateur au processus qui l'appelle.

Pour cela, nous remplaçons dans le code de l'appel système l'instruction `new->euid = uid` qui affecte la nouvelle identité du pro-

2. C'est-à-dire un système dont le MCH ne contient pas la technologie Intel VT-d, ou dont cette extension matérielle n'est pas activée par le BIOS au démarrage.

3. Cet article s'intéresse aux versions du noyau postérieures à 2.6.29.

4. Intel vPro désigne un ensemble d'extensions matérielles qui facilitent la virtualisation des processeurs (Intel VT-x) et des périphériques d'E/S (Intel VT-d), qui permettent l'administration à distance des machines (Intel AMT) et qui renforcent la sécurité d'un système (Intel TxT).

cessus par l'instruction `new->euid = 0`. Nous rappelons que l'identifiant 0 correspond à celui du super-utilisateur. Une fois la charge malveillante injectée, tout processus effectuant cet appel système se verra attribuer les privilèges du super-utilisateur.

L'exemple ci-dessous présente le comportement du système avant injection du code malveillant en mémoire et après son injection. Comme prévu, `euid` associé au processus après injection du code malveillant correspond à celui du super-utilisateur.

```
# Avant injection de code                # Apres injection de code
login: user                              login: user
password:                                password:
$> id                                    bash-3.2$ id
uid=1002(user) gid=1002(user)            uid=1002(user) gid=1002(
    user)
groups=1002(user)                        euid=0(root) groups=1002(
    user)
$> whoami                                bash-3.2$ whoami
user                                     root
```

3.2 Comportement d'un système utilisant Intel VT-d vis-à-vis d'attaques DMA

Dans une deuxième étape, nous réactivons la détection de l'extension matérielle Intel VT-d par le BIOS. Ainsi, le noyau du système va pouvoir contrôler les accès des périphériques à la mémoire principale et restreindre leurs accès à la mémoire physique réservée par le noyau pour effectuer du DMA.

À des fins de comparaison, nous rejouons notre attaque sur la machine. Nous donnons ci-dessous le comportement du système avant l'injection de code et après l'injection de code. L'injection de code à visiblement échoué car nous n'obtenons pas les privilèges du super-utilisateur.

```
# Avant injection de code                # Apres injection de code
login: user                              login: user
password:                                password:
$> id                                    $> id
uid=1002(user) gid=1002(user)            uid=1002(user) gid=1002(
    user)
groups=1002(user)                        groups=1002(user)
$> whoami                                $> whoami
user                                     user
```

La trace ci-dessous confirme que l'accès mémoire effectué par le périphérique FireWire au moment de l'injection de code a été bloqué par Intel VT-d.

```
$# lspci | grep 1394
00:1e.0 FireWire (IEEE 1394): Ricoh Co Ltd R5C832 IEEE 1394
    Controller
$# dmesg | grep DMAR
[90.818494] DMAR:[DMA Read] Request device [00:1e.0] fault addr
     0x1c000000
[90.818496] DMAR:[fault reason 06] PTE Read access is not set
```

Nous pouvions nous attendre à ce que notre attaque échoue en présence d'Intel VT-d. Il est configuré par le noyau Linux pour autoriser uniquement les périphériques à effectuer un accès à la zone DMA du segment de données du noyau. Or, notre attaque vise à modifier le code du noyau chargé en mémoire principale. Comme cet accès est considéré illégitime par les DRHU, ces derniers le bloquent et remontent l'accès frauduleux au noyau du système d'exploitation.

L'extension matérielle Intel VT-d est donc une technologie efficace pour restreindre l'accès des périphériques à des régions spécifiques de la mémoire physique. Dans la section suivante, nous présentons quelques vecteurs d'attaque sur cette technologie que nous avons identifiés au cours de nos travaux.

4 Identification d'une limite matérielle dans Intel VT-d

Suite à notre analyse d'Intel VT-d, nous avons identifié plusieurs vecteurs d'attaque sur cette technologie. Nous les présentons dans cette section. Puis, nous détaillons une limite matérielle pour laquelle une preuve de concept est fournie en section 5 et nous évaluons les risques liés à son exploitation.

4.1 Vecteurs d'attaque sur Intel VT-d

Nous avons classé les vecteurs d'attaque sur Intel VT-d en deux catégories : ceux qui concernent la modification des structures de données stockées en mémoire principale sur lesquelles repose le contrôle d'accès d'Intel VT-d ; puis ceux qui touchent des informations utilisées par le contrôle d'accès — le `source-id` par exemple — provenant d'autres périphériques.

Modification de structures de données stockées en mémoire. La plupart des composants de l'architecture x86 (GDT, IDT, etc.) reposent sur des structures de données stockées dans la mémoire principale. Ceci est également vrai pour l'extension Intel VT-d, laquelle s'aide de tables à plusieurs niveaux d'indirection pour déterminer la politique de contrôle d'accès à appliquer à un périphérique (section 2). La protection de ces parties critiques contre d'éventuelles actions malveillantes est à la charge du noyau du système d'exploitation. Or, les systèmes d'exploitation COTS sont rarement exempts de failles de sécurité. Lorsque ces failles sont exploitables, l'intégrité du noyau peut être compromise, tout comme les structures qu'il protège. Malheureusement, le noyau d'un système d'exploitation constitue aujourd'hui la cible privilégiée des malveillances. La modification de ces structures peut permettre à un attaquant, par exemple, de contourner certaines des protections du système. Pour protéger efficacement ces structures, il est nécessaire de s'appuyer sur des moyens de protection s'exécutant à un niveau plus privilégié que le noyau lui-même. É. Lacombe *et al.* [14] proposent une approche basée sur la préservation de contraintes s'appuyant sur les extensions de virtualisation matérielles pour empêcher toute atteinte à l'intégrité du noyau en mémoire, mais également de certains éléments de son support de contrôle (registres de la CPU, *chipset*, etc.).

Exploitation des méta-données fournies par les périphériques. Le second vecteur d'attaque concerne les méta-données fournies par les périphériques lors de leurs accès.

Les DRHU identifient le périphérique qui effectue un accès grâce au champ `source-id` qu'il fournit lors de son accès. De cette identification va découler la politique de sécurité à appliquer à celui-ci. Puisqu'aucun contrôle d'intégrité n'est effectué sur ce champ, on se doute qu'il est envisageable par exemple d'octroyer à un périphérique les mêmes droits d'accès à la mémoire qu'un autre périphérique en usurpant son identité. Le `source-id` dans l'accès effectué par le périphérique malveillant correspond alors à l'identifiant BDF d'un autre périphérique. Il est donc nécessaire pour les DRHU de s'assurer de l'identité du périphérique effectuant l'accès. Cela peut s'effectuer par exemple en signant les messages échangés entre les périphériques et la mémoire principale. Cela dit, pour des raisons de performance

et d'architecture cela n'est pas envisageable pour le moment. En effet, l'ajout d'un mécanisme de signature introduirait probablement un surcoût non-négligeable. Par ailleurs, le fait que l'architecture matérielle actuelle emploie différents standards de bus compliquerait la mise en place d'une telle solution. Comme chaque pont qui connecte un périphérique à la mémoire principale traduit l'accès d'un standard vers un autre standard, la signature d'un accès ne peut s'effectuer que de point-à-point et non de bout-en-bout.

Au cours de nos travaux, nous avons constaté que l'association d'un périphérique à son `source-id` n'est pas forcément bijective. En effet, plusieurs périphériques peuvent partager un unique `source-id`. Intel VT-d est alors incapable de distinguer ces périphériques et les fait accéder aux mêmes régions de la mémoire. Il est alors difficile de se fier à cette méta-donnée pour contrôler de manière stricte l'accès de chaque périphérique à la mémoire. Cette limite résulte probablement d'un choix de conception. Mais, dans la mesure où celle-ci peut être exploitée pour mener des attaques (section 5), nous la considérons dans la suite de l'article comme une vulnérabilité de la technologie.

Dans les parties qui suivent, nous focalisons notre attention sur cette vulnérabilité et nous voyons en quoi elle peut être problématique pour la sécurité d'un système.

4.2 Partage du `source-id` par les périphériques PCI connectés au même bus de données

Le standard PCI (pour *Peripheral Component Interconnect*) [20,19] a longtemps été utilisé comme norme de bus système dans un ordinateur. Cependant, avec l'augmentation croissante de la fréquence des processeurs au fil des années (66 MHz en 1993 à plus de 3 GHz en 2003), la bande passante du PCI est rapidement devenue un goulot d'étranglement dans les ordinateurs. Pour pallier ce problème, Intel a alors encouragé au cours de l'été 2004 l'utilisation du PCI Express [5] [21] dans les bus système. Afin d'assurer la retro-compatibilité des périphériques PCI sur la nouvelle architecture, des ponts PCI-

5. Le PCI Express a été conçu pour supplanter le standard PCI. Il correspond à l'ancien standard 3GIO, 3rd Generation Input/Output. Les acronymes PCIe ou PCI-E désignent également le standard PCI Express.

PCI Express (figure 6) ont été rajoutés au niveau du *chipset*. Ceux-ci interconnectent les bus PCI aux bus PCI Express et se chargent de faire la traduction de toutes les requêtes d'un standard à l'autre.

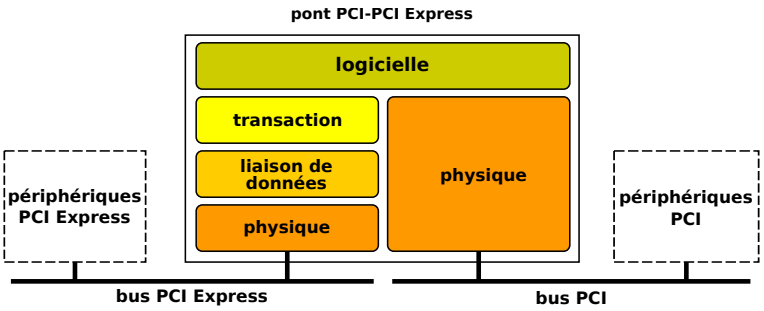

Figure 6. Pont PCI-PCI Express

Dans la pratique, le fonctionnement des ponts PCI-PCI Express se rapproche plus de celui d'un mandataire (*proxy*) que d'un pont. Pour nous en convaincre, analysons comment s'effectue la traduction d'un accès DMA depuis un périphérique PCI. Les accès DMA depuis des périphériques PCI, contrairement aux périphériques PCI Express, ne contiennent pas de champ `source-id`. Pour combler ce manque, le pont PCI-PCI Express prend possession de l'accès DMA lors de la traduction de celui-ci vers le PCI Express. Celui-ci utilise son identifiant BDF comme `source-id` pour l'accès traduit puis le relaie sur le bus PCI Express.

Du fait de ce mécanisme, tous les accès effectués par les périphériques PCI appartenant à un même bus vont être vus sur les bus PCI Express comme effectués par le pont PCI-PCI Express. Puisque les accès de ces périphériques possèdent le même `source-id`, les DRHU les associent au même domaine de protection et donc aux mêmes régions de la mémoire physique. On comprend dès lors que des problèmes de sécurité liés au partage de ressources (confidentialité, intégrité, disponibilité) peuvent s'en suivre. Un périphérique malveillant peut tirer partie de cette situation pour accéder et/ou altérer les régions de la mémoire principale utilisées par un autre périphérique. Une telle attaque est présentée en section 5.

Il est important de noter que beaucoup de machines de bureau disposent encore aujourd'hui d'au moins deux ports d'extension PCI [6]. Ces deux ports sont généralement connectés au même bus. Dans la mesure où des périphériques sont connectés à ces ports, la vulnérabilité que nous décrivons dans cet article peut être exploitée sur cette population de machines. Notons également que l'implémentation de l'IOMMU par AMD s'appuie également sur le `source-id` pour discriminer des périphériques. Ainsi, les conclusions que nous exprimons pour Intel VT-d sont également généralisables à cette technologie.

Une solution à long terme à cette vulnérabilité serait de passer de l'architecture matérielle hybride actuelle vers une architecture uniforme n'utilisant plus que du PCI Express. Cela conduirait à la suppression des ponts PCI-PCI Express, et par conséquent à la suppression de la vulnérabilité. En attendant, nous formulons quelques recommandations qui permettent de résoudre les problèmes de sécurité qu'elle implique. Nous les détaillons en section 6.

5 Preuve de concept

Cette section présente une preuve de concept pour la vulnérabilité détaillée en section 4. Nous commençons tout d'abord par présenter le scénario d'attaque ainsi que la plate-forme expérimentale que nous avons considéré. Nous donnons ensuite quelques précisions techniques qui permettent de mieux appréhender l'attaque que nous avons mise en place. Finalement, nous concluons sur les résultats que nous obtenons.

5.1 Description de notre plate-forme expérimentale

Considérons le scénario suivant pour notre preuve de concept. *Alice*, *Bob* et *Eve* travaillent pour une grande entreprise. *Alice* et *Bob* échangent des informations confidentielles à travers le réseau local. *Ève* souhaite accéder à ces informations auxquelles normalement elle n'a pas accès. Elle décide alors d'écouter leurs échanges, plus particulièrement les données que *Bob* envoie à *Alice*.

6. Constat effectué sur les cartes-mère Intel dont les spécifications techniques sont disponibles à `http://ark.intel.com/ProductCollection.aspx?familyId=1125`

Il n'est pas rare dans les grandes entreprises que le parc informatique soit composé de machines identiques (c'est-à-dire avec la même configuration matérielle, avec un système d'exploitation identique, etc.). Nous admettons ainsi qu'*Alice*, *Bob* et *Ève* disposent de machines de bureau identiques en tout point et exécutant un noyau Linux. La figure 7 présente la configuration de ces machines. Elles possèdent l'extension matérielle Intel VT-d dont le support a été activé à la fois dans le noyau et dans le BIOS. Ces machines possèdent également une carte Ethernet et une carte FireWire connectées au même bus PCI.

Ève découvre que sa machine souffre de la vulnérabilité matérielle présentée en section 4. Comme toutes les machines sont identiques, elle déduit que les autres machines de l'entreprise, en particulier celle de *Bob*, souffrent également de cette vulnérabilité. Elle y trouve là un moyen d'espionner *Bob*. Elle met en place en attaque DMA consistant à injecter des trames Ethernet malveillantes dans les tampons mémoires associés à la carte Ethernet de *Bob*. Dans notre cas d'étude, ces trames correspondent à des requêtes ARP. Cette attaque DMA est menée depuis un périphérique FireWire, par exemple un iPod modifié [8,9] par ses soins qu'elle prête à *Bob*.

Nous donnons dans la suite quelques précisions techniques sur le fonctionnement d'une carte Ethernet, nécessaires à la compréhension de notre attaque.

5.2 Injection des données dans une carte Ethernet

Cette partie s'intéresse au fonctionnement de la carte Ethernet — une VIA Rhine VT6102 [23] — installée dans la machine de *Bob*. Nous étudions uniquement la réception de trames.

Nous détaillons pour commencer les mécanismes et les structures de données mises en jeu lors de la réception de trames Ethernet. À partir de là, nous mettons en évidence les points d'injection. Enfin, nous détaillons le mode opératoire suivi.

Le tampon de réception dans la carte Ethernet VIA Rhine VT6102. La carte VIA Rhine VT6102 utilise un tampon circulaire pour stocker temporairement les données reçues depuis le réseau. Une mémoire tampon est utilisée pour chaque trame reçue. Cette

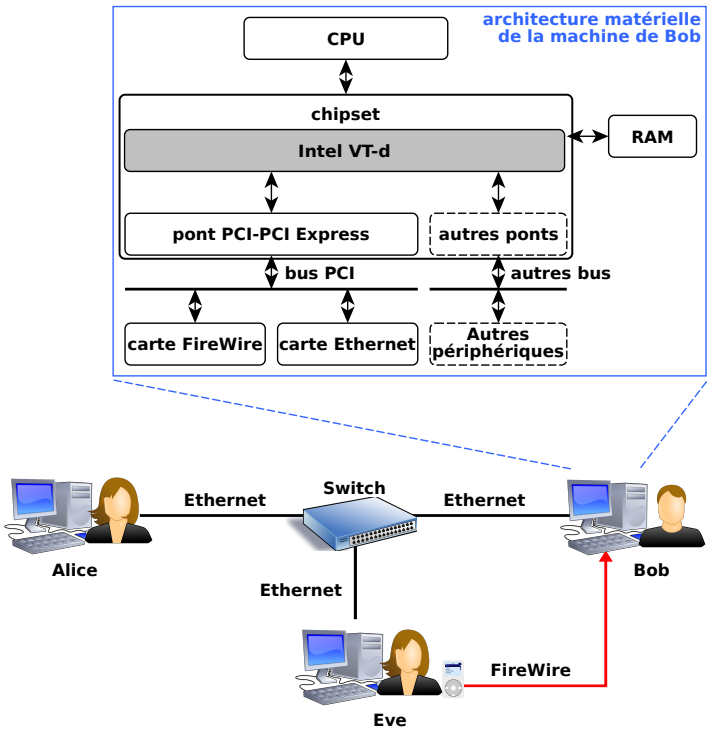

FIGURE 7. Plate-forme expérimentale considérée

mémoire tampon est libérée dès que le système d'exploitation l'a traitée. En mémoire principale, le tampon de réception est représentée par une liste circulaire de descripteurs de trame (figure 8). Un descripteur de trame est défini comme une structure de données de 16 octets dont le rôle est de décrire une mémoire tampon où va être stockée temporairement une trame Ethernet. Elle se compose de quatre champs. Le champ `adresse` indique où se situe, dans l'espace d'adressage physique, la mémoire tampon décrite par ce descripteur. Le champ `statut` décrit l'état de ce tampon. Il informe par exemple le système d'exploitation si la mémoire tampon contient une trame ou non, si celle-ci est valide ou si elle contient des erreurs, etc. Le champ `longueur` donne la taille des données stockées. Finalement, le champ `descripteur suivant` chaîne le descripteur de

trames courant à son suivant. Notons que l'allocation de la liste de descripteurs de trames ainsi que des mémoires tampons associées est effectuée par le noyau du système d'exploitation.

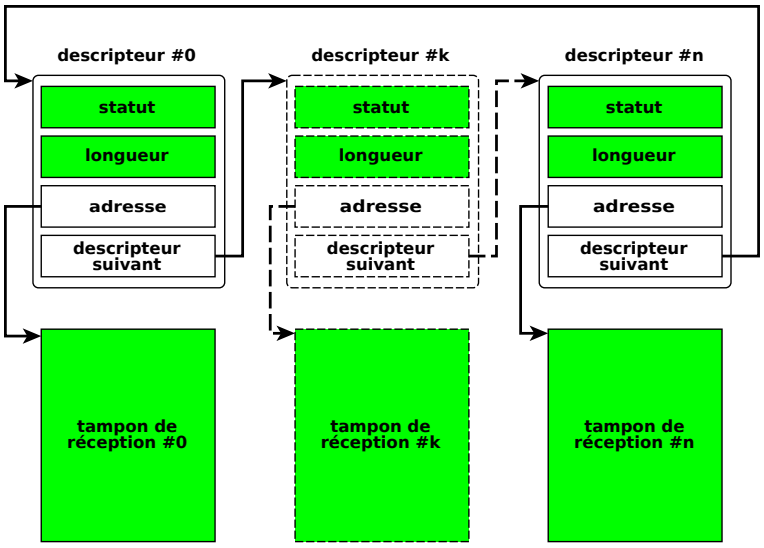

Figure 8. Le tampon de réception dans la carte Ethernet VIA Rhine VT6102

Lorsqu'une trame est reçue depuis le réseau, la carte Ethernet la stocke temporairement dans une des mémoires tampons disponibles. Le transfert de données depuis la carte réseau vers la mémoire s'effectue par un accès DMA. La carte Ethernet accède également aux descripteurs de trames afin de décrire de manière cohérente la mémoire tampon où est copiée la trame reçue. Afin que le périphérique puisse accéder à ces structures de données par DMA, le système d'exploitation les alloue dans la partie de la mémoire réservée aux accès DMA. Une fois la trame stockée dans le tampon, la carte réseau effectue une interruption matérielle qui va activer le gestionnaire d'interruption associé. Après avoir effectué certaines opérations, ce gestionnaire va consommer les trames Ethernet reçues qu'il n'a pas encore traitées. Nous présentons ci-après le mode opératoire de notre attaque.

Injecter des trames Ethernet par DMA. Notre idée, dans cette attaque, est de nous aider de la carte FireWire présente sur la machine de *Bob* pour injecter des trames Ethernet malveillantes en mémoire principale, directement dans le tampon de réception de sa carte réseau. Nous allons donc recopier le fonctionnement de la carte réseau lors de la réception de trames Ethernet.

Dans cette optique, l'attaquant commence par (1) injecter des données dans la région de la mémoire principale utilisée par la carte Ethernet. (2) Il modifie ensuite les descripteurs de trames associés aux mémoires tampons altérées afin qu'ils décrivent des données cohérentes en mémoire. (3) Finalement, il attend que le système d'exploitation de *Bob* traite les trames injectées. Typiquement, ces trames sont traitées à la réception d'une interruption matérielle issue de la carte réseau. Une interruption est générée par exemple après réception d'une nouvelle trame Ethernet.

Le mode opératoire que nous décrivons ici est appliqué pour la corruption du cache ARP de la machine appartenant à *Bob*. Nous présentons ci-après les résultats obtenus.

5.3 Application à la corruption de cache ARP

Dans un réseau Ethernet, chaque station dispose d'un identifiant physique unique correspondant à l'adresse MAC (*Media Access Control*) de sa carte réseau. À cet adressage physique s'ajoute un adressage logique : chaque carte réseau dispose d'une adresse IP (*Internet Protocol*) sur laquelle repose toutes les communications. Le protocole ARP (Adress Resolution Protocol) [22] est un protocole de type requête-réponse qui associe dynamiquement une adresse MAC à une adresse IP.

À la réception d'une requête ARP (`ARP-REQUEST`), une station met à jour automatiquement son cache ARP. Il est alors possible de modifier des entrées dans la table ARP d'une machine simplement en lui envoyant des requêtes `ARP-REQUEST`. Précisons qu'il est également possible d'obtenir les mêmes résultats en envoyant des réponses `ARP-REPLY`, mais cette méthode est moins efficace[7]. Dans le cadre

7. Un IDS pourrait, par exemple, facilement détecter qu'une ARP-REPLY ne correspond à aucune requête ARP-REQUEST.

d'attaques de type *ARP-poisoning*, ces réponses font évidemment correspondre une adresse MAC erronée à une adresse IP donnée.

Rappelons qu'*Ève* souhaite prétendre être *Alice* aux yeux de *Bob* afin d'écouter leur conversation. Pour cela, elle modifie l'entrée, dans le cache ARP de la machine de *Bob*, qui fait correspondre l'adresse IP d'*Alice* à son adresse MAC. La modification de cette entrée repose sur l'exploitation de la vulnérabilité matérielle découverte plus tôt. Afin d'être sûre de son coup, *Ève* met en place tout d'abord l'attaque sur sa propre machine, puis la rejoue sur la machine de *Bob*. Comme les machines sont identiques, elle est sûre que le rejeu de l'attaque sur la machine de *Bob* donnera les mêmes résultats.

À titre indicatif, nous donnons ci-dessous le contenu initial du cache ARP de la machine de *Bob* :

```
$> hostname
Bob
$> arp
Address    HWtype   HWaddress           Flags  Mask   Iface
Alice      ether    00:50:56:8a:3b:6b   C              eth0
Eve        ether    00:90:27:6a:58:74   C              eth0
```

Nous donnons ci-dessous le mode opératoire suivi par *Ève* lors de l'attaque :

1. *Ève* commence par forger les trames Ethernet qu'elle va injecter dans la machine de *Bob*. Elle peut utiliser par exemple SCAPY [3]. Elle forge des réponses `ARP-REQUEST` qui vont associer l'adresse MAC de sa carte réseau à l'adresse IP associée à la machine d'*Alice*.

```
$> scapy
Welcome to Scapy (2.0.0.5 beta)
>>> bob = {'ip': '192.168.1.1', 'mac': '00:25:00:9f:3b:30'}
>>> alice = {'ip': '192.168.1.2', 'mac': '00:50:56:8a:3b:6b
    '}
>>> eve = {'ip': '192.168.1.3', 'mac': '00:90:27:6a:58:74'}
>>> # Construction de la trame
... arp_request_packet = ARP(op='who-has', hwsrc=eve['mac'
    ],
... psrc=alice['ip'], hwdst='00:00:00:00:00:00', pdst=bob['
    ip'])
>>> ethernet_frame = Ether(dst='ff:ff:ff:ff:ff:ff', src=eve
    ['mac'])
>>> raw_frame = ethernet_frame/arp_reply_packet;
>>> # Ecriture de la trame dans un fichier
... file_handle = open('arp_reply_raw_frame', 'w')
>>> file_handle.write(str(raw_frame))
>>> file_handle.close()
```

2. *Ève* programme ensuite son iPod pour injecter ces trames malveillantes dans les tampons associés à la carte réseau de *Bob* dès que l'iPod se trouve connecté à sa machine. Elle localise ces tampons via les descripteurs de trames. Ceux-ci sont alloués lors de l'initialisation de la carte réseau et leur localisation en mémoire est invariante dans le temps. Comme les machines d'*Ève* et *Bob* sont identiques, les descripteurs de trames sont localisés aux mêmes adresses physiques sur les deux machines.
3. *Ève* demande à *Bob* de lui copier des fichiers (documents, musique, etc.) sur son périphérique. Ne se doutant de rien, *Bob* connecte l'iPod malveillant à sa machine. L'iPod injecte alors les paquets forgés dans les mémoires tampons associées à la carte réseau. Pour éviter tout risque d'écrasement de ces trames malveillantes, *Ève* configure son iPod pour injecter une même trame dans plusieurs mémoires tampons. Il met également à jour les champs `statut` et `longueur` des descripteurs de trame afin qu'ils décrivent des données cohérentes en mémoire. Nous rappelons que l'injection des trames malveillantes s'effectue aux dépens d'Intel VT-d. Ce dernier est incapable de différencier les accès effectués par la carte FireWire et la carte réseau car ils possèdent le même `source-id`. Les DRHU voient les accès effectués par la carte FireWire, pilotée par l'iPod, comme étant effectués par la carte réseau.
4. Finalement, elle attend que le système d'exploitation de *Bob* traite les trames Ethernet injectées. Rappelons qu'*Ève* peut déclencher le traitement de celles-ci par l'envoi par le réseau d'un paquet valide à destination de la machine de *Bob*. À sa réception, la carte réseau de *Bob* va générer une interruption matérielle, ce qui provoquera le traitement des trames injectées.

Le contenu du cache ARP de *Bob* suite à l'attaque est donné ci-dessous. Nous remarquons que l'adresse MAC d'*Alice* a changé prouvant que l'injection de trames Ethernet dans les tampons associés à la carte réseau a fonctionné.

```
$> hostname
Bob
$> arp
Address    HWtype   HWaddress           Flags  Mask   Iface
Alice      ether    00:90:27:6a:58:74   C              eth0
Eve        ether    00:90:27:6a:58:74   C              eth0
```

Cette preuve de concept met en évidence deux problèmes. Tout d'abord, elle nous a prouvé l'existence d'une vulnérabilité dans la technologie Intel VT-d. Ensuite, nous avons démontré combien il est problématique de laisser des périphériques partager une même zone mémoire. La section qui suit présente quelques contre-mesures à l'exploitation de cette vulnérabilité.

6 Contre-mesures

En section 4, nous avons vu que les ponts PCI-PCI Express empêchent Intel VT-d de fournir une isolation stricte de la mémoire associée aux périphériques. Des problèmes peuvent découler de cette situation, et nous l'avons démontré en section 5 au travers d'une preuve de concept. Bien entendu, lorsqu'à long terme l'architecture matérielle convergera vers le PCI Express, les ponts PCI-PCI Express seront supprimés et la vulnérabilité que nous exploitons sera obsolète. En attendant, nous donnons quelques recommandations afin d'empêcher son exploitation.

La solution qui nous vient naturellement à l'esprit consiste à désactiver les canaux DMA à partir du noyau. Cela dit, une telle stratégie est délicate, voire impossible, à mettre en œuvre. En l'absence de DMA, le processeur interroge régulièrement les périphériques sur la disponibilité de données et effectue lui-même le transfert vers la mémoire. On se rend alors compte que cela est très pénalisant en cycles CPU. Autrement dit, il est inacceptable de désactiver le DMA pour certains périphériques (comme les cartes graphique par exemple). De plus, désactiver les canaux DMA requiert la modification des pilotes de périphérique afin qu'ils interrogent régulièrement les périphériques sur la disponibilité de données.

Une meilleure approche serait d'isoler chaque périphérique PCI sur un bus séparé. Ce faisant, la confusion entre les identités des périphériques disparaîtrait. La limite ne serait ainsi plus exploitable car l'isolation de la mémoire des périphériques par Intel VT-d est effectuée de manière stricte. Nous préconisons cette solution à court terme, en attendant la convergence vers une architecture « tout-PCI Express ».

7 Conclusion

Dans ce papier, nous avons analysé l'efficacité du service fourni par une IOMMU. Une IOMMU est un composant matériel permettant à un système d'exploitation de contrôler l'accès des périphériques à la mémoire principale. Notre étude s'est focalisée sur la technologie Intel VT-d qui en implémente une. Nous avons présenté son principe de fonctionnement ainsi que sa mise en œuvre sur le système d'exploitation Linux. Nous avons confronté ce composant matériel à plusieurs attaques DMA. L'une d'entre elles est détaillée dans cet article. Le fait qu'elles aient toutes échoué dès lors que l'IOMMU a été activée confirme que cette technologie est efficace pour parer les attaques DMA. Néanmoins, notre analyse a révélé quelques vulnérabilités dans cette technologie. Pour l'une de ces vulnérabilités, nous avons décrit une preuve de concept et formulé des recommandations pour empêcher son exploitation. Parmi celles-là, nous préconisons l'utilisation d'un seul périphérique PCI par bus PCI.

Notre analyse s'est focalisée sur les attaques perpétrées par les périphériques d'E/S qui visent à corrompre des données en mémoire principale. Des attaques pourraient également tenter d'exploiter la fonctionnalité qui permet aux périphériques d'interagir directement entre eux, par exemple par des transactions peer-to-peer, sans passer par la mémoire principale. Lorsque ces transactions transitent entre le *northbridge* et le *southbridge*, l'IOMMU est *a priori* capable de contrôler ces intéractions. Mais comme les périphériques sont majoritairement connectés au *southbridge*, les échanges entre ces périphériques s'effectuent de manière directe et l'IOMMU n'est donc d'aucun secours pour contrôler ces accès et empêcher des interactions malveillantes comme l'altération de l'état d'un périphérique par un autre, le vol d'information, etc. Il serait alors intéressant de voir comment bloquer de telles interactions. Serait-il faisable, par exemple, d'introduire dans l'architecture matérielle des composants qui pourraient bloquer ou non l'échange de données entre périphériques ?

8 Remerciements

Nous souhaitons remercier les différents relecteurs pour leurs conseils et leurs remarques constructives qui ont permis d'améliorer la qualité de cet article.

Références

1. Inc. Advanced Micro Devices. *AMD I/O Virtualization Technology (IOMMU) - Architectural specification*, February 2009. http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf.
2. Damien Aumaitre. Voyage au coeur de la mémoire. In *Actes du 6ème Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2008)*, pages 378–437. École Supérieure et d'Application des Transmissions, June 2008. http://actes.sstic.org/SSTIC08/Voyage_Coeur_Memoire/.
3. Philippe Biondi. Scapy. http://www.secdev.org/projects/scapy/.
4. Adam Boileau. Hit by a bus : Physical access attacks with FireWire. In *RUXCON 2006*, October 2006. http://www.ruxcon.org.au/files/2006/firewire_attacks.pdf.
5. Brian Carrier and Joe Grand. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation*, 1(1) :50–60, February 2004. http://www.digital-evidence.org/papers/tribble-preprint.pdf.
6. Silvio Cesare. Runtime kernel kmem patching, November 1998. http://vx.netlux.org/lib/vsc07.html.
7. Christophe Devine and Guillaume Vissian. Compromission physique par le bus PCI. In *Actes du 7ème Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2009)*, pages 169–193. École Supérieure et d'Application des Transmissions, June 2009. http://actes.sstic.org/SSTIC09/Compromission_physique_par_le_bus_PCI/.
8. Maximillian Dornseif. 0wned by an iPod - hacking by FireWire. In *PacSec/core04*, 11-12 November 2004. http://md.hudora.de/presentations/%23firewire-pacsec.
9. Maximillian Dornseif. FireWire - all your memory are belong to us. In *CanSecWest/core05*, 4–5 May 2005. http://md.hudora.de/presentations/%23firewire-cansecwest.
10. Loïc Duflot. *Contribution à la sécurité des systèmes d'exploitation et des microprocesseurs*. PhD thesis, Université de Paris XI, October 2007. http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lti/these-duflot.pdf.
11. Inc. Intel Corporation. *Universal Host Controller Interface (UHCI) Design Guide*, March 1996. http://download.intel.com/technology/usb/UHCI11D.pdf.
12. Inc. Intel Corporation. *Intel Virtualization Technology for Directed I/O - Architecture Specification*, September 2008. http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf.

13. David Kleidermacher. Methods and applications of system virtualization using Intel® Virtualization Technology (Intel® VT). *Intel Technology Journal*, 13(1) :74–83, March 2009. http://download.intel.com/technology/itj/2009/v13i1/pdf/ITJV13I1.Journal-Web.PDF.
14. Éric Lacombe, Vincent Nicomette, and Yves Deswarte. Enforcing kernel constraints by hardware-assisted virtualization. *Journal in Computer Virology*, pages 1–21, August 2009. http://www.springerlink.com/content/v0w56774150764vr/.
15. Anthony Lineberry. Malicious code injection via /dev/mem. In *Black Hat Europe 2009*, March 2009. http://www.dtors.org/papers/malicious-code-injection-via-dev-mem.pdf.
16. Antonio Martin. FireWire memory dump of a windows XP computer : a forensic approach. Technical report, friendsglobal.com, 2007. http://www.friendsglobal.com/papers/FireWire%20Memory%20Dump%20of%20Windows%20XP.pdf.
17. David Maynor. 0wn3d by everything else - USB/PCMCIA Issues. In *CanSecWest-/core05*, 4-5 May 2005. http://cansecwest.com/core05/DMA.ppt.
18. Microsoft, Compaq, and National Semiconductor. *Open Host Controller Interface Specification for USB*. Microsoft and Compaq and National Semiconductor, September 1999. ftp://ftp.compaq.com/pub/supportinformation/papers/hcir1_0a.pdf.
19. PCI Special Interest Group. *PCI-to-PCI Bridge Architecture Specification – Revision 1.1*. PCI Special Interest Group, 2575 N.E. Kathryn #17 - Hillsboro, Oregon 97124, December 1998. http://www.pcisig.com/specifications/conventional/pci_to_pci_bridge_architecture/.
20. PCI Special Interest Group. *PCI Local Bus Specification – revision 2.3*. PCI Special Interest Group, 5440 SW Westgate Drive – Suite 217 – Portland, Oregon 97221, March 2002. http://www.pcisig.com/specifications/conventional/conventional_pci_23/.
21. PCI Special Interest Group. *PCI Express^{TM} Base Specification – Revision 1.1*. PCI Special Interest Group, March 2005. http://www.pcisig.com/specifications/pciexpress/specifications/.
22. David C. Plummer. An Ethernet address resolution protocol, November 1982. http://tools.ietf.org/html/rfc826.
23. Inc. VIA Technologies. *VT6102 PCI Fast Ethernet controller with ACPI function - Datasheet*. VIA Technologies, Inc., 5020 Brandin Court Fremont, CA 94538 USA, 1 August 1999. http://people.freebsd.org/~wpaul/VIA/VT6102_021.PDF.