Attaques DMA *peer-to-peer* et contremesures

Fernand Lone Sang[1,2], Vincent Nicomette[1,2],
Yves Deswarte[1,2], and Loïc Duflot[3]

[1] CNRS ; LAAS ; 7 Avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
[2] Université de Toulouse ; UPS, INSA, INP, ISAE ; UT1, UTM, LAAS ;
F-31077 Toulouse Cedex 4, France
[3] ANSSI ; 51 Boulevard de la tour Maubourg, F-75700 Paris Cedex 07, France

Résumé Au cours des sept dernières années, les attaques perpétrées depuis des contrôleurs d'entrée-sortie (ex. cartes réseaux, contrôleurs FireWire) ont connu un intérêt grandissant. On ne dénombre plus les preuves de concept d'escalade de privilège abusant du mécanisme d'accès direct à la mémoire (*Direct Memory Access* ou DMA) permettant à un contrôleur de lire ou de modifier le contenu de la mémoire principale de la machine cible. Cette classe d'attaques reste pourtant encore méconnue à ce jour.

La présente étude se concentre principalement sur l'exploitation du mécanisme DMA dans l'architecture matérielle Intel PC. Nous explorons l'idée d'utiliser le DMA autrement que comme moyen de corruption de la mémoire principale. Nous nous intéressons particulièrement à l'usage du DMA comme vecteur d'attaque sur d'autres contrôleurs d'entrée-sortie. Nous détaillons dans quelles mesures ces attaques sont possibles sur les *chipsets* actuels et nous illustrons notre propos par la preuve de concept d'une attaque sur une carte graphique. Finalement, nous discutons des différentes mesures existantes permettant de limiter l'impact de telles attaques.

Mots-clés DMA, *peer-to-peer*, *chipset*, architecture matérielle

1 Introduction

Il devient aujourd'hui de plus en plus difficile de protéger efficacement les systèmes informatiques. Au fur et à mesure de l'accroissement de leur complexité, la surface d'attaque de ces systèmes s'est élargie et le nombre d'attaques réussies ne cesse de croître, en dépit des nombreux mécanismes de protection mis en place.

Les malveillances visant les systèmes informatiques s'appuient majoritairement sur des vecteurs d'attaque liés au logiciel s'exécutant sur le processeur principal. La plupart de ces malveillances exploitent en effet des fonctionnalités trop permissives du système (ex. chargeur de modules noyau, périphériques virtuels tels que `/dev/mem` ou `/dev/kmem`) ou des erreurs d'implémentations logicielles (ex. débordement de tampon, chaînes de format). Cette classe d'attaques est relativement bien connue à ce jour et de nombreux mécanismes permettent de s'en protéger. Depuis quelques années, nous voyons apparaître une

nouvelle classe d'attaques s'appuyant sur certaines fonctionnalités des contrôleurs d'entrée-sortie (ex. cartes réseaux, contrôleurs FireWire) telle que le mécanisme d'accès direct à la mémoire (*Direct Memory Access* ou DMA). C'est un mécanisme qui permet de transférer directement des données en mémoire principale grâce à un composant spécifique de chaque contrôleur d'entrée-sortie : le *DMA engine*. Le processeur se trouve alors déchargé des transferts d'entrée-sortie et n'intervient plus, dans le meilleur des cas, que pour initier et terminer ces transferts.

Les attaques DMA suscitent aujourd'hui un intérêt croissant comme en témoignent les nombreuses preuves de concept d'escalade de privilège existantes. Nous les classons selon deux catégories, suivant que l'accès à la mémoire est initié par le processeur ou par le contrôleur d'entrée-sortie [17,18].

La première catégorie d'attaques concerne les contrôleurs qui requièrent l'intervention du processeur pour mettre en place un accès à la mémoire. De telles attaques nécessitent ainsi l'exécution d'un code malveillant utilisant le système d'exploitation afin de mettre en place un transfert DMA. Une preuve de concept publiée dans [10] montre comment un attaquant programme des transferts DMA dans les contrôleurs USB de type UHCI [13] dans le but de modifier la valeur du `securelevel` dans un système OpenBSD [25].

La seconde catégorie d'attaques DMA concerne les contrôleurs qui initient eux-mêmes leurs accès DMA suite à des ordres reçus depuis l'extérieur, par exemple depuis des périphériques qui leur sont connectés. Les preuves de concept de telles attaques sont nombreuses et utilisent des contrôleurs d'entrée-sortie variés : contrôleurs PCI programmables [5,8], contrôleurs FireWire [9,23,4,3,21], contrôleurs USB On-The-Go [22], contrôleurs Ethernet [12], *etc.* Dans cet article, nous nous concentrons principalement sur cette seconde catégorie d'attaque.

Malgré les études nombreuses sur le sujet, nous nous rendons compte que les attaques DMA restent encore à ce jour méconnues et les mécanismes de protections associés, telles que les *Input/Output Memory Management Units* (I/O MMUs), sont encore peu usités et leurs limites sont encore mal connues [19]. Les attaques DMA sont le plus souvent associées à la corruption de la mémoire principale (ex. modification de l'image du noyau en mémoire). Arrigo Triulzi, au travers de ses différents travaux [30,31] sur des architectures matérielles plus anciennes [4], a montré que les attaques DMA peuvent aussi bien cibler la mémoire principale que les mémoires embarquées dans les contrôleurs d'entrée-sortie. Qu'en est-il de l'architecture matérielle Intel PC actuelle ? Si un attaquant réussit à implanter un *rootkit* dans un contrôleur d'entrée-sortie [30,31,7], que lui est-il possible de faire malgré les mécanismes de protection existants ? Peut-il lire ou modifier la mémoire d'autres contrôleurs ? Et si oui, comment peut-on se protéger contre de telles attaques ? Telles sont les questions auxquelles nous tentons de répondre.

4. Arrigo Triulzi s'est *a priori* focalisé sur l'ancienne architecture Intel PC, dans laquelle le *chipset* est basé sur un bus système PCI [27,26]. L'architecture Intel PC actuelle repose, quant à elle, sur l'utilisation de bus PCI Express [28].

Cet article est structuré comme suit : en section 2, nous commençons par rappeler les principes de l'architecture Intel PC, ainsi que les différents mécanismes qui permettent au processeur de dialoguer avec les contrôleurs d'entrée-sortie. Ces rappels techniques permettent au lecteur de mieux appréhender les attaques DMA *peer-to-peer* que nous décrivons dans la section 3. Elle présente également les résultats obtenus quant à l'identification des *chipsets* actuels qui permettent de mettre en œuvre ce type d'attaques. Dans la section 4, nous nous plaçons du point de vue d'un attaquant. Nous y décrivons la preuve de concept que avons implémentée, montrant qu'il est possible de manipuler la mémoire d'une carte graphique depuis un périphérique FireWire. Dans la section 5, nous analysons quelques contremesures existantes pour maîtriser les interactions entre les contrôleurs d'entrée-sortie. Finalement, la section 6 conclut notre article et propose quelques perspectives à nos travaux.

2 Quelques éléments d'architecture

L'architecture matérielle Intel PC est aujourd'hui l'architecture la plus répandue dans les systèmes informatiques. Au fil des années, cette architecture a beaucoup évolué. Dans cette section, nous la présentons succinctement telle qu'elle est à l'heure actuelle. Nous détaillons également les différents mécanismes qui permettent au processeur d'interagir avec les contrôleurs d'entrée-sortie. Le principe de fonctionnement de ces mécanismes nous permet d'introduire des concepts qui facilitent la compréhension des attaques DMA *peer-to-peer* dont il est question dans cet article.

2.1 Architecture matérielle typique Intel PC

La figure 1 présente une vue simplifiée d'une architecture matérielle Intel PC classique. Elle se compose de divers éléments. Les deux éléments principaux sont le processeur (*Central Processing Unit* ou CPU) et le *chipset*. Le processeur est l'élément sur lequel les différents composants logiciels (ex. applications, système d'exploitation) sont exécutés. Quant au *chipset*, il se charge de connecter le processeur à d'autres composants, tels que les contrôleurs de mémoire, les contrôleurs de bus (ex. contrôleurs USB, contrôleurs de disque), les contrôleurs d'entrée-sortie (ex. contrôleur Ethernet, contrôleur FireWire). Comme cet article est dédié aux attaques perpétrées depuis les contrôleurs d'entrée-sortie, nous allons principalement étudier ici l'architecture des *chipsets*.

La majorité des *chipsets* Intel se caractérise par la combinaison de deux puces, le *northbridge* et le *southbridge*, interconnectées par un bus propriétaire appelé *Direct Media Interface* (DMI).

Le *northbridge*, également appelé *Memory Controller Hub* (MCH) ou « pont nord », représente l'élément en charge d'interconnecter le processeur, la mémoire principale et les différents contrôleurs d'entrée-sortie. Ces derniers peuvent être connectés au *northbridge*, soit directement, soit par l'intermédiaire d'un pont PCI Express, lorsque ceux-ci requièrent de larges bandes passantes (ex. les

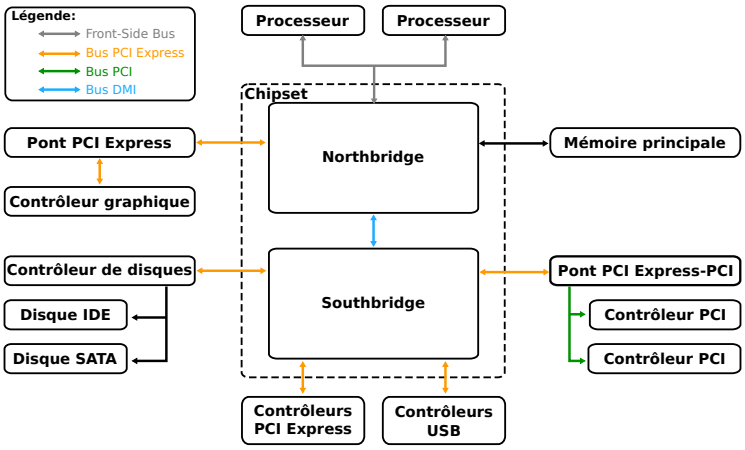

Figure 1. Vue simplifiée d'une architecture matérielle Intel PC typique

contrôleurs graphiques), soit au travers du *southbridge*, lorsque ceux-ci nécessitent moins de ressources.

Le *southbridge*, également nommé *I/O Controller Hub* (ICH) ou « pont sud », assure la connexion entre les contrôleurs d'entrée-sortie et le *northbridge*. Il fournit des interfaces qui lui permettent de communiquer sur différents types de bus tels que les bus USB, les bus PCI, les bus ATA/IDE, *etc.* La communication avec ces bus s'effectue à travers de ponts (ex. pont PCI Express-PCI) ou grâce à des contrôleurs de bus (ex. contrôleurs USB, contrôleurs ATA/IDE, contrôleurs SMBus). Il est important de noter qu'un même modèle de *southbridge* peut être utilisé dans des *chipsets* différents. Autrement dit, pour un modèle de *southbridge*, il est possible de trouver plusieurs modèles de *northbridge* compatibles.

Dans la sous-section suivante, nous présentons les mécanismes qui permettent au processeur de communiquer avec les différents contrôleurs d'entrée-sortie de la machine. Nous nous attardons plus particulièrement sur le mécanisme dit de projection mémoire ou *Memory Mapped I/O*.

2.2 Accès aux contrôleurs d'entrée-sortie

Les interactions entre le processeur et les contrôleurs s'effectuent au travers de registres. Ces registres peuvent être projetés dans divers espaces d'adressage. Sur les architectures matérielles Intel PC, nous distinguons trois espaces d'adressage distincts permettant de dialoguer avec les contrôleurs, chacun disposant de son propre mécanisme d'accès :

– **Espace *Programmed I/O* (PIO) ou *Port Mapped I/O* (PMIO).** Il s'agit d'un espace d'adressage sur 16 bits, logiquement indépendant de l'espace d'adressage mémoire principal. Sur les processeurs Intel x86, l'accès aux registres des contrôleurs projetés dans cet espace s'effectue par des instructions spécifiques : les instructions `in` en assembleur pour des opéra-

tions de lecture, et les instructions `out` en assembleur pour des opérations d'écriture.

- **Espace *Memory Mapped I/O* (MMIO).** Cet espace d'adressage fait partie de l'espace d'adressage principal. Le *chipset* projette les registres et la mémoire embarquée dans les contrôleurs dans l'espace d'adressage principal et rend transparent l'accès à ces registres et à cette mémoire depuis le processeur : il y accède comme il accèderait à la mémoire principale, c'est-à-dire avec des instructions `mov` en assembleur. C'est le *chipset* qui est ensuite en charge d'aiguiller les accès vers les contrôleurs concernés. Afin d'éviter tout conflit avec la mémoire principale, cet espace d'adressage est généralement localisé dans les adresses hautes de l'espace d'adressage principal.
- **Espace de configuration PCI/PCI Express.** Il s'agit d'un espace d'adressage distinct dans lequel sont projetés les registres de configuration d'un contrôleur PCI ou PCI Express. L'espace de configuration s'étend sur 256 octets pour un contrôleur PCI ou sur 4096 octets dans le cas du PCI Express. Les registres projetés dans cet espace permettent d'identifier un contrôleur et de configurer son interface PCI ou PCI Express. L'agencement de cet espace de configuration est différent selon que le contrôleur interrogé est un pont ou un contrôleur d'entrée-sortie. Il existe différentes méthodes d'accès à cet espace de configuration. Le lecteur est invité à consulter la norme PCI [27] et la norme PCI Express [28] pour plus de détails à ce sujet.

Le BIOS est chargé de configurer le *chipset* au démarrage de la machine afin de permettre au processeur de dialoguer avec les contrôleurs d'entrée-sortie. Pour cela, il parcourt l'arborescence des bus PCI Express et configure l'interface d'entrée-sortie (c'est-à-dire, PCI-Express) de chaque composant qu'il découvre. Lorsque le composant découvert est un contrôleur de bus ou un contrôleur d'entrée-sortie, il peut être nécessaire de projeter certains de ses registres en mémoire. Ces registres permettent, entre autres, de configurer les services fournis par le contrôleur découvert (ex. accélération matérielle pour une carte graphique). Le BIOS détermine si le contrôleur dispose de registres à projeter en mémoire en lisant plusieurs registres de l'espace de configuration PCI-Express, appelées *Base Address Registers* (BARs). Lorsque c'est le cas, le BIOS indique alors au contrôleur l'adresse à partir de laquelle ceux-ci vont être projetés, en ré-écrivant dans ces mêmes registres de configuration. En revanche, lorsque le composant découvert est un pont, le BIOS configure ce dernier pour qu'il aiguille correctement les accès aux différents espaces d'adressage. Cela se traduit en pratique par l'initialisation de différents registres de configuration (ex. *Memory Base-Limit, I/O Base-Limit*), lesquels indiquent au pont configuré la plage d'adresses que celui-ci doit transmettre aux bus fils qu'il contrôle.

Afin de mieux comprendre comment le *chipset* procède pour projeter les registres d'un contrôleur d'entrée-sortie dans l'espace d'adressage principal, nous détaillons les mécanismes qui permettent à un accès MMIO effectué depuis le processeur d'être acheminé vers le contrôleur d'entrée-sortie concerné. Pour cela,

nous nous appuyons sur la figure 2, laquelle représente un exemple d'accès MMIO initié par le processeur à destination du contrôleur Ethernet connecté au *south-bridge*. Afin d'être complet dans notre explication, nous avons donné en figure 2 une architecture plus détaillée d'un *chipset*. Nous y avons également représenté la configuration des divers composants telle qu'elle est effectuée par le BIOS lors du démarrage de la machine.

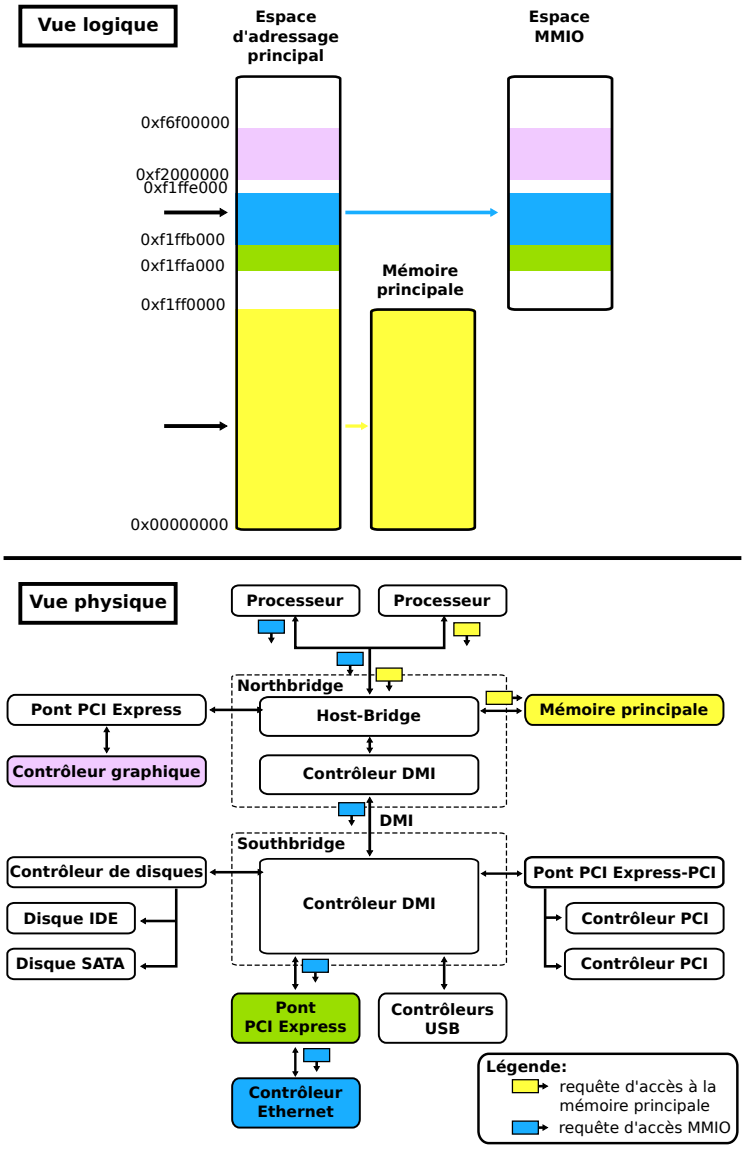

Figure 2. Exemples d'accès mémoire et d'accès MMIO depuis le processeur

Dans notre exemple, nous considérons que le système d'exploitation souhaite modifier l'état d'un registre du contrôleur Ethernet. Nous détaillons ci-dessous les différentes étapes qui permettent à la requête d'accès initiée par le processeur de parvenir au contrôleur Ethernet :

1. **Le processeur génère une requête d'accès MMIO.** Un logiciel s'exécutant sur le processeur (ex. pilote du contrôleur Ethernet considéré) effectue une écriture à l'adresse virtuelle `0xff80f1ffc000`. L'unité de gestion mémoire (*Memory Management Unit* ou MMU) placée dans le processeur traduit alors cette adresse virtuelle en une adresse physique (ex. `0xf1ffc000`), laquelle correspond à l'adresse d'un registre du contrôleur Ethernet projeté dans l'espace d'adressage principal. Une requête est générée sur le bus qui connecte le processeur au *northbridge*.
2. **Le *host-bridge* aiguille la requête vers le *southbridge*.** Au sein du *northbridge*, c'est le *host-bridge* qui prend en charge la requête d'accès effectuée par le processeur. En fonction de l'adresse physique contenue dans la requête, il détermine que celle-ci n'est ni destinée à la mémoire principale, ni à un contrôleur connecté au *northbridge*. Il la relaie alors au *southbridge* via le contrôleur DMI.
3. **Le contrôleur DMI transmet la requête au pont PCI Express.** À la réception de la requête d'accès venant du *northbridge*, le contrôleur DMI au sein du *southbridge* détermine le contrôleur vers lequel aiguiller la requête, toujours en se basant sur l'adresse physique contenue dans la requête. Elle est ainsi aiguillée vers le pont PCI Express qui connecte le contrôleur DMI au contrôleur Ethernet.
4. **Le pont PCI Express relaie la requête au contrôleur Ethernet.** Le pont PCI Express vérifie avant tout si la requête qu'il vient de recevoir lui est destinée. Il compare pour cela l'adresse physique contenue dans la requête (`0xf1ffc000`) aux adresses où sont projetées ses registres (`0xf1ffa000`-`0xf1ffafff`). Comme la requête ne lui est pas adressée, ce dernier ne traite pas la requête. Il vérifie ensuite que celle-ci est destinée à un de ses bus fils en consultant la plage d'adresses physiques qui leur est associée (`0xf1ffb000`-`0xf1ffdfff`). Auquel cas, il la transmet à son bus fils afin qu'un contrôleur placé en aval puisse la traiter. La figure 3 illustre ces actions.
5. **Le contrôleur Ethernet traite la requête.** La requête arrive finalement au contrôleur d'entrée-sortie. Celui-ci reconnaît que la requête lui est destinée car l'adresse physique contenue dans celle-ci appartient à la plage d'adresse physique `0xf1ffb000-0xf1ffdfff` (figure 4). Il traite la requête et met à jour en conséquence le contenu du registre projeté à l'adresse physique `0xf1ffc000` avec la nouvelle valeur fournie dans la requête.

Nous venons de détailler le cheminement d'un accès MMIO depuis le processeur vers un registre du contrôleur Ethernet. Puisque nous avons effectué une écriture, il n'est pas nécessaire que le contrôleur renvoie une réponse. Dans le cas d'un accès en lecture, lequel nécessite que le contrôleur cible renvoie le résultat de la requête, il est important de préciser que son transit dans les contrôleurs

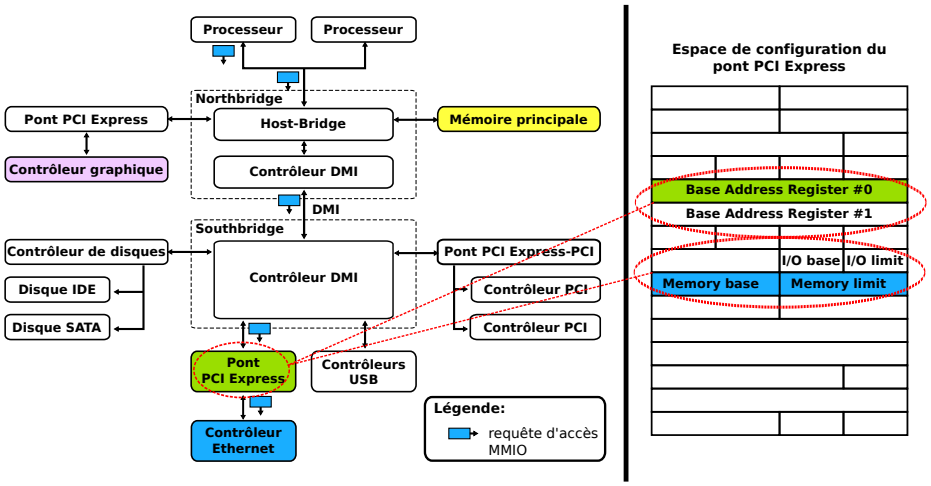

Figure 3. Actions d'un pont PCI Express à la réception d'une requête d'accès

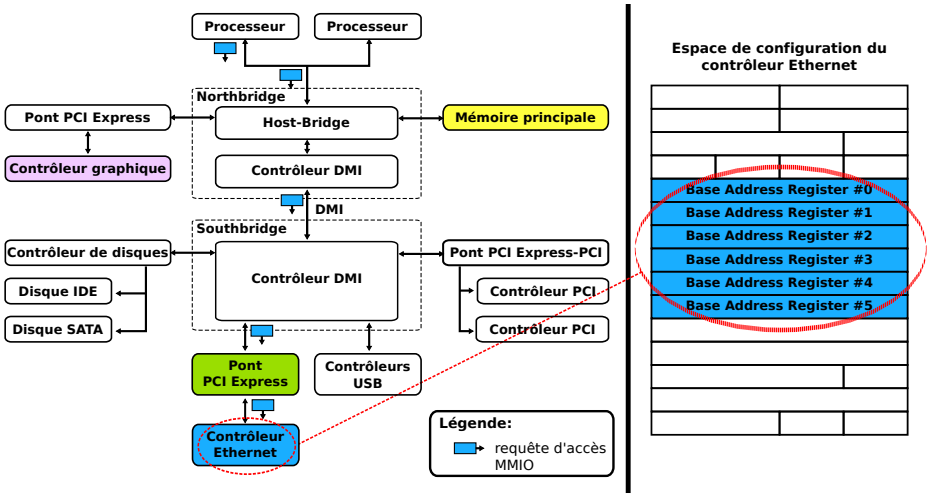

Figure 4. Actions d'un contrôleur d'entrée-sortie à la réception d'une requête

intermédiaires provoque la réservation d'un circuit virtuel. La réponse suit donc ce circuit virtuel pour être acheminée au composant à l'origine de la requête.

Le mécanisme de projection mémoire effectué par le *chipset* permet la propagation d'une requête au travers des bus PCI Express jusqu'à atteindre le contrôleur cible. À la réception d'une requête depuis le bus père, chaque pont décide de bloquer ou de transmettre cette requête à ses bus fils en fonction de sa configuration. Pour l'espace MMIO (respectivement, l'espace PIO et l'espace de configuration), la décision s'effectue en fonction d'une plage d'adresses phy-

siques (respectivement, une plage de ports et une plage de bus) mais les principes restent les mêmes, quelque soit le type d'accès effectué.

La section suivante présente la catégorie d'attaques que nous considérons dans cet article, à savoir les attaques DMA *peer-to-peer* entre contrôleurs d'entrée-sortie. Dans un premier temps, nous allons présenter ce que nous entendons par attaques DMA *peer-to-peer* et nous rappelons en quoi nous considérons ces attaques comme différentes des attaques DMA présentées jusqu'ici. Ensuite, nous illustrons ce type d'attaques au travers d'un exemple d'accès DMA *peer-to-peer*. Puis, nous décrivons les expérimentations que nous avons effectuées pour identifier les *chipsets* sur lesquels ce type d'attaques est possible. Nous discutons finalement des résultats obtenus.

3 Attaques DMA *peer-to-peer*

L'accès direct à la mémoire (ou DMA) est une fonctionnalité matérielle supportée par la majorité des contrôleurs d'entrée-sortie actuels. C'est un mécanisme qui permet au processeur principal de déléguer les transferts d'entrée-sortie à un composant spécifique que l'on nomme *DMA engine*.

De nombreuses preuves de concept d'attaques abusant de ce mécanisme existent. Celles-ci ciblent principalement la mémoire principale (ex. modification du code ou des données du noyau). Il est également possible d'abuser de ce mécanisme pour modifier la mémoire ou les registres associés à des contrôleurs d'entrée-sortie, dès lors qu'ils sont projetés dans l'espace d'adressage principal. Comme indiqué précédemment, Arrigo Triulzi s'est intéressé à cette problématique sur des architectures anciennes [30,31]. Dans cet article, nous complétons ses travaux en montrant que ces attaques sont encore d'actualité sur certains *chipsets* Intel récents. Nous tentons également de comprendre les raisons pour lesquelles ces attaques échouent sur d'autres *chipsets*.

Afin de les distinguer des attaques DMA ciblant la mémoire principale, nous allons désigner, dans la suite, par « attaque DMA *peer-to-peer* » les attaques DMA visant soit un registre, soit la mémoire embarquée dans un contrôleur d'entrée-sortie. Du point de vue d'un l'attaquant, l'intérêt de telles attaques est multiple. Ces actions malveillantes s'effectuent directement entre contrôleurs d'entrée-sortie. Il n'est pas nécessaire de faire intervenir le processeur, ni d'utiliser la mémoire principale comme relais. De ce fait, les attaques DMA *peer-to-peer* sont beaucoup plus difficiles à détecter et donc à contrer. Par ailleurs, puisqu'elles se basent directement sur des fonctionnalités matérielles, ces attaques sont généralement dépendantes du matériel mais indépendantes du système d'exploitation.

Dans la sous-section suivante, nous détaillons les mécanismes du *chipset* permettant d'acheminer une requête d'accès effectuée par un contrôleur d'entrée-sortie vers un autre contrôleur d'entrée-sortie.

3.1 Exemple d'accès DMA *peer-to-peer*

Les transferts de données entre contrôleurs d'entrée-sortie (ex. copie d'un document d'une clé USB vers un disque) s'effectuent généralement en deux étapes. Le composant source transfère dans un premier temps les données à copier dans la mémoire principale. Ensuite, le composant de destination vient lire ces données en mémoire principale. Pour réduire la charge associée à ce mode de transfert de données, certains *chipsets* permettent une communication directe entre contrôleurs d'entrée-sortie, supprimant ainsi le besoin d'utiliser la mémoire principale comme relai. Dans cette partie, nous nous intéressons aux mécanismes internes aux *chipsets* qui permettent d'acheminer une requête DMA *peer-to-peer* vers un autre contrôleur d'entrée-sortie. Nous allons détailler ces mécanismes au travers de l'exemple d'un accès depuis un contrôleur FireWire connecté au *southbridge* vers un contrôleur graphique relié au *northbridge*.

L'architecture matérielle que nous considérons dans notre exemple est représentée avec sa configuration sur la figure 5.

Le contrôleur FireWire effectue un accès en écriture à la mémoire de la carte vidéo, afin par exemple de modifier l'image qui est affichée sur le moniteur. Nous détaillons le cheminement de la requête d'accès DMA au travers du *chipset* :

1. **Le contrôleur FireWire effectue une requête DMA.** Le contrôleur effectue un accès en écriture vers l'adresse physique `0xf3000000`, laquelle correspond à une zone mémoire hébergée par le contrôleur graphique. La requête correspondante est donc générée sur le bus d'entrée-sortie auquel le contrôleur FireWire est relié. La requête est donc diffusée à tous les contrôleurs connectés à ce bus, en particulier au pont PCI Express-PCI.
2. **Le pont PCI Express-PCI relaie la requête au contrôleur DMI.** Le pont PCI Express-PCI ne projette aucun registre dans l'espace d'adressage principal. Par conséquent, il ne lui est pas nécessaire de vérifier si la requête lui est adressée. Ainsi, lors de la réception de la requête, le pont PCI Express-PCI vérifie directement si la requête reçue est destinée à l'un de ses bus fils. Comme l'adresse physique contenue dans l'accès DMA n'appartient pas à la plage d'adresses physiques associée à ses bus fils (`0xfed1c800-0xfed1e7fff`), le pont PCI Express-PCI transmet la requête au bus parent via le contrôleur DMI afin qu'un contrôleur en amont puisse la prendre en charge.
3. **Le contrôleur DMI transmet la requête au *northbridge*.** Le contrôleur DMI prend en charge la requête transmise par le pont PCI Express-PCI. Il vérifie que la requête n'est pas destinée à un de ses bus fils, puis la transmet au *northbridge*.
4. **Le *host-bridge* aiguille la requête au pont PCI Express.** Au sein du *northbridge*, c'est le *host-bridge* qui est en charge d'aiguiller la requête reçue du *southbridge* vers le bon contrôleur. Il identifie que la requête doit être transmise au pont PCI Express du *northbridge*, puis la relaie.
5. **Le pont PCI Express propage la requête au contrôleur graphique.** Le pont PCI Express vérifie si la requête lui est destinée. Dans notre exemple, ce n'est pas le cas. Il passe donc à l'étape suivante consistant à vérifier que

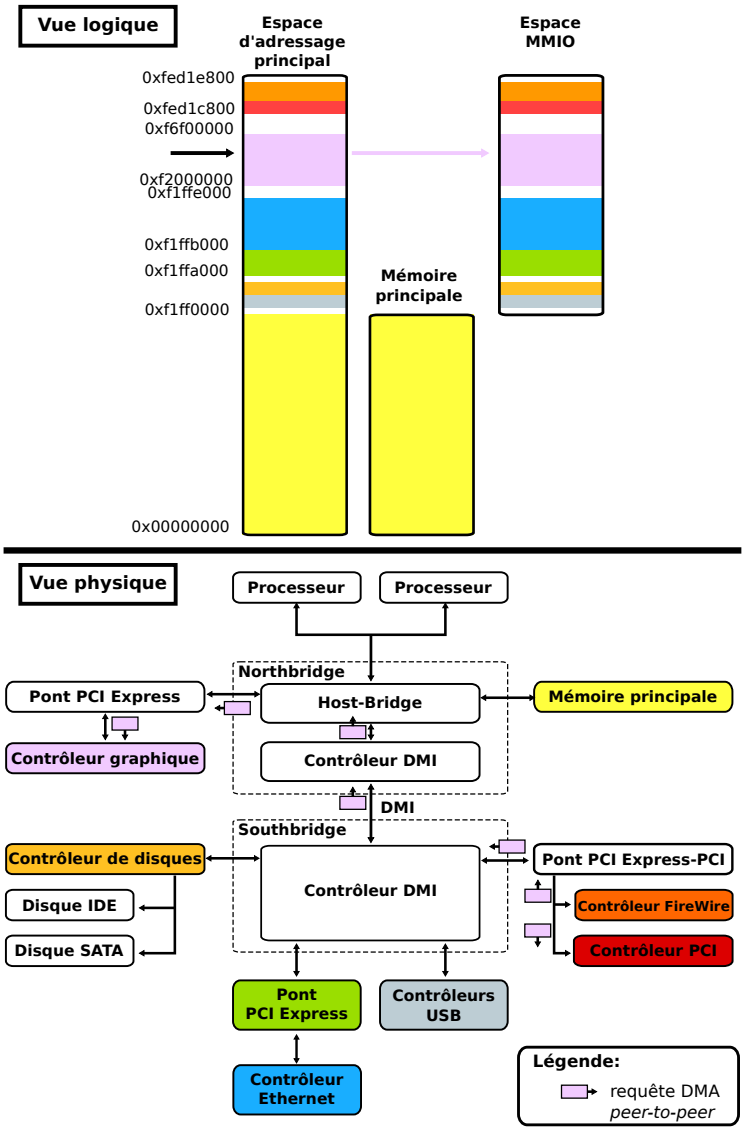

Figure 5. Illustration d'une attaque DMA *peer-to-peer*

l'accès effectué s'adresse à un composant connecté à son bus fils. L'adresse physique `0xf3000000` est bien dans l'intervalle `0xf2000000`-`0xf6efffff` : la requête est transmise au contrôleur graphique.

6. **Le contrôleur graphique traite la requête.** Finalement, le contrôleur graphique identifie la requête comme lui étant destinée, puis la traite en conséquence.

Nous retrouvons, dans cet exemple, les mécanismes similaires à ceux utilisés lors de la projection mémoire effectuée par le *chipset*. Dans la prochaine sous-section, nous détaillons les expérimentations que nous avons menées sur différents *chipsets* Intel récents afin d'identifier ceux qui permettent de mettre en œuvre ce type de transfert de contrôleur à contrôleur et donc qui rendent possible d'éventuelles attaques DMA *peer-to-peer*. En fonction des résultats obtenus, nous déduisons dans quelle mesure ces attaques sont possibles sur les *chipsets* actuels et nous expliquons les raisons pour lesquelles ces mêmes attaques peuvent échouer sur d'autres *chipsets*.

3.2 Identification de *chipsets* possédant cette fonctionnalité

La communication directe entre contrôleurs d'entrée-sortie n'est pas supportée sur tous les *chipsets* actuels. Notre étude nous a naturellement poussé à identifier ceux qui autorisent les transactions *peer-to-peer* et donc les attaques exploitant cette fonctionnalité, puis à mettre en évidence quelques contremesures. Pour cela, nous avons défini un certain nombre de tests qui nous permettent de déterminer si un *chipset* permet ou non ce mode de communication. Nous les décrivons dans la suite du document.

Descriptif des tests. Afin d'identifier les *chipsets* actuels qui permettent de mettre en place des transferts DMA *peer-to-peer*, nous avons procédé empiriquement : nous avons lancé ce type de transfert de données entre différents contrôleurs d'entrée-sortie et nous avons consigné les résultats dans un tableau.

Nous avons décrit, en section 2, le *chipset* comme la combinaison de deux composants, le *northbridge* et le *southbridge*. Ainsi, nous avons distingué dans notre tableau, dans un premier temps, les transferts DMA *peer-to-peer* initiés par des contrôleurs d'entrée-sortie connectés au *northbridge* des transferts effectués par des contrôleurs reliés au *southbridge*.

Seuls des contrôleurs d'entrée-sortie de type PCI Express sont connectés au *northbridge*. Par conséquent, ce sont les seuls capables d'effectuer des requêtes DMA. La situation est légèrement différente pour le *southbridge* car nous distinguons deux types de contrôleurs d'entrée-sortie capables d'effectuer des requêtes DMA : les contrôleurs de type PCI et les contrôleurs de type PCI Express. Aussi avons-nous distingué dans nos tests les transferts initiés depuis des contrôleurs PCI de ceux effectués depuis des contrôleurs PCI Express. Nous verrons dans la suite que cette distinction est nécessaire car les résultats diffèrent en fonction du type de contrôleur utilisé.

Finalement, nous avons analysé indépendamment, dans notre tableau, les requêtes DMA en écriture et en lecture.

Pour nos expérimentations, nous avons utilisé principalement deux contrôleurs d'entrée-sortie pour effectuer des requêtes DMA *peer-to-peer* : un contrôleur FireWire de type PCI et un autre de type PCI Express. Nous les avons connectés au *northbridge* (respectivement, au *southbridge*) grâce aux différents connecteurs d'extensions (en anglais, *slots*) présents sur la carte-mère, puis nous avons généré des requêtes de lecture puis d'écriture vers la mémoire d'autres contrôleurs

d'entrée-sortie présents sur le *chipset*. Afin de déterminer si la requête a réussi ou échoué, nous comparons son résultat avec le résultat d'une requête similaire effectuée par le processeur. Comme nos requêtes sont en concurrence avec les requêtes générées par le noyau du système d'exploitation, il est possible que les transferts échouent partiellement. Nous utilisons le degré de similitude entre les résultats pour déterminer si une requête a réussi ou échoué. Ainsi, si les résultats sont fortement similaires, nous pouvons en déduire que la requête est parvenue au contrôleur. Dans le cas contraire, nous considérons que la requête a été bloquée par un composant du *chipset*. Afin de nous assurer que l'échec d'une requête DMA n'est pas lié à notre contrôleur FireWire, nous avons réalisé de nombreux tests, en exploitant la fonction DMA dans des contrôleurs d'entrée-sortie différents (ex. contrôleurs réseaux, contrôleurs de disque).

Dans la partie qui suit, nous présentons les résultats de nos expérimentations sur différents *chipsets* actuels. Nous n'avons pas considéré, au cours de notre étude, les nouvelles architectures Intel dans lesquelles le *northbridge* est intégré directement au CPU. Il serait alors intéressant de vérifier si cette nouvelle architecture a un impact sur les résultats que nous obtenons.

Résultats d'expérimentations sur différents *chipsets*. Nous avons procédé à des expérimentations sur différents *chipsets* Intel récents. La table 1 énumère ceux que nous avons considérés dans notre étude. Pour chaque *chipset* expérimenté, nous précisons les modèles de *northbridge* et de *southbridge* concernés, ainsi que le modèle de machine sur laquelle les expérimentations ont été faites.

La table 2 consigne les résultats que nous avons obtenus. Par souci de lisibilité, nous avons regroupé les machines qui présentent des comportements similaires. Il est important de préciser que nous avons effectué deux fois les tests avec le contrôleur FireWire PCI. Dans la première, nous avons laissé intact la configuration du *chipset*. La seconde fois, marquée par le signe † dans le tableau, nous avons modifié volontairement la configuration du *southbridge*. À titre informatif, nous donnons la commande que nous avons utilisée : `setpci -s <identifiant du pont> 0x4c.b=6`. Cette commande modifie un des registres du pont PCI Express-PCI projetés dans l'espace de configuration de façon à ce qu'il transmette aux autres contrôleurs les requêtes *peer-to-peer* issues des contrôleurs PCI. Le lecteur est invité à se référer aux spécifications des *chipsets* pour obtenir plus d'informations sur les modifications opérées par cette commande. Il est probable qu'un attaquant puisse également altérer la configuration du pont PCI Express-PCI depuis un contrôleur d'entrée-sortie si celui-ci est capable de générer des requêtes de configuration sur les bus d'entrée-sortie. Des cartes FPGA ou certaines cartes réseaux telle que [2] peuvent être détournées pour effectuer de telles requêtes.

L'analyse de la table 2 nous permet d'identifier les transactions *peer-to-peer*, et donc les éventuelles attaques DMA *peer-to-peer*, qu'il est possible de mettre en place dans la majorité des *chipsets* actuels. À l'exception du *chipset* Intel x58, nous constatons que les *chipsets* que nous avons étudiés décrivent des comportements similaires.

	Chipset	*Northbridge*	*Southbridge*	Modèle de machine
Machine 0	Intel Q35	MCH 82Q35	ICH9	DELL Optiplex 755 (*Desktop*)
Machine 1	Intel Q45	MCH 82Q45	ICH10	DELL Optiplex 960 (*Desktop*)
Machine 2	Intel 945GM	GMCH 945GM	ICH7-M	MacBook Pro A1150 (*Laptop*)
Machine 3	Intel Q45 Mobile	GMCH GM45	ICH9-M	DELL Latitude E6400 (*Laptop*)
Machine 4	Intel x58	IOH x58	ICH10	Machine assemblée (*Desktop*)

Table 1. Liste des *chipsets* expérimentés

		Machine 0, Machine 1						Machine 2, Machine 3					Machine 4					
Source \ Destination		*Northbridge*	*Southbridge*					*Northbridge*	*Southbridge*				*Northbridge*	*Southbridge*				
		CG	CU	CD	CP	CR	CPe	CG	CU	CD	CP	CR	CG	CU	CD	CP	CR	CPe
Northbridge	FireWire PCIe	W	-	-	-	-	-	W	-	-	-	-	RW	RW	RW	RW	RW	RW
Southbridge	FireWire PCIe	W	-	-	-	-	NA	W	-	-	NA	-	RW	-	-	-	-	-
	FireWire PCI	W	-	-	RW	-	-	W	-	-	RW	-	RW	-	-	RW	-	-
	FireWire PCI†	W	RW	RW	RW	-	RW	W	RW	RW	RW	-	RW	RW	RW	RW	-	RW

Légende : **CG** pour contrôleur graphique **CU** pour contrôleurs USB **CPe** pour contrôleur PCI Express
CD pour contrôleurs de disques **CP** pour contrôleur PCI **CR** pour contrôleur réseau
R pour lecture réussie **W** pour écriture réussie **NA** pour non-applicable (*slots* insuffisants)
XX pour contrôleur intégré au chipset **PCI†** pour configuration du pont PCIe-PCI modifiée (bit *Peer Decode Enable* activé)

Table 2. Résultats expérimentaux

Au sein du *northbridge*, seules les requêtes d'écriture entre contrôleurs sont possibles. Pour aller plus loin dans notre analyse, nous pouvons préciser quels canaux de communication il est possible d'établir. Les requêtes d'écriture entre deux contrôleurs PCI Express connectés au *northbridge* semblent être autorisées. Il en va de même pour les requêtes d'écriture du contrôleur DMI vers un contrôleur PCI Express. En revanche, la situation inverse n'est pas possible : toute écriture depuis le *northbridge* dans un contrôleur connecté au *southbridge* est bloquée. Ces différents accès *peer-to-peer* sont particulièrement intéressants pour améliorer les performances des applications de calcul scientifique sur cartes graphiques. Il est possible, par exemple, d'exploiter ces fonctionnalités pour charger des données, issues de contrôleurs de disque ou de contrôleurs réseaux par exemple, directement dans la mémoire de la carte graphique afin que cette dernière puisse les traiter, réduisant ainsi la durée des transferts d'entrée-sortie. Soulignons cependant que les composants logiciels actuels (ex. pilotes, applications) ne sont pas conçus pour profiter de ces mécanismes et qu'il est probablement nécessaire d'y effectuer des changements conséquents pour cela. Du point de vue d'un attaquant, la restriction aux requêtes d'écriture au sein du *northbridge* limite énormément les possibilités d'attaque. L'attaquant doit connaître *a priori* la configuration du *chipset* afin de localiser les zones où écrire. Par ailleurs, comme il ne peut pas faire de lecture vers les contrôleurs d'entrée-sortie, il n'a aucun retour direct sur la réussite ou l'échec de son attaque.

En ce qui concerne les différents modèles de *southbridge* que nous avons étudiés, les résultats sont cohérents d'un *chipset* à l'autre. Il est possible d'établir un canal de communication depuis un contrôleur d'entrée-sortie quelconque vers un contrôleur du *northbridge*. Les requêtes autorisées entre ces contrôleurs sont ensuite définies par le *northbridge*. Les contrôleurs PCI également peuvent communiquer directement entre-eux puisqu'ils sont connectés au même bus d'entrée-sortie. Finalement, il est possible d'effectuer, depuis ces mêmes contrôleurs, des requêtes de lecture et d'écriture vers d'autres contrôleurs PCI Express dès lors que la configuration initiale du pont PCI Express-PCI a été altérée pour les permettre. Le *southbridge* est alors plus enclin que le *northbridge* à autoriser les accès DMA *peer-to-peer*. Pour ces raisons, il est plus intéressant, du point de vue d'un attaquant, d'exploiter un contrôleur connecté au *southbridge* pour effectuer des attaques DMA *peer-to-peer*.

La possibilité d'effectuer des accès *peer-to-peer* au sein du *chipset* résulte probablement d'un choix de conception, lié aux différents cas d'utilisation que nous pouvons imaginer. Cependant, nous avons remarqué, au cours de nos expérimentations, que ces accès, bien que légitimes, sont dans certains cas trop permissifs et peuvent induire par conséquent des problèmes de sécurité. En effet, le *chipset* vérifie uniquement qu'un composant est autorisé à communiquer avec un autre composant. Aucune restriction n'existe quant au contenu auquel le contrôleur malveillant accède. Sur certains *chipsets*, nous avons pu par exemple provoquer l'extinction du moniteur en écrivant, depuis un contrôleur FireWire, dans les registres de la carte graphique.

À partir de là, nous pouvons nous demander les raisons pour lesquelles les requêtes DMA *peer-to-peer* peuvent échouer. En pratique, la requête *peer-to-peer* est bloquée par un des contrôleurs intermédiaires traversés par la requête. Lorsqu'un contrôleur intermédiaire détecte qu'une requête doit être aiguillée sur un autre de ses bus fils, ce dernier prend la décision de bloquer ou non la requête. Ainsi, le *host-bridge* limite les requêtes *peer-to-peer* au sein du *northbridge*. De manière analogue, c'est le contrôleur DMI qui est en charge de le faire dans le *southbridge*.

Parmi les *chipsets* que nous avons étudiés, le modèle Intel x58 est le plus intéressant pour effectuer des attaques DMA *peer-to-peer*. En effet, il est beaucoup plus permissif que les autres *chipsets*. Il est important de préciser qu'il existe probablement d'autres modèles de *chipsets* aussi permissifs. Nous pensons en particulier aux *chipsets* Intel 5100 [15], Intel 5500 et Intel 5520 [6], lesquels peuvent être disponibles sur les serveurs dotés de processeurs Intel Xeon. Faute de matériel, nous n'avons cependant pas pu effectuer d'expérimentation sur ces *chipsets*.

Dans la section suivante, nous allons nous focaliser sur le *chipset* Intel x58 et présenter un exemple d'attaque sur une carte graphique exploitant la possibilité d'effectuer des échanges *peer-to-peer* au sein du *chipset*.

4 Preuve de concept d'attaque sur une carte graphique

Nous avons implémenté une preuve de concept afin de montrer ce qui est réalisable en pratique avec cette classe d'attaque. Dans celle-ci, nous exploitons le mécanisme DMA d'un contrôleur malveillant (ex. carte FireWire, carte réseau) pour manipuler la mémoire vidéo d'une carte graphique. Nous rappelons brièvement pour commencer l'architecture d'une carte graphique.

4.1 Architecture d'une carte graphique

La figure 6 présente une vue simplifiée de l'architecture d'une carte graphique. La connaissance des éléments qui la constituent permet de mieux appréhender le scénario d'attaque que nous détaillons dans la suite.

Une carte graphique est constituée de divers composants qui collaborent entre eux pour produire une image que l'on affiche sur un moniteur. Parmi ces composants, nous distinguons :

La mémoire de la carte. La mémoire embarquée dans la carte graphique a différentes fonctions. Elle se découpe en plusieurs régions de mémoire parmi lesquelles :

- les registres de contrôle qui permettent au système d'exploitation de piloter les différents composants de la carte graphique. En particulier, ils permettent de soumettre des commandes au processeur graphique.
- la mémoire vidéo (ou *framebuffer*) qui sert à stocker les pixels produits par le processeur graphique qui seront par la suite convertis en signaux analogiques ou numériques afin d'être affichés sur un moniteur.

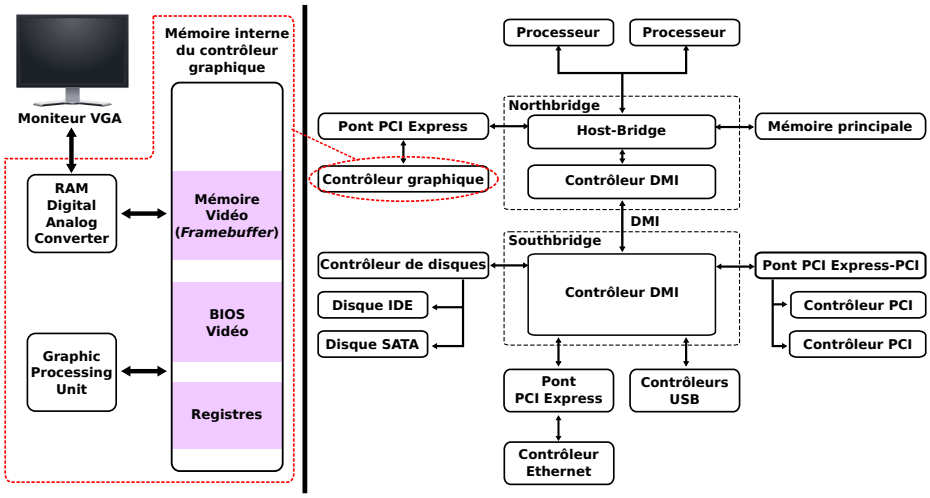

Figure 6. Architecture simplifiée d'une carte graphique

– la mémoire VBIOS ou BIOS vidéo qui contient les routines exécutées par le BIOS au démarrage de la machine. Ces routines permettent par exemple d'initialiser la carte vidéo et de l'utiliser dès le démarrage.

Le *Graphics Processing Unit* (GPU). Il s'agit du processeur graphique qui permet de décharger le processeur principal des calculs graphiques. Il traite les objets graphiques (ex. ensemble de points ou de lignes dans l'espace) fournis par le système d'exploitation et en déduit les pixels à afficher. Les pixels calculés sont stockés dans la mémoire vidéo.

Le *RAM Digital-Analog Converter* (RAMDAC). Cet élément se charge de convertir le contenu de la mémoire vidéo en signaux analogiques compatibles avec la norme VGA des moniteurs.

Dans notre preuve de concept, nous nous intéressons uniquement à la mémoire vidéo de la carte graphique car c'est elle qui détermine ce qui est affiché sur le moniteur. Nous utilisons un contrôleur malveillant pour manipuler par DMA cette mémoire afin de permettre la recopie à distance de ce qui est affiché sur la machine victime ou de faire afficher à la machine victime un contenu de notre choix. Les prochaines sous-sections détaillent le scénario d'attaque que nous avons considéré et discutent des résultats obtenus.

4.2 Scénario d'attaque considéré

Dans notre scénario d'attaque, nous considérons deux acteurs : une victime, que nous nommons dans la suite *Bob*, et un attaquant, que nous appelons *Ève*. Nous supposons que la machine qu'utilise quotidiennement *Bob* possède un *chipset* sur lequel il est possible de mettre en place des communications *peer-to-peer*.

Nous faisons également l'hypothèse qu'*Ève* a la maîtrise d'un contrôleur d'entrée-sortie placé dans la machine de *Bob*, à partir duquel elle peut d'effectuer des attaques DMA. Elle peut, par exemple, exploiter une vulnérabilité dans la carte réseau [11] présente dans la machine de *Bob* et y placer un *rootkit* comme présenté dans [30,31,7]. Pour rendre l'attaque plus simple, nous considérons qu'*Ève* commande le contrôleur FireWire présent dans la machine de *Bob* grâce à un périphérique qui lui est connecté (ex. un iPod modifié, un ordinateur portable). Plusieurs articles récents ont montré le réalisme pratique d'une telle supposition. Nous rappelons qu'un contrôleur FireWire autorise les périphériques qui lui sont connectés à fournir les adresses physiques vers lesquelles effectuer les accès DMA. Cette hypothèse implique donc qu'*Ève* possède un accès physique à la machine de *Bob*. Grâce à cet accès physique (ici, un port FireWire), *Ève* connecte son ordinateur portable à la machine de *Bob* pour piloter le contrôleur FireWire grâce à ce dernier. Elle exploite cet accès pour manipuler la mémoire vidéo de la carte graphique de *Bob*. La preuve de concept que nous avons conçue montre qu'*Ève* peut dans cette situation lire la mémoire vidéo. Dans cette preuve de concept, nous utilisons le résultat de cette lecture pour afficher l'écran de *Bob* sur une autre machine, l'ordinateur portable d'*Ève*. Nous aurions pu de la même façon modifier ce qui est affiché sur l'écran de *Bob* en écrivant par DMA dans cette même mémoire vidéo. La figure 7 représente ce scénario d'attaque.

4.3 Résultats obtenus

Une vidéo présentant la mise en œuvre de notre preuve de concept est disponible [20]. Nous avons déroulé l'attaque sur une machine de type *desktop* disposant d'une carte-mère Intel DX58SO, laquelle embarque un *chipset* Intel x58 (plus précisément, IOH x58 / ICH10). Sur cette vidéo, nous attaquons une carte graphique nVidia GeForce 9800 GT [24] depuis un contrôleur FireWire de type PCI, piloté depuis un périphérique malveillant (ici, un ordinateur portable). Nous reconstituons l'écran de *Bob* à raison de deux images par secondes. Nous considérons cela tout-à-fait acceptable pour récupérer des informations affichées à l'écran (ex. couple identifiant-mot de passe), mais insuffisant pour récupérer un flux en temps réel (ex. capture d'un flux vidéo).

Nous avons déterminé plusieurs facteurs qui impactent les performances que nous obtenons pour notre attaque :

- L'application exécutée sur l'ordinateur portable d'*Ève*, qui lit la mémoire de la carte graphique au travers du contrôleur FireWire, a été développée dans le langage Python [29]. Les bibliothèques de fonctions que nous avons utilisées pour communiquer sur les bus FireWire et reconstituer l'écran de *Bob* sont des *wrappers* Python de bibliothèques développées dans le langage C [16]. Nous aurions pu obtenir des performances meilleures en évitant cette sur-couche et en implémentant notre application dans un langage natif.
- Dans notre attaque, nous avons utilisé le contrôleur FireWire disponible sur la carte-mère. Ce contrôleur est de type PCI. Comme la bande passante des bus PCI Express est supérieure à celle des bus PCI, nous aurions pu

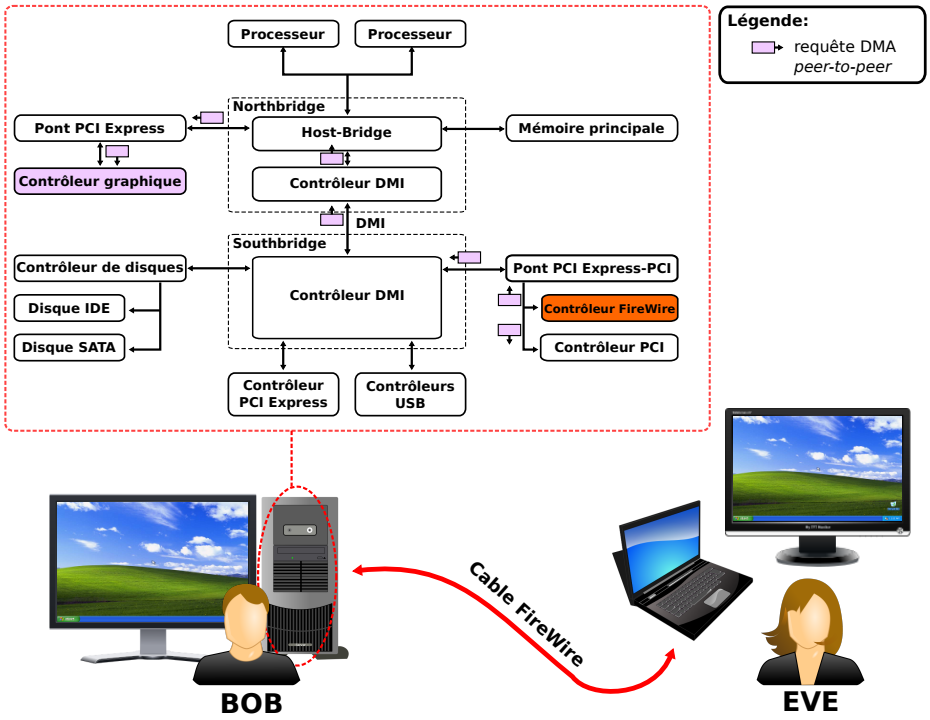

Figure 7. Scénario d'attaque

améliorer la fluidité de la recopie d'écran en impliquant uniquement des contrôleurs de type PCI Express dans l'attaque. Cependant, à l'heure actuelle, peu de cartes-mère embarquent un contrôleur FireWire PCI Express et nous avons voulu nous rapprocher au mieux de la réalité.

– Nous avons remarqué au cours de nos expérimentations que les performances de notre attaque sont fortement dépendantes du contrôleur d'entrée-sortie que nous ciblons. En effet, certains contrôleurs d'entrée-sortie ne gèrent que des accès par octets, d'autres par mots, *etc.* Les types d'accès supportés par le contrôleur d'entrée-sortie attaqué impactent directement les résultats des expérimentations.

5 Contre-mesures

Afin de maîtriser les transferts de données directs entre contrôleurs d'entrée-sortie, nous pouvons nous appuyer sur différents mécanismes matériels existants. Dans cette section, nous présentons les technologies sur lesquelles nous avons effectué des expérimentations, et nous discutons de leur mise en œuvre dans les *chipsets* actuels.

5.1 *Input/Output Memory Management Unit* (I/O MMU)

Il est possible d'utiliser des technologies matérielles implémentant le principe d'I/O MMU (telles qu'AMD IOMMU [1] ou Intel VT-d [14]) pour limiter les capacités d'interaction *peer-to-peer* entre contrôleurs d'entrée-sortie.

Une I/O MMU est un composant matériel intégré aux *chipsets* récents qui permet de virtualiser l'espace d'adressage physique pour les contrôleurs d'entrée-sortie et de filtrer entre autres leurs accès à cet espace. Les adresses utilisées pour les transferts par les contrôleurs d'entrée-sortie sont dans ce cas virtuelles, car elles sont susceptibles d'être traduites par l'I/O MMU. En effet, lorsqu'une requête DMA transite dans une I/O MMU, celle-ci l'intercepte et traduit l'adresse DMA contenue dans la requête en une autre adresse. Cette adresse correspond à une adresse physique si la requête est destinée à la mémoire principale ou à une autre adresse DMA virtuelle si la requête est destinée à un autre contrôleur d'entrée-sortie. Il est également possible de spécifier, pour chaque contrôleur d'entrée-sortie, des permissions d'accès en lecture et/ou en écriture à la mémoire virtuelle qui leur est associée. La configuration de ces services s'effectue par l'intermédiaire d'un jeu de tables de configuration stockées en mémoire principale, mises en place et protégées par le système d'exploitation.

Dans les *chipsets* actuels, les composants implémentant une I/O MMU sont généralement placés dans le *northbridge*, en coupure entre le bloc formé par les contrôleurs d'entrée-sortie et la mémoire principale. L'agencement de ces composants dans l'architecture matérielle fait en sorte qu'ils voient transiter toute requête à destination de la mémoire principale. De ce fait, ils protègent efficacement la mémoire principale de tout accès arbitraire à celle-ci depuis un contrôleur d'entrée-sortie. En ce qui concerne les communications *peer-to-peer*, tout échange d'un contrôleur d'entrée-sortie dans le *northbridge* vers un contrôleur quelconque, qu'il soit situé dans le *northbridge* ou le *southbridge*, transite obligatoirement par le composant implémentant une I/O MMU. Ainsi, ces échanges peuvent être contrôlés. Il en va de même pour la situation inverse, c'est-à-dire que toute communication à destination d'un composant du *northbridge* est également maîtrisée. En revanche, lorsque les échanges n'impliquent que des composants situés dans le *southbridge*, l'I/O MMU n'est d'aucun secours. En effet, comme ces échanges s'effectuent en interne dans le *southbridge*, ceux-ci ne transitent à aucun moment dans un composant implémentant une I/O MMU. Pour cette raison, les requêtes ne peuvent être contrôlées. Il est assez simple de vérifier cela sur les modèles de *southbridge* actuels en mettant en place un transfert DMA *peer-to-peer* d'un contrôleur PCI vers d'autres contrôleurs du *southbridge*.

Afin de contrôler entièrement les communications au sein du chipset, il est envisageable d'avoir à la fois une I/O MMU dans le *northbridge* et une dans le *southbridge*. Cependant, cette solution rend encore plus complexe l'architecture matérielle résultante car il est nécessaire que les deux I/O MMUs se fassent mutuellement confiance : si une première I/O MMU a déja effectué un contrôle d'accès sur une requête, la seconde I/O MMU peut ne pas vérifier une seconde fois le même accès. Dès lors qu'une des deux I/O MMUs est corrompue, la sécurité du système peut être remise en question.

Une autre manière de se protéger serait par exemple de rediriger systématiquement toutes les communications internes au *southbridge* vers le *northbridge* pour que ceux-ci soient vérifiées par l'I/O MMU. Bien que cette solution puisse induire des performances moindres, c'est vers cette logique qu'évolueront les prochaines générations de *chipsets* grâce à l'extension *Access Control Services* définie dans le standard PCI Express.

5.2 *Access Control Services* (ACS)

Afin de répondre aux problèmes liés aux communications *peer-to-peer* dans les architectures actuelles, le PCI Special Interest Group (PCI-SIG), consortium en charge de rédiger les spécifications PCI Express, a défini de nouvelles fonctionnalités matérielles regroupées sous le nom d'*Access Control Services*. Ces fonctionnalités matérielles permettent de mettre en œuvre différents types de contrôle d'accès dans les composants PCI Express qui les implémentent. Nous décrivons ci-dessous quelques fonctionnalités définies par le PCI-SIG à titre d'exemple :

- ***ACS Source Validation.*** Ce service permet d'adresser une partie des problèmes liés à l'usurpation d'identité d'un contrôleur d'entrée-sortie. En effet, les contrôleurs qui implémentent ce service vérifient systématiquement que les requêtes reçues sur les ports fils utilisent bien l'identité d'un contrôleur fils. Ce type de vérification empêcherait par exemple un périphérique corrompu d'envoyer une requête DMA pour le compte d'un autre périphérique et ainsi d'accéder aux zones mémoires qui lui sont réservées.
- ***ACS Upstream Forwarding.*** Ce service permet de rediriger systématiquement toute requête reçue depuis un port fils vers le port père, même si celui-ci est à destination d'un autre port fils. Cette fonctionnalité est intéressante par exemple pour rediriger les requêtes internes du *southbridge* au *northbridge* afin que celles-ci soient validées par une I/O MMU.
- ***ACS P2P Egress Control.*** Ce service permet de bloquer toute communication *peer-to-peer* d'un port fils à destination d'un autre port fils.

La mise en œuvre de ces extensions au sein du *southbridge* permet de définir des comportements différents vis-à-vis des communications entre contrôleurs d'entrée-sortie. Par exemple, la mise en place de l'*ACS Upstream Forwarding* au sein du *southbridge* implique de rediriger vers le *northbridge* toutes les requêtes internes au *southbridge*. Ces requêtes peuvent être contrôlées dès lors qu'une I/O MMU est activée au sein du *northbridge*, puis être ré-aiguillées par la suite au *southbridge*. Il est également possible de bloquer systématiquement toute communication entre contrôleurs d'entrée-sortie connectés au *southbridge* en activant l'*ACS P2P Egress Control*. Par défaut, aucun de ces services de contrôle d'accès n'est activé. Le système d'exploitation est chargé de les activer et de les configurer. Dans le cas d'un système Linux, ces fonctionnalités sont detectées par le système d'exploitation mais ne sont jamais activées. Il est à l'heure actuelle nécessaire de les configurer à la main, en modifiant directement la configuration des composants. Il convient cependant d'être prudent dans l'activation de ces mécanismes. Les pilotes des différents contrôleurs d'entrée-sortie ont été écrits sans prendre en compte l'existence de tels dispositifs de filtrage ou

de virtualisation de l'espace mémoire. Il existe donc un risque que l'activation de ces mécanimes bloque le fonctionnement de certains périphériques.

Ces fonctionnalités matérielles sont relativement récentes et commencent à être intégrées aux nouveaux *chipsets*. À ce jour, elles sont mises en œuvre uniquement dans des composants du *northbridge*. Nous espérons que les prochaines générations de *chipsets* proposeront leur mise en œuvre au sein de *southbridge* afin de mieux contrôler les échanges entre contrôleurs d'entrée-sortie connectés à ce dernier.

6 Conclusion et perspectives

Aujourd'hui, les attaques mettant en œuvre le matériel deviennent de plus en plus nombreuses. Un contrôleur d'entrée-sortie peut par exemple contenir une porte dérobée (*backdoor*) dès sa fabrication. Il est également possible d'exploiter à distance le *firmware* d'un contrôleur d'entrée-sortie et d'y cacher un *rootkit*.

Cet article part de l'hypothèse qu'un contrôleur d'entrée-sortie peut être exploité par un attaquant. Nous nous intéressons aux actions malveillantes qu'il est possible d'effectuer avec ce seul point d'entrée, et cela malgré les différents mécanismes de protection qui existent actuellement. Notre étude se focalise de manière générale sur les attaques DMA. Associées à tort à des attaques ciblant uniquement la mémoire principale, nous montrons dans cet article que les attaques DMA peuvent cibler également les registres et les mémoires embarqués des contrôleurs d'entrée-sortie. Le grand avantage de ce type d'attaques est qu'elles sont pratiquement indétectables par des programmes s'exécutant sur l'ordinateur cible. Afin d'illustrer cela, nous détaillons une preuve de concept d'attaque sur une carte graphique depuis un contrôleur FireWire. Nous décrivons également dans cet article les mécanismes matériels existants qui limitent l'impact de cette classe d'attaque et nous discutons de leur mise en œuvre dans les *chipsets* actuels et futurs.

Nous nous sommes intéressés, dans cet article, à un type d'action malveillante qu'il est possible de mettre en œuvre depuis un contrôleur d'entrée-sortie. En perspective à nos travaux, il serait intéressant de voir quelles autres actions malveillantes sont également possibles depuis des contrôleurs d'entrée-sortie de type PCI ou PCI Express. Comme les contrôleurs d'entrée-sortie offrent actuellement peu de souplesse, nous pensons implémenter un contrôleur d'entrée-sortie à base de FPGA nous permettant de générer différents types de requêtes sur les bus d'entrée-sortie. À partir de là, nous serons en mesure de tester exhaustivement les fonctionnalités d'un *chipset* et d'y déceler des faiblesses (ex. reconfigurer un contrôleur à partir d'un autre, contourner une I/O MMU).

Remerciements

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

Références

1. Advanced Micro Devices (AMD) : *AMD I/O Virtualization Technology (IOMMU) - Architectural Specification*, février 2009. http://support.amd.com/us/Processor_TechDocs/48882.pdf.
2. Alteon Networks : *Gigabit Ethernet/PCI Network Interface Card - Host/-NIC Software Interface Definition*, juin 1999. https://lhcb-comp.web.cern.ch/lhcb-comp/DAQ/Event-Building/docs/pcinic.pdf.
3. Aumaitre, Damien : 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, Rennes, juin 2008. http://actes.sstic.org/SSTIC08/Voyage_Coeur_Memoire/.
4. Adam Boileau : Hit by a Bus : Physical Access Attacks with FireWire. *In RUXCON 2006*, octobre 2006. http://www.ruxcon.org.au/files/2006/firewire_attacks.pdf.
5. Brian Carrier et Joe Grand : A Hardware-based Memory Acquisition Procedure for Digital Investigations. *Digital Investigation Journal*, 1(1):50–60, février 2004. http://www.digital-evidence.org/papers/tribble-preprint.pdf.
6. Intel Corporation : *Intel® 5520 Chipset and Intel® 5500 Chipset - Datasheet*, mars 2009. http://www.intel.com/assets/pdf/datasheet/321328.pdf.
7. Guillaume Delugré : Closer to metal : reverse-engineering the Broadcom NetExtreme's firmware. *In Hack.lu*, Luxembourg, 27-29 octobre 2010. http://esec-lab.sogeti.com/dotclear/public/publications/10-hack.lu-nicreverse_slides.pdf.
8. Christophe Devine et 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, juin 2009. http://actes.sstic.org/SSTIC09/Compromission_physique_par_le_bus_PCI/.
9. Maximillian Dornseif : 0wned by an iPod - hacking by Firewire. *In PacSec/core04*, 11-12 novembre 2004. http://md.hudora.de/presentations/#firewire-pacsec.
10. Loïc Duflot : *Contribution à la sécurité des systèmes d'exploitation et des microprocesseurs*. Thèse de doctorat, Université de Paris XI, octobre 2007. http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lti/these-duflot.pdf.
11. Loïc Duflot, Yves-Alexis Perez, Guillaume Valadon et Olivier Levillain : Can you still trust your Network Card ? *In CanSecWest/core10*, 24-26 mars 2010. http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf.
12. Loïc Duflot, Yves-Alexis Perez, Guillaume Valadon et Olivier Levillain : Quelques éléments en matière de sécurité des cartes réseau. *In Actes du 8ème Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2010)*, pages 213–235, Rennes, juin 2010. http://www.sstic.org/media/SSTIC2010/SSTIC-actes/Peut_on_faire_confiance_aux_cartes_reseau/.
13. Intel Corporation : *Universal Host Controller Interface (UHCI) Design Guide*, mars 1996. http://download.intel.com/technology/usb/UHCI11D.pdf.
14. Intel Corporation : *Intel Virtualization Technology for Directed I/O - Architecture Specification*, septembre 2008. http://download.intel.com/technology/computing/vptech/Intel(r)_VT_for_Direct_IO.pdf.

15. Intel Corporation : *Intel® 5100 Memory Controller Hub Chipset - Datasheet*, juin 2009. http://www.intel.com/Assets/PDF/datasheet/318378.pdf.
16. W. Brian Kernighan et M. Dennis Ritchie : *The C Programming Language*. Prentice Hall, Inc., février 1978.
17. Éric Lacombe, Vincent Nicomette et Yves Deswarte : Une approche de virtualisation assistée par le matériel pour protéger l'espace noyau d'actions malveillantes. *In Actes du 7ème Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2009)*, pages 321–346, Rennes, juin 2009. http://actes.sstic.org/SSTIC09/Une_approche_de_virtualisation_assistee_par_le_materiel_pour_proteger_l_espace_noyau_d-actions_malveillantes/.
18. Éric Lacombe : *Sécurité des noyaux de systèmes d'exploitation*. Thèse de doctorat, INSA de Toulouse, 2009. http://tel.archives-ouvertes.fr/docs/00/46/25/34/PDF/these-eric_lacombe.pdf.
19. Fernand Lone Sang, Éric Lacombe, Vincent Nicomette et Yves Deswarte : Analyse de l'efficacité du service fourni par une IOMMU. *In Actes du 8ème Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 2010)*, pages 189–214, Rennes, juin 2010. http://www.sstic.org/media/SSTIC2010/SSTIC-actes/Analyse_de_l_efficacite_du_service_fourni_par_une_/.
20. Fernand Lone Sang, Vincent Nicomette et Yves Deswarte : Démonstration d'une attaque DMA *peer-to-peer* depuis un périphérique FireWire vers un contrôleur graphique, janvier 2011. http://homepages.laas.fr/nicomett/Videos/.
21. Antonio Martin : FireWire memory dump of a Windows XP computer : a forensic approach. Rapport technique, FriendsGlobal, 2007. http://www.friendsglobal.com/papers/FireWire%20Memory%20Dump%20of%20Windows%20XP.pdf.
22. David Maynor : 0wn3d by everything else - USB/PCMCIA Issues. *In CanSecWest/core05*, 4-5 mai 2005. http://cansecwest.com/core05/DMA.ppt.
23. Michael Becher and Maximillian Dornseif and Christian N. Klein : FireWire - all your memory are belong to us. *In CanSecWest/core05*, 4-5 mai 2005. http://md.hudora.de/presentations/#firewire-cansecwest.
24. NVIDIA Corporation : Nvidia GeForce 9800 GT - fiche produit, 2010. http://www.nvidia.com/object/product_geforce_9800gt_us.html.
25. OpenBSD core team : The OpenBSD project, 2010. http://www.openbsd.org.
26. PCI Special Interest Group (PCI-SIG) : *PCI-to-PCI Bridge Architecture Specification - Revision 1.1*, décembre 1998. http://www.pcisig.com/specifications/conventional/pci_to_pci_bridge_architecture/.
27. PCI Special Interest Group (PCI-SIG) : *PCI Local Bus Specification - Revision 2.3*, mars 2002. http://www.pcisig.com/specifications/conventional/conventional_pci_23/.
28. PCI Special Interest Group (PCI-SIG) : *PCI Express™ Base Specification - Revision 1.1*, mars 2005. http://www.pcisig.com/specifications/pciexpress/specifications/.
29. Python Software Foundation : Python Programming Language - Official Website, 2010. http://www.python.org/.
30. Arrigo Triulzi : Project Maux Mk.II - « I 0wn the NIC, Now I want a Shell! ». *In PacSec/core08*, 12-13 novembre 2008. http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-PACSEC08-Project-Maux-II.pdf.

31. Arrigo Triulzi : The Jedi Packet Trick takes over the Deathstar (or : « Taking NIC Backdoors to the Next Level »). *In CanSecWest/-core10*, 24-26 mars 2010. http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-CANSEC10-Project-Maux-III.pdf.