Protection des systèmes informatiques contre les attaques par entrées-sorties

Fernand Lone Sang

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Discipline ou spécialité :

Domaine STIC : Réseaux, Télécommunications, Systèmes et Architecture

---

Présentée et soutenue par :

Fernand Lone Sang

Le 27 novembre 2012

Titre :

Protection des systèmes informatiques contre les attaques par entrées-sorties

---

JURY

Rapporteurs :	Olivier Festor	INRIA Lorraine - LORIA
	Jean-Louis Lanet	Université de Limoges
Examinateurs :	Loïc Duflot	ANSSI
	Mohamed Kaâniche	LAAS-CNRS
	Frédéric Raynal	QuarksLAB
Directeurs de Thèse :	Yves Deswarte	LAAS-CNRS
	Vincent Nicomette	INSA de Toulouse

École doctorale :

École Doctorale Mathématique Informatique Télécommmunications de Toulouse (EDMITT)

Unité de recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS)

Directeurs de Thèse :

Yves Deswarte et Vincent Nicomette

Remerciements

Les travaux présentés dans ce mémoire ont été effectués au sein du Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Je remercie Messieurs Raja Chatila, Jean-Louis Sanchez et Jean Arlat, qui ont successivement assuré la direction du LAAS-CNRS depuis mon entrée, pour m'avoir accueilli au sein de ce laboratoire. Je remercie également Madame Karama Kanoun et Monsieur Mohamed Kaâniche, Directeurs de Recherche CNRS, responsables successifs de l'équipe de recherche Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF), pour m'avoir permis de réaliser ces travaux dans cette équipe et pour avoir fourni un cadre propice au travail et aux échanges qui sont indispensables pour effectuer une thèse dans les meilleures conditions.

Je remercie ensuite mes directeurs de thèse, Monsieur Yves Deswarte, Directeur de Recherche CNRS, et Monsieur Vincent Nicomette, Professeur des Universités à l'Institut National des Sciences Appliquées (INSA) de Toulouse, pour m'avoir guidé durant ces trois années de thèse. Ces travaux n'auraient pas pu aboutir sans leur concours, leurs précieux conseils, leur sens de l'écoute, leur soutien constant et leur disponibilité malgré un emploi du temps très (très) chargé. Au delà de leurs qualités humaines, je tiens également à souligner leurs compétences scientifiques et leur grande culture en sécurité informatique dont j'ai pu bénéficier. Travailler avec eux durant ces trois années fut très appréciable, une expérience extrêmement enrichissante, mais surtout une grande chance. Merci à eux pour la confiance qu'ils m'ont toujours témoignée et, dans les quelques moments de doute, pour leur aide qui a contribué au bon déroulement des travaux présentés dans ce mémoire.

J'adresse également mes sincères remerciements aux membres du jury qui ont accepté de juger mon travail. Je leur suis très reconnaissant pour l'intérêt qu'ils ont porté à mes travaux :

— Yves Deswarte, Directeur de Recherche au LAAS-CNRS, Toulouse ;
— Loïc Duflot, Ingénieur de Recherche à l'ANSSI, Paris ;
— Olivier Festor, Directeur de Recherche à l'INRIA Lorraine, Nancy ;
— Mohamed Kaâniche, Directeur de Recherche au LAAS-CNRS, Toulouse ;
— Jean-Louis Lanet, Professeur à l'Université de Limoges ;
— Vincent Nicomette, Professeur à l'INSA de Toulouse ;
— Frédéric Raynal, Ingénieur de Recherche et Fondateur de QuarksLAB, Paris.

Je suis ravi et honoré que Messieurs Olivier Festor et Jean-Louis Lanet aient accepté d'être les rapporteurs de cette thèse, je les en remercie profondément. Les judicieux conseils qu'ils m'ont prodigués ont contribué à améliorer la qualité du manuscrit que vous avez entre les mains. Des remerciements particuliers vont à Monsieur Mohamed Kaâniche pour m'avoir fait l'honneur de présider le jury en dépit de ses nombreuses obligations. Merci également à Monsieur Adrian Perrig pour les discussions que nous avons eues sur plusieurs travaux qui sont décrits dans ce manuscrit. Cela aurait été un réel honneur de l'avoir dans ce jury, mais il n'a malheureusement pas pu se libérer en raison de son emploi du temps extrêmement chargé.

Les remerciements suivants s'adressent à Éric Lacombe qui, alors que j'étais en projet de fin d'études, m'a initié aux méandres du noyau Linux et a tracé la voie aux travaux qui sont présentés dans ce manuscript. Il m'a présenté d'autres facettes de la sécurité informatique, initié à la programmation bas niveau et m'a encouragé à poursuivre cette voie alors que je venais du domaine des réseaux. Il m'a également permis de rencontrer et de cotoyer des gens extrêmement talentueux. Pour tout cela, je lui suis particulièrement reconnaissant.

Je remercie également tous les membres de l'équipe de recherche TSF, permanents, doctorants, post-doctorants et stagiaires avec lesquels j'ai partagé ces années de travail. Merci à tous les membres de l'équipe pour leur sympathie et leur soutien. Une mention particulière à Yves

Crouzet pour les nombreux prêts de matériels durant ma thèse : puisse la *Crou-Crou'Tech* toujours s'enrichir de petites bricoles toujours utiles et qui sont une mine d'or pour les /a.kœʁ/ ; à Géraldine Vache-Marconato pour les nombreuses péripéties (techniques et non techniques) extraordinaires qui ont ponctué ma vie de doctorant ; et à Éric Alata pour m'avoir appris à persévérer davantage face à un problème (très) difficile et pour m'avoir continuellement encouragé à terminer le Challenge SSTIC avec lui. Enfin, j'ai partagé toute cette thèse avec ceux qui ont été mes collègues de bureau (Anthony, Maxime) et ceux qui le sont encore (Miruna, Miguel et Rim), qui ont toujours été un support pour moi. En particulier, je dois beaucoup à Miruna qui a accepté la pénible tâche de relire mes articles ainsi que mes chapitres de thèse pour chasser toutes les coquilles qui restaient. Elle a énormément contribué à la qualité de ce manuscrit.

Je n'oublie bien sûr pas de remercier l'ensemble des services techniques et administratifs du LAAS-CNRS, qui contribuent activement à la vie du laboratoire. Je tiens à remercier particulièrement Matthieu Herrb, Ingénieur de Recherche, pour sa disponibilité permanente pour des réunions durant lesquelles il nous a fait profiter de son savoir technique incommensurable, de son expérience en programmation bas-niveau. Matthieu, sache que tu as été une réelle source d'inspiration et de motivation pour moi.

Mes remerciements s'adressent également à l'ensemble des personnes qui m'ont tendu leurs mains au cours ma thèse et même après cette aventure tumultueuse. Pour commencer, un grand Merci au comité d'organisation et au comité de programme de SSTIC pour m'avoir laissé présenter mes travaux trois années de suite à ce symposium francophone de qualité sur la sécurité informatique. Merci, ensuite, à l'ensemble des membres de l'ANSSI avec qui j'ai pu converser et avec qui j'ai eu la chance de travailler. J'ai énormément apprécié les échanges au niveau scientifique et humain. Je souhaite que la synergie qui s'est créée perdure au delà des trois années qui viennent de s'écouler et que nous tirions de nombreux enseignements de nos compétences et de nos expériences respectives. Merci également à Ludovic Mé et Carlos Aguilar Melchor pour m'avoir ouvert les portes du monde académique après ma thèse, puis Fabien Perigaud, Florent Marceau, Arnauld Mascret et Frédéric Raynal, pour les portes du monde de l'industrie. Merci à tous pour la confiance que vous m'avez accordée, et je renouvelle ma volonté d'avoir l'opportunité de collaborer avec chacun de vous dans l'avenir.

Je remercie également Vincent Nicomette, Slim Abdellatif, Éric Alata, Daniela Dragomirescu, Christophe Chassot et l'ensemble du corps enseignant du département DGEI de l'INSA de Toulouse, qui m'ont fait confiance et permis de faire de l'enseignement durant la préparation de cette thèse. Ces heures passées à préparer ou animer des cours m'ont permis de déterrer des enseignements enfouis dans mes (bons) souvenirs d'élève ingénieur.

Je ne peux pas ne pas mentionner tous mes amis qui m'ont soutenu, conseillé, aidé et distrait tout au long du périple qui m'a amené là j'en suis aujourd'hui : Adrien, Alioune, Anja, Arnaud, Denis, Erwan, Estelle, Farid, Guillaume, Houssen, Julien, Karim, Karine, Laetitia, Laurence, Laurie, Lidao, Livantsoa, Malik, Nicolas, Princy, Priscilia, Rémy, Shreya, Sophie, Stephen, Timothée et tous ceux que j'oublie forcément. Merci à tous pour leur soutien, leur humour et tous ces moments inoubliables passés ensemble.

Enfin, mes pensées vont bien entendu à l'ensemble de ma famille : mes parents, mes frères, mes oncles et mes tantes, mes cousins et mes cousines. Je leur dois mon envie d'aller toujours plus loin dans les activités que j'entreprends. Merci à eux pour leur soutien inconditionnel dans toutes les épreuves qui m'ont amené jusqu'ici. Je ne peux conclure sans remercier Jieyu, avec qui j'ai débuté cette aventure qu'est la thèse de doctorat, pour avoir été toujours à mes côtés et pour m'avoir soutenu en permanence.

« Pass on what you have learned, Luke. There is... another... Sky... walker. »

— Master Yoda, *Star Wars : Episode VI – Return of the Jedi*, 1983

« An invasion of armies can be resisted, but not an idea whose time has come. »

— Victor Hugo, *Histoire d'un crime*, 1852

Table des matières

Table des figures

Liste des tableaux

Introduction générale

1 Contexte et problématique

Aujourd'hui, les systèmes informatiques occupent une place prédominante dans les entreprises, dans les administrations et dans le quotidien des particuliers. Ce phénomène a été catalysé, entre autres, par l'essor de l'Internet qui séduit chaque jour de plus en plus d'internautes par les nombreux avantages et la diversité des services rendus accessibles. Ils peuvent ainsi bénéficier à moindre coût de moyens de communication rapides, partager des ressources de traitement et de stockage de grandes capacités (*Cloud Computing*), faciliter les échanges commerciaux et financiers (*e-Commerce, e-Banking*), fournir et utiliser de nombreux services en ligne (*e-Administration, e-Health, e-Learning, etc.*), participer à des communautés virtuelles et à des réseaux sociaux, se divertir (*e-Gaming, e-Television*, etc.) et, plus généralement, partager et accéder à l'information [Deswarte et Gambs 12]. Notre dépendance croissante aux systèmes informatiques dans divers aspects de la vie quotidienne et leur omniprésence soulèvent inévitablement des questions quant à leur sécurité et à la sécurité des informations qui leurs sont confiées.

En dépit des efforts conséquents qui ont été investis depuis un certain nombre d'années pour tenter d'endiguer les problèmes de sécurité, force est de constater que le nombre de vulnérabilités dans les systèmes informatiques et, de surcroît, les activités malveillantes qui essaient et qui réussissent à les exploiter, continuent régulièrement à se multiplier. Les données statistiques fournies par Kaspersky Lab [Namestnikov 12], spécialisé dans la sécurité des systèmes d'information, tendent à confirmer ce phénomène. Notamment, il dénombrait pas moins de 946 393 693 attaques réussies au travers de l'Internet en 2011 contre seulement 580 371 937 en 2010, représentant ainsi une progression d'environ 39%. Cet échec est justifié, en partie, par la complexité toujours croissante des systèmes informatiques actuels. En effet, pour réduire les coûts de développement, ces systèmes utilisent de plus en plus de composants sur étagère (ou COTS pour *Commercial Off-The-Shelf*), même pour des applications plus ou moins critiques. Ces composants peuvent être des matériels (microprocesseurs, *chipsets* [1], périphériques, etc.), des programmes de base (systèmes d'exploitation, outils de développement ou de configuration, etc.), des programmes génériques (systèmes de gestion de base de données, serveurs Web, etc.) ou dédiés à des fonctions plus spécifiques (pare-feux, détection d'intrusions, supervision, etc.). Les composants génériques, de par la multiplicité de leurs utilisations possibles, sont souvent beaucoup plus complexes que ce qui est vraiment utile pour un système particulier. Cette complexité les fragilise vis-à-vis de la sécurité. En effet, il est pratiquement impossible de les vérifier parfaitement, c'est-à-dire de garantir, d'une part, l'absence de bogues et, d'autre part, l'absence de fonctions cachées. Par conséquent, des attaquants (ou *pirates informatiques*) peuvent profiter de cette situation pour réussir à pénétrer dans ces systèmes et utiliser les res-

[1] Le *chipset* est un ensemble de puces électroniques chargé d'interconnecter les processeurs à d'autres composants matériels tels que les mémoires, les cartes graphiques, les cartes réseau, les contrôleurs de disque, etc.

sources matérielles et logicielles qui y sont associées.

Les attaquants font aujourd'hui de plus en plus preuve d'ingéniosité pour attaquer les systèmes informatiques : les malveillances peuvent cibler les programmes qui s'exécutent sur le système mais également le système d'exploitation lui-même et en particulier, son noyau. Corrompre le noyau d'un système d'exploitation est particulièrement intéressant du point de vue d'un attaquant car cela signifie corrompre potentiellement tous les programmes qui s'exécutent au-dessus de ce noyau, voire prendre complètement le contrôle du système. À cet effet, il dispose de divers vecteurs d'attaque. La plupart des scénarios d'attaque s'appuient sur des vecteurs d'attaque liés au logiciel s'exécutant sur le processeur principal : un attaquant peut, par exemple, utiliser des fonctionnalités trop permissives du système (le chargeur de modules noyau, les périphériques virtuels, etc.) ou exploiter des erreurs d'implémentations logicielles (les débordements de tampons ou d'entiers, les chaînes de format, etc.). Cette classe d'attaques est relativement bien connue à ce jour, et de nombreux mécanismes (les piles non-exécutables, la distribution aléatoire de l'espace d'adressage, la technique du canari, etc.) permettent de s'en protéger. Depuis quelques années, nous constatons que les scénarios d'attaque évoluent et deviennent chaque jour plus complexes. En effet, certaines attaques ne reposent plus exclusivement sur l'utilisation de composants logiciels, et nous observons de plus en plus d'attaques impliquant des composants matériels, tels que le *chipset*, les contrôleurs d'entrées-sorties (un contrôleur Ethernet, un contrôleur de clavier, etc.) ou les périphériques (un iPod avec une connectique FireWire, une souris USB, etc.). Celles-ci détournent des fonctionnalités légitimes du matériel, tels que les mécanismes entrées-sorties (accès direct à la mémoire, interruption, etc.) à différentes fins malveillantes (escalade de privilèges, fuite d'informations, déni de service, etc.).

Pour se protéger de ces vecteurs d'attaque multiples, nous constatons alors que les contre-mesures, qui étaient jusqu'à présent presqu'exclusivement logicielles, ne suffisent plus et qu'une bonne connaissance de la plate-forme matérielle devient de plus en plus importante pour une meilleure maîtrise de la sécurité d'un système informatique. Ainsi, il est aujourd'hui essentiel de connaître non seulement le fonctionnement des programmes, mais également de veiller au bon comportement de l'intégralité de la chaîne de traitement, y compris des composants matériels moins connus, tels que le *chipset*, les contrôleurs d'entrées-sorties ou les périphériques. Cette problématique constitue le principal fil conducteur de nos travaux.

2 Présentation de nos travaux

Les travaux décrits dans ce manuscrit de thèse se focalisent sur la sécurité en vie opérationnelle des systèmes informatiques de type COTS vis-à-vis d'attaques impliquant des composants matériels. Nous nous intéressons plus particulièrement aux attaques dites « par entrées-sorties », lesquelles se distinguent des attaques usuelles par les vecteurs d'attaque qu'elles emploient : ces actions malveillantes détournent les mécanismes d'entrées-sorties, c'est-à-dire les mécanismes de communication entre le processeur principal et les autres composants. Depuis un certain nombre d'années, nous constatons que ces attaques atypiques suscitent un engouement croissant, dépassant même pour certaines d'entre elles le stade de preuve de concept. Récemment, l'entreprise de solutions logicielles Qihoo 360 [QiHoo 360 11] a signalé l'apparition d'un maliciel reprenant les techniques d'attaque présentées dans la preuve de concept IceLord [IceLord 07] publiée quelques années auparavant sur le *Chinese Software Developer Network* (CSDN). Ce maliciel, initialement identifié comme étant le virus BMW [2] puis renommé

[2] BMW est l'acronyme de « BIOS MBR Windows » et ne désigne en aucun cas la marque du constructeur automobile.

par la suite Trojan.Mebromi [3], est capable de reprogrammer les BIOS du constructeur OEM (*Original Equipment Manufacturer*) Award Software afin d'y adjoindre une ROM d'extension malveillante. Celle-ci a pour fonction d'assurer, d'une part, sa persistance et, d'autre part, de camoufler ses activités malveillantes vis-à-vis de toutes solutions logicielles antivirales. C'est d'ailleurs une caractéristique importante des attaques par entrées-sorties : puisqu'en général, elles n'utilisent, pour leur mise en œuvre, ni le processeur ni les mémoires qui lui sont accessibles, elles sont extrêmement difficiles à détecter par des logiciels de sécurité exécutés par les processeurs (anti-virus, anti-spyware, anti-rootkit, etc.).

En dépit des nombreuses études sur le sujet, nous constatons que cette classe d'attaques est encore méconnue à ce jour et certaines interrogations restent sans réponse. Quelles sont les actions malveillantes qu'un attaquant peut espérer effectuer dans un système informatique s'il parvient, par exemple, à implanter une porte dérobée ou un cheval de Troie dans un composant matériel ? Une telle implantation peut être réalisée délibérément par le fabricant ou alors à son insu lors de la conception du composant matériel (en altérant le « compilateur » qui produit le masque de la puce), lors de sa fabrication (en altérant le masque de la puce) ou en vie opérationnelle (en altérant son *firmware*). Ce sont des menaces qui, aujourd'hui, sont d'autant plus plausibles que des équipements de réseau peuvent être conçus par des pays hostiles pour intégrer des fonctions cachées pour prendre le contrôle de réseaux adverses. Récemment, la commission de renseignement de l'U.S. House of Representatives a publié un rapport [Rogers et Ruppersberger 12] qui montre l'inquiétude grandissante du gouvernement américain sur les portes dérobées qui pourraient être présents dans les équipements fabriqués par des pays hostiles. Cette inquiétude a été partagée par de nombreux pays dont la France [Bockel 12]. Vis-à-vis de telles menaces, jusqu'à quel point les contre-mesures matérielles que l'on peut mettre en œuvre pour renforcer la sécurité des systèmes informatiques récents sont-elles efficaces ? Quels choix d'implémentation (pour les plates-formes matérielles) pourraient pallier les insuffisances des contre-mesures actuelles et réduire les risques liés à l'exploitation d'un tel vecteur d'attaque ? C'est à ces questions que tentent de répondre nos travaux.

Nous commençons ce manuscrit par un premier chapitre rappelant la terminologie et les concepts de base associés à la sécurité des systèmes informatiques. Ceci nous permet alors de présenter les attaques qui mettent en défaut la sécurité de ces systèmes et de discuter des façons différentes de les classer que nous retrouvons dans la littérature. Les classifications existantes ne mettent cependant pas suffisamment l'accent sur les caractéristiques des attaques que nous considérons pour nos travaux. Aussi, nous nous proposons de les préciser en décomposant les attaques en séquences d'actions élémentaires dont certaines sont malveillantes. L'objectif poursuivi dans cette caractérisation est d'identifier l'ensemble des attaques élémentaires sur lesquelles sont basées les attaques réelles, afin de mettre en évidence ensuite quelques contre-mesures que l'on peut mettre en œuvre pour s'en protéger.

Le deuxième chapitre détaille notre démarche pour établir un modèle d'attaques général pour les systèmes informatiques. À partir d'une représentation fonctionnelle d'un système informatique, nous construisons un graphe de flux de données entre les différents composants du système. En appliquant ensuite à ce graphe un ensemble de règles déduites des attaques élémentaires identifiées dans le chapitre précédent, nous déduisons les séquences d'attaques qui sont possibles dans un système informatique. Parmi celles-ci, nous retrouvons celles qui nous intéressent pour nos travaux : les attaques par entrées-sorties.

Dans un troisième chapitre, nous instancions le modèle d'attaques précédemment établi

[3] Une analyse approfondie du fonctionnement de ce maliciel a été publiée dans [Giuliani 11].

pour un système informatique typique. Avec le point de vue de l'attaquant, nous mettons en œuvre notre modèle au travers de quelques exemples d'attaques par entrées-sorties pour des systèmes informatiques organisés autour de l'architecture PC. Ces exemples présentent, entre autres, différentes façons d'exploiter le mécanisme de *bus mastering* présent dans la majorité des contrôleurs d'entrées-sorties sur étagère. Il s'agit d'une fonctionnalité matérielle légitime leur permettant de prendre temporairement le rôle de maître sur les bus d'entrées-sorties afin d'effectuer des transferts de données avec la mémoire centrale (accès direct à la mémoire) ou avec d'autres contrôleurs (accès pair-à-pair), déchargeant ainsi le processeur principal des transferts d'entrées-sorties. Ces attaques nous permettent alors d'introduire les contre-mesures matérielles qui ont été proposées (et pour certaines implémentées dans le *chipsets* récents), puis de discuter de certaines insuffisances que nous avons identifiées au sein de celles-ci.

Finalement, dans notre dernier chapitre, nous étudions de manière plus détaillée les attaques identifiées dans notre modèle. Dans cette approche, nous explorons l'idée d'appliquer aux composants matériels les techniques de recherche de vulnérabilités généralement utilisées pour les composants logiciels. Nous nous sommes plus particulièrement intéressés aux techniques de *fuzzing* appliquées au dénominateur commun à tous les composants matériels, à savoir les bus d'entrées-sorties. Afin de pallier les difficultés liées à l'utilisation d'un contrôleur d'entrées-sorties sur étagère pour injecter des fautes sur les bus d'entrées-sorties, nous avons mis au point notre propre contrôleur d'entrées-sorties en utilisant les technologies de logique programmable. Ce démonstrateur, que nous avons nommé *IronHide*, a été conçu dans l'optique d'être entièrement programmable et de pouvoir générer des requêtes quelconques, des requêtes valides mais aussi et surtout des requêtes invalides, sur les bus d'entrées-sorties.

Nous concluons ce manuscrit par une synthèse de nos contributions et présentons quelques perspectives à nos travaux. Parmi ces perspectives, nous projetons de raffiner davantage notre modèle d'attaques. Nous discutons également d'autres cas d'utilisation que nous avons envisagés pour le contrôleur d'entrées-sorties *IronHide*, à la fois comme un outil d'analyse d'attaques par entrées-sorties mais également comme un moyen de protection contre ces mêmes attaques.

Chapitre 1

Actions malveillantes impactant la sécurité des systèmes informatiques

Les enjeux de la sécurité des systèmes informatiques sont primordiaux, étant donné les pertes humaines, économiques et financières importantes que peut engendrer la compromission de ces systèmes. Comprendre et caractériser les attaques informatiques nous paraît alors essentiel afin d'en déduire un modèle sur lequel il soit possible de s'appuyer, d'une part, pour étudier ces attaques et éventuellement en identifier d'autres qui sont encore inconnues, et d'autre part, pour concevoir puis mettre en place des mécanismes visant à protéger le système contre ces attaques. Les travaux décrits dans ce manuscrit se concentrent plus particulièrement sur ce premier objectif. Dans ce chapitre, nous essayons d'identifier les éléments du système informatique sur lesquels reposent une attaque.

Dans notre démarche, nous repartons de la terminologie et des concepts de base associés à la sûreté de fonctionnement et à la sécurité des systèmes, les deux concepts étant étroitement liés (section 1.1). Cette introduction permet, entre autres, de préciser les trois propriétés – confidentialité, intégrité et disponibilité – sur lesquelles repose la sécurité d'un système. Nous nous intéressons ensuite aux actions qui cherchent à mettre en défaut ces propriétés pour un système particulier : le système informatique. La caractérisation de ces attaques nous a permis d'obtenir, en section 1.2 , une classification des actions malveillantes selon différents niveaux d'abstraction, sur laquelle reposent les chapitres suivants. Finalement, en section 1.3, nous portons une attention particulière aux méthodes ainsi qu'aux techniques visant à prévenir l'introduction de vulnérabilités et à les éliminer aussi bien lors du développement du système informatique que lors de son fonctionnement en vie opérationnelle.

1.1 Sécurité des systèmes informatiques

Avant de préciser la notion de sécurité des systèmes informatiques, il convient de définir au préalable ce que nous entendons par le terme *système*. Nous précisons ensuite la terminologie et les concepts fondamentaux liés à la sûreté de fonctionnement dans laquelle s'inscrit plus particulièrement la sécurité des systèmes. Finalement, nous rappelons la définition de la sécurité des systèmes informatiques telle qu'elle a été initialement décrite dans les *Information Technology Security Evaluation Criteria* [ITSEC 91] puis, par la suite, reprise dans les Critères Communs [Common Criteria 12], c'est-à-dire la sécurité comme la combinaison de trois propriétés : la confidentialité, l'intégrité et la disponibilité de l'information.

1.1.1 Concepts et terminologie des systèmes [Laprie 04]

Un *système* est une entité qui interagit avec d'autres entités, donc d'autres systèmes, y compris le matériel, le logiciel, les humains et le monde physique avec ses phénomènes naturels. Ces autres systèmes constituent l'*environnement* du système considéré. La *frontière* du système est la limite commune entre le système et son environnement.

La *fonction* d'un système est ce à quoi il est destiné. Elle est décrite par la spécification fonctionnelle. Le *comportement* d'un système est ce que le système fait pour accomplir sa fonction et est décrit par une séquence d'états. L'ensemble des états de traitement, de communication, de mémorisation, d'interconnexion et des conditions physiques constituent son *état total*.

La *structure* d'un système est ce qui lui permet de générer son comportement. D'un point de vue structurel, un système est constitué d'un ensemble de composants interconnectés en vue d'interagir. Un *composant* est un autre système, etc. La décomposition s'arrête lorsqu'un système est considéré comme étant un système atomique : aucune décomposition ultérieure n'est envisagée ou n'est envisageable, soit par nature, soit parce que dénuée d'intérêt.

Le *service* délivré par un système, dans son rôle de *fournisseur*, est son comportement tel que perçu par ses utilisateurs ; un *utilisateur* est un autre système qui reçoit un service du fournisseur. Un système peut être, séquentiellement ou simultanément, fournisseur et utilisateur d'un autre système, c'est-à-dire délivrer un service à cet autre système et en recevoir. La partie de la frontière du système où ont lieu les interactions avec ses utilisateurs est l'*interface de service*. La partie de l'état total du fournisseur qui est perceptible à l'interface de service est son *état externe* ; le reste est son *état interne*.

Il est à noter qu'un système accomplit généralement plusieurs fonctions et délivre plusieurs services. Les concepts de fonction et de service peuvent donc être vus comme constitués de *fonctions élémentaires* et de *services élémentaires*.

1.1.2 Sûreté de fonctionnement

La sûreté de fonctionnement fournit un cadre conceptuel intéressant pour situer la sécurité par rapport à d'autres propriétés des systèmes informatiques. En effet, depuis de nombreuses années, les spécialistes de la sûreté de fonctionnement ont développé une terminologie et des méthodes dont l'application à la sécurité peut être enrichissante. Nous rappelons, dans cette sous-section, les définitions et les concepts de base de la sûreté de fonctionnement directement adaptés de [Laprie *et al.* 96] et de [Laprie 04].

La *sûreté de fonctionnement* d'un système est la propriété qui permet à ses utilisateurs de placer une *confiance justifiée* dans le *service* que le système leur délivre [Avižienis *et al.* 04, Laprie 04]. Cette propriété englobe trois notions différentes (cf. figure 1.1) : ses *attributs*, les propriétés complémentaires qui la caractérisent ; ses *entraves*, les circonstances indésirables – mais non-inattendues – qui sont causes ou résultats de la non-sûreté de fonctionnement ; ses *moyens*, les méthodes et techniques qui cherchent à rendre un système capable d'accomplir correctement sa fonction et à donner confiance dans cette aptitude.

1.1.2.1 Attributs de la sûreté de fonctionnement

Les attributs de la sûreté de fonctionnement sont définis par diverses propriétés dont l'importance relative dépend de l'application et de l'environnement auxquels est destiné le système informatique considéré :

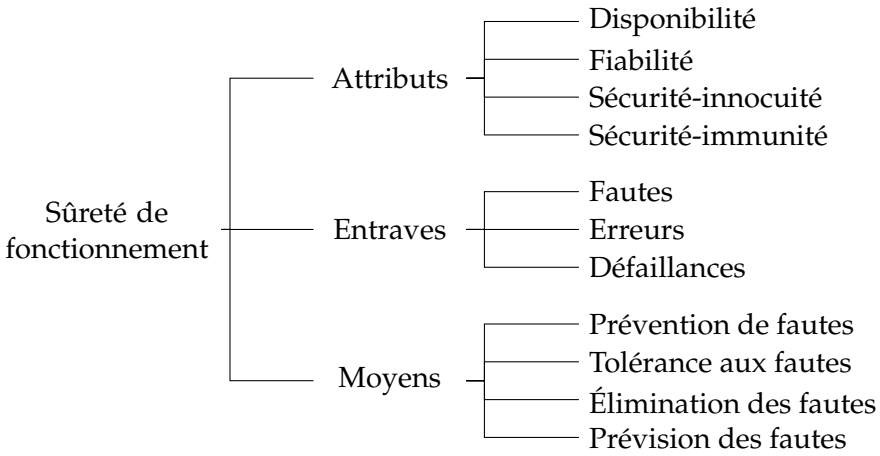

FIGURE 1.1 – Arbre de la sûreté de fonctionnement

— par rapport à la capacité du système à *être prêt à délivrer le service*, la sûreté de fonctionnement est perçue comme la *disponibilité* (en anglais, *availability*),
— par rapport à la *continuité de service*, la sûreté de fonctionnement est perçue comme la *fiabilité* (en anglais, *reliability*),
— par rapport à l'évitement de *conséquences catastrophiques sur l'environnement*, la sûreté de fonctionnement est perçue comme la *sécurité-innocuité* (en anglais, *safety*),
— par rapport à la *préservation de la confidentialité, de l'intégrité et de la disponibilité* des informations, la sûreté de fonctionnement est perçue comme la *sécurité-immunité* (en anglais, *security*).

1.1.2.2 Entraves à la sûreté de fonctionnement

Un service correct est délivré par un système lorsqu'il accomplit sa fonction. Une *défaillance du service*, souvent simplement dénommée *défaillance*, est un événement qui survient lorsque le service dévie de l'accomplissement de la fonction du système. Le service délivré étant une séquence d'états externes, une défaillance du service signifie qu'au moins un état externe dévie du service correct. La partie de l'état du système qui dévie du fonctionnement correct, autrement dit qui est anormale ou incorrecte, est une *erreur* ; une *faute* est la cause adjugée ou supposée d'une erreur, et peut être interne ou externe au système.

La relation de causalité entre fautes, erreurs et défaillances peut être exprimée comme suit. Une faute activée produit une erreur qui peut se propager dans un composant ou d'un composant à un autre, et est susceptible de provoquer une défaillance. La défaillance d'un composant cause une faute permanente ou temporaire interne pour le système qui le contient, tandis que la défaillance d'un système cause une faute permanente ou temporaire externe pour les systèmes avec lesquels il interagit. Ce processus de propagation est illustré sur la figure 1.2.

Deux principales classes de fautes sont à considérer dès lors que nous nous intéressons aux fautes malveillantes : les logiques malignes et les intrusions. Nous reprenons les définitions de ces termes telles qu'elles ont été introduites dans le projet MAFTIA (*Malicious- and Accidental-Fault Tolerance for Internet Applications*) [Adelsbach *et al.* 03]. Les *logiques malignes* sont des parties du système conçues pour provoquer des dégâts (les bombes logiques, les virus, etc.) ou pour faciliter des intrusions futures au travers de vulnérabilités créées volontairement telles que les portes dérobées. Les logiques malignes peuvent être introduites dès la création du sys-

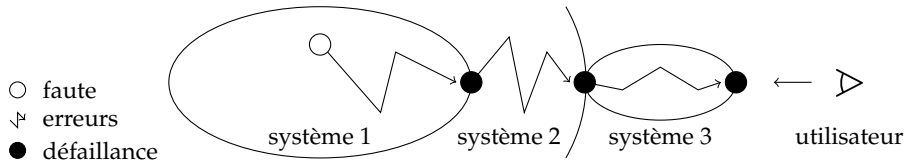

FIGURE 1.2 – Propagation des erreurs dans un système de systèmes

tème par un concepteur malveillant, ou en phase opérationnelle par l'installation d'un composant logiciel ou matériel contenant un cheval de Troie ou par une intrusion. La définition d'une intrusion est étroitement liée aux notions d'attaque et de vulnérabilité. Une *attaque* est une faute d'interaction malveillante visant à violer une ou plusieurs propriétés de sécurité. C'est une faute externe créée avec l'intention de nuire. Une *vulnérabilité* est une faute accidentelle ou intentionnelle (avec ou sans volonté de nuire) dans la spécification des besoins, la spécification fonctionnelle, la conception ou la configuration du système, ou dans la façon selon laquelle il est utilisé. La vulnérabilité peut être exploitée pour créer une intrusion. Une *intrusion* est une faute malveillante interne, mais d'origine externe, résultant d'une attaque qui a réussi à exploiter une vulnérabilité. Elle est susceptible de produire des erreurs pouvant provoquer une défaillance vis-à-vis de la sécurité, c'est-à-dire une violation de la politique de sécurité du système.

1.1.2.3 Moyens pour la sûreté de fonctionnement

Le développement d'un système sûr de fonctionnement passe par l'utilisation combinée d'un ensemble de méthodes qui sont réparties en quatre classes de moyens :

- la *prévention de fautes :* comment empêcher, par construction, l'occurrence ou l'introduction de fautes ; elle est principalement obtenue par des méthodes de spécification et de développement relevant de l'ingénierie des systèmes ;
- la *tolérance aux fautes :* comment fournir un service à même de remplir la fonction du système en dépit des fautes ; elle est mise en œuvre par la détection d'erreurs et le rétablissement du système ;
- l'*élimination des fautes :* comment réduire le nombre et la sévérité des fautes ; elle peut être réalisée pendant la phase de développement d'un système par vérification, diagnostic et correction, ou pendant sa phase opérationnelle par maintenance ;
- la *prévision des fautes :* comment estimer la présence, la création et les conséquences des fautes ; elle est effectuée par évaluation du comportement du système par rapport à l'occurrence des fautes, à leur activation et à leurs conséquences.

Ces moyens sont interdépendants et peuvent être groupés de différentes façons. Ainsi, la prévention de fautes et l'élimination des fautes peuvent être groupées dans l'*évitement des fautes*, c'est-à-dire comment tendre vers un système exempt de fautes, alors que la tolérance aux fautes et la prévision des fautes peuvent être groupées dans l'*acceptation des fautes*, à savoir comment vivre avec un système comportant des fautes. Un autre groupement consiste à associer la prévention de fautes et la tolérance aux fautes pour constituer la *fourniture de la sûreté de fonctionnement*, et donc l'aptitude à délivrer un service de confiance ; l'élimination des fautes et la prévision des fautes sont alors groupées en l'*analyse de la sûreté de fonctionnement*, destinée à obtenir une confiance justifiée dans le service, donc à s'assurer que les spécifications fonctionnelles et de sûreté de fonctionnement sont adéquates et que le système les satisfait.

1.1.3 Sécurité des systèmes informatiques [Deswarte 03]

Nous nous intéressons à présent à la sécurité-immunité que nous appellerons simplement sécurité lorsqu'il n'y aura pas d'ambigüité avec la sécurité-innocuité. Plutôt que de définir la sécurité des systèmes informatiques vis-à-vis de leur seule capacité à résister à des agressions externes physiques (incendie, inondation, bombes, etc.) ou logiques (erreurs de saisie, intrusions, piratages, logiques malicieuses, etc.), nous préférons, à l'instar des *Information Technology Security Evaluation Criteria* [ITSEC 91] et des Critères Communs [Common Criteria 12], considérer la sécurité comme la combinaison de trois propriétés [4] : la confidentialité, l'intégrité et la disponibilité. Il convient cependant de préciser que ces deux points de vue sont complémentaires. En effet, la sûreté de fonctionnement considère la sécurité comme l'association des trois propriétés (confidentialité, intégrité et disponibilité) vis-à-vis des actions autorisées (c'est-à-dire des services) alors que ces trois propriétés se rapportent à l'information dans le cadre des ITSEC. Le terme d'information doit être pris ici dans son sens le plus large, couvrant non seulement les données et les programmes, mais aussi les flots d'information, les traitements et la connaissance de l'existence de données, de programmes, de traitements et de communications. Cette notion d'information doit aller jusqu'à couvrir le système informatique lui-même dont parfois l'existence doit être tenue secrète (confidentialité). Pour être plus précis, nous distinguerons données et méta-données : les *données* correspondent à des informations identifiées et accessibles à un certain niveau d'abstraction, alors que les *méta-données* correspondent à des informations indirectes reliées aux données ou aux services. Bien évidemment, une méta-donnée à un certain niveau d'abstraction correspond à une donnée réelle à un niveau plus bas. À titre d'exemple, les identifiants internes (*uid*, *gid*, etc.) sont considérés comme des données ou comme des méta-données selon que l'on se place au niveau du système d'exploitation (données) ou au niveau d'une application (méta-données).

1.1.3.1 Confidentialité

La *confidentialité* est la propriété qu'une information ne soit pas révélée à des utilisateurs non autorisés à la connaître. Cela signifie que le système informatique doit empêcher les utilisateurs de lire une information confidentielle s'ils n'y sont pas autorisés, et empêcher les utilisateurs autorisés à lire une information de la divulguer à d'autres utilisateurs non autorisés. Cette seconde condition est souvent négligée car plus difficile à assurer.

Un incident survenu au *Massachusetts Institute of Technology* en 1965 illustre une atteinte typique contre la confidentialité. Lors de son discours pour le prix Alan Turing, F. J. Corbató [Corbató 91] raconta que cet incident a provoqué l'affichage de la liste de tous les utilisateurs et de leurs mots de passe en clair sur tous les terminaux du système à temps partagé du campus. Cette mésaventure était la conséquence d'une faute de conception et d'une erreur d'opérateur : le système était conçu pour n'avoir qu'un seul opérateur, et donc un seul tampon avait été mis à sa disposition pour mettre à jour des fichiers. Ce jour-là, deux opérateurs mettaient à jour en même temps deux fichiers, d'une part, le fichier de la liste des utilisateurs (contenant également leurs mots de passe en clair) pour y ajouter un nouvel utilisateur et, d'autre part, le fichier du message de bienvenue qui s'affiche sur les terminaux en début de session. Quand ce dernier fichier a été écrit, le tampon unique contenait la liste des utilisateurs qui a donc remplacé le message de bienvenue.

[4] Ce sens est généralement choisi par les spécialistes de l'audit de sécurité lorsqu'ils doivent évaluer les risques liés à l'informatique pour une entreprise.

1.1.3.2 Intégrité

L'*intégrité* est la propriété qu'une information ne soit pas altérée. Cela signifie que le système informatique doit empêcher une modification indue de l'information, c'est-à-dire une modification par des utilisateurs non autorisés ou une modification incorrecte par des utilisateurs autorisés, s'assurer qu'aucun utilisateur ne puisse empêcher la modification légitime de l'information, faire en sorte que l'information soit créée et, enfin, vérifier qu'elle est correcte. Il convient de préciser que le terme modification doit être entendu ici au sens large, comprenant à la fois la création d'une nouvelle information, la mise à jour d'une information existante, et la destruction d'une information.

Un virus informatique [Cohen 86, Filiol 03] est un exemple intéressant d'atteinte contre l'intégrité car une infection par ce type de parasites informatiques implique nécessairement une perte d'intégrité dans le système. Un virus est un segment de programme qui, lorsqu'il s'exécute, se reproduit en s'adjoignant à un autre programme. L'analogie avec les virus biologiques tient à leur comportement : un virus biologique ne peut se reproduire par lui-même ; pour se reproduire, il modifie le code génétique des cellules qu'il infecte pour que les cellules ainsi modifiées produisent des copies du virus ; de la même façon, un virus informatique ne peut être activé (et donc se reproduire) que par l'exécution d'un programme porteur du virus. La propagation du virus constitue alors une attaque contre l'intégrité des programmes, l'exécution du virus pouvant, par ailleurs, avoir d'autres effets qui peuvent être des attaques contre la confidentialité, la disponibilité et l'intégrité des données.

1.1.3.3 Disponibilité

La *disponibilité* est la propriété qu'une information soit accessible lorsqu'un utilisateur autorisé en a besoin. Cela signifie que le système informatique doit fournir l'accès à l'information pour que les utilisateurs autorisés puissent la lire ou la modifier, et faire en sorte qu'aucun utilisateur ne puisse empêcher les utilisateurs autorisés d'accéder à l'information. Pour reprendre l'exemple de l'incident survenu au *Massachusetts Institute of Technology*, si, par hasard, l'inverse s'était passé, à savoir la copie du message de bienvenue dans le fichier de la liste des utilisateurs, plus personne n'aurait pu se connecter au système, et c'eût été une atteinte grave à la disponibilité. Il convient de noter que la disponibilité implique l'intégrité, puisqu'il ne servirait à rien de rendre accessible une information fausse. La disponibilité implique également des contraintes plus ou moins précises sur le temps de réponse du service fourni par le système informatique. Une atteinte contre la disponibilité est souvent appelée *déni de service*.

1.1.3.4 Autres facettes de la sécurité

La sécurité peut parfois représenter d'autres caractéristiques, telles que l'*intimité* (pour traduire le terme anglo-saxon *privacy*), l'*authenticité*, l'*auditabilité*, la *pérennité*, l'*exclusivité*, etc. Ces propriétés de sécurité peuvent être exprimées par les propriétés de disponibilité, d'intégrité et de confidentialité appliquées à des données et à des méta-données.

Les propriétés de sécurité que nous venons de rappeler s'appliquent à la notion générale de système. Dans la suite de ce manuscrit, nous appliquons ces propriétés aux systèmes informatiques que nous définissons comme suit. Un *système informatique* est la composition de deux systèmes, l'un matériel et l'autre logiciel, organisés dans le but de remplir une fonction ou une

mission informatique dans un environnement donné. Un système informatique peut alors désigner un simple ordinateur, ou encore un réseau d'ordinateurs. Il est également possible de modéliser un réseau d'ordinateurs comme un système de systèmes informatiques. Les deux visions concordent avec notre définition, seule la granularité de la modélisation change. Aussi, pour plus de clarté, nous allons retenir cette seconde vision pour la suite de ce manuscrit et nous allons distinguer un ordinateur et un réseau d'ordinateurs.

1.2 Classification des attaques sur les systèmes informatiques

Tout acte sur un système dont l'intention est de nuire au moins à l'une des propriétés de sécurité (cf. sous-section 1.1.3) est qualifié de malveillant et constitue, de ce fait, une attaque sur ce système. Nous trouvons dans la littérature des manières différentes de classer les attaques [5]. Certaines taxonomies les organisent en fonction d'un unique critère. Parmi ces critères, les plus récurrents sont :

— la *cause* de l'attaque (utilisateur interne ou externe, intrus, etc.) [Anderson 80] ;
— le *mode* ou le *type* de l'attaque (virus, ver, écoute passive, déguisement, etc.) [Parker 75, Neumann et Parker 89] ;
— le *résultat* de l'attaque (divulgation, perturbation, etc.) [Shirey 94, Brinkley et Schell 95] ;
— la *vulnérabilité* exploitée par l'attaque ; plusieurs aspects peuvent alors être considérés : la *phase de création ou d'occurrence* de la vulnérabilité (lors de la conception, du développement, de l'exploitation du système, etc.), la *nature* de la vulnérabilité (concurrence, défaut de validation, etc.) [Abbott *et al.* 76, Bisbey et Hollingworth 78].

D'autres taxonomies considèrent les attaques comme un processus et les classent en fonction de plusieurs critères qui constituent alors des axes ou des dimensions de classification. Les critères retenus pour ces classifications dépendent de l'objectif poursuivi par les auteurs. Les taxonomies publiées dans [Landwehr *et al.* 94, Bishop 95, Aslam 95, Lindqvist et Jonsson 97, Howard 97, Krsul 98]reposent ainsi sur ce principe. Cependant, aucune d'elles ne nous satisfait pleinement pour nos travaux. Les classifications génériques ont, certes, le mérite d'englober la majorité (voire la totalité) des attaques mais sont trop peu précises sur la définition des caractéristiques des attaques. À l'opposé, les classifications plus spécifiques les décrivent de manière trop précise et ne couvrent potentiellement pas l'ensemble des attaques possibles, encore moins celles qui restent encore inconnues.

Aussi, dans cette section, nous présentons une classification des actions malveillantes impactant la sécurité des systèmes informatiques à laquelle nous avons abouti pour nos travaux. Celle-ci s'inspire de la logique des taxonomies d'attaques précitées et va nous servir, entre autres, de fondation pour le modèle d'attaques que nous détaillons dans le chapitre 2. Il convient cependant de noter que cette classification ne constitue pas encore, du moins dans son état actuel, une taxonomie qui puisse être une alternative à celles que nous avons précitées. Nous envisageons encore de la retravailler, de la raffiner sur certains aspects et de la valider en la confrontant à de nombreux exemples réels d'attaques afin qu'elle puisse être considérée en tant que telle.

Dans notre démarche, nous avons considéré une attaque comme une séquence d'actions, dont l'intention de certaines d'entre elles – les attaques élémentaires – est de nuire au moins à l'une des propriétés de disponibilité, d'intégrité ou de confidentialité du système cible ou d'un système ayant une quelconque relation avec lui (par exemple, un sur-système qui le contient,

[5] L'analyse des différentes taxonomies d'attaques ne fait pas l'objet de cette section et ne sera pas abordée dans ce manuscrit. Aussi, le lecteur intéressé pourra consulter les analyses publiées dans [Lough 01, Igure et Williams 08].

un système externe qui en dépend, etc.). De la définition que nous avons établie pour un système informatique, nous conjecturons que ces attaques élémentaires peuvent intervenir à plusieurs niveaux d'abstraction du système :

— soit sur les systèmes logiciels, plus précisément sur leur état ou leur structure ;
— soit sur l'état ou sur la structure des systèmes matériels, lesquels impactent les systèmes logiciels qui en dépendent ;
— soit sur les canaux de communication par lesquels il interagit avec d'autres systèmes ;
— soit sur son environnement, par la mise en œuvre d'attaques par des canaux auxiliaires.

Partant de cette hypothèse, nous avons ensuite identifié les vecteurs d'attaque possibles pour chaque niveau d'abstraction. La classification obtenue est représentée sur la figure 1.3. Il est opportun de remarquer, dès à présent, qu'une même technique d'attaque peut être utilisée pour nuire à différents niveaux d'abstraction du système informatique. Dans ce cas, cette technique peut être incluse dans plusieurs classes distinctes dans la mesure où elle porte sur des informations de nature différente. Par exemple, la majorité des vecteurs d'attaque considérés dans cette classification peuvent être mises en œuvre par de l'injection de fautes. Bien évidemment, il peut être envisagé d'affiner la classification que nous proposons afin de préciser toutes les techniques d'attaques qui sont susceptibles d'être utilisées pour chaque niveau d'abstraction, mais une telle granularité ne ferait qu'augmenter la complexité du modèle proposé au chapitre 2 sans réellement y apporter de bénéfice.

1.2.1 Attaques agissant au niveau des systèmes logiciels

Les systèmes logiciels constituent un premier niveau d'abstraction à partir duquel un attaquant peut mettre en défaut la sécurité d'un système informatique. Dans ce contexte, le terme système logiciel doit être pris dans son sens le plus général. Il désigne tous types de logiciels dans un système informatique, couvrant aussi bien les programmes d'application, le système d'exploitation et son noyau, que les logiciels implantés (en anglais, *firmware*) dans les composants matériels. Une attaque, à ce niveau d'abstraction, repose alors soit sur l'utilisation d'une fonctionnalité logicielle légitime du système (éventuellement accessible grâce à une erreur dans la configuration logicielle), soit sur l'exploitation d'une fonctionnalité logicielle vulnérable à des fins malveillantes. La présente sous-section discute de ces deux vecteurs d'attaque.

1.2.1.1 Abus d'une fonctionnalité logicielle légitime

Ce vecteur d'attaque se fonde sur le fait que l'accès à plusieurs fonctionnalités logicielles légitimes, potentiellement dangereuses pour la sécurité du système informatique, ne sont pas suffisamment restreintes par la politique de sécurité mise en œuvre par le système d'exploitation. Ces fonctionnalités logicielles peuvent alors servir des objectifs malveillants et porter éventuellement atteinte à la sécurité de composants du système situés à d'autres niveaux d'abstraction. Cette partie présente succinctement quelques unes d'entres elles. Bien que les exemples que nous énonçons concernent majoritairement les systèmes d'exploitation de type Unix, nous précisons que ces fonctionnalités logicielles (et donc, des vecteurs d'attaque similaires) existent dans d'autres types de systèmes d'exploitation (Windows, BSD, etc.).

Dans les systèmes d'exploitation de type Unix, l'appel système `ptrace()` [6] fait partie des nombreuses fonctionnalités logicielles légitimes que les attaquants détournent pour perpétrer des attaques. Un programme peut, au travers de cet appel système, accéder (en lecture et en

[6] Le nom de cet appel système est issu de la contraction de « *process trace* » (littéralement, trace d'un processus).

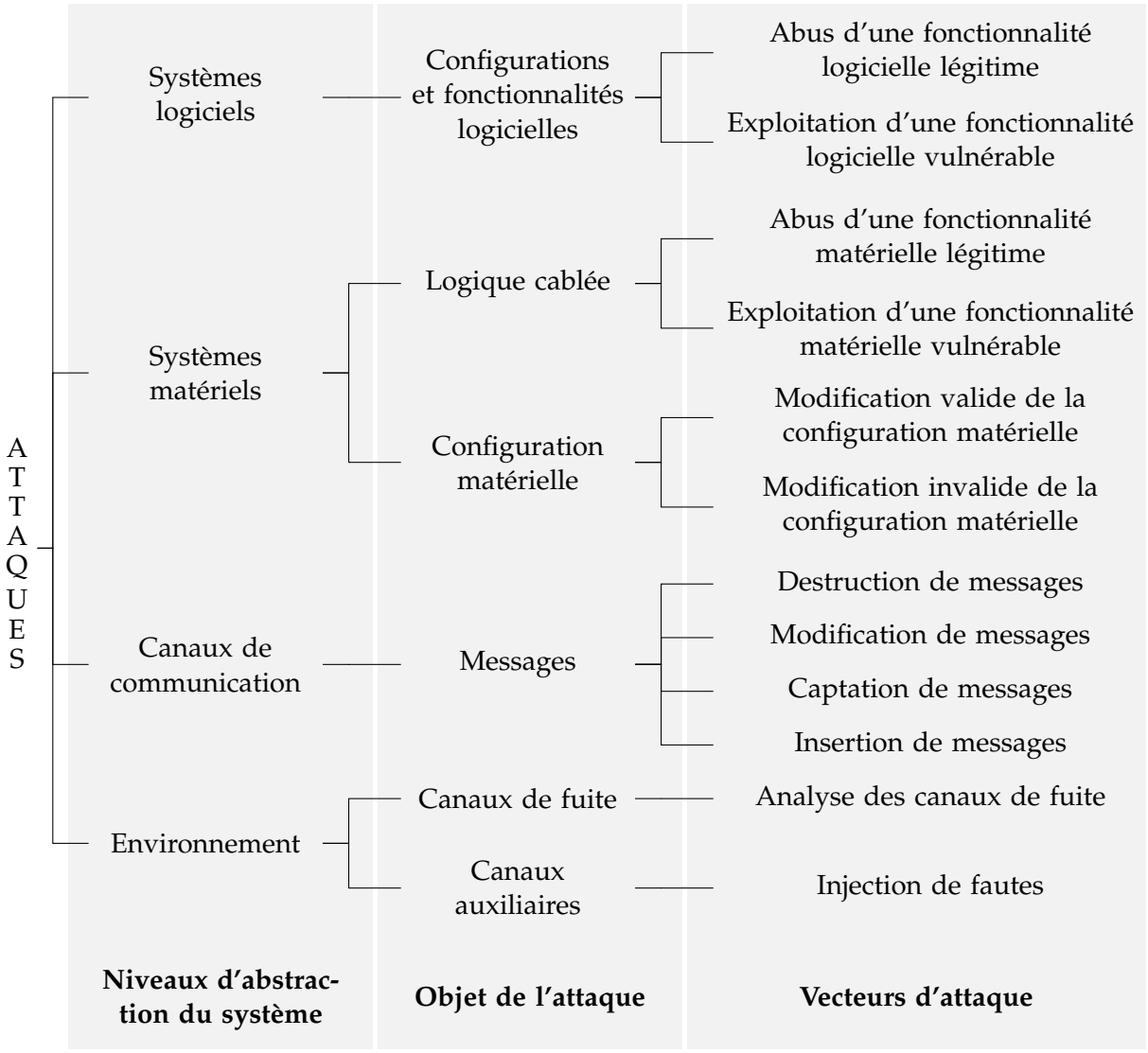

FIGURE 1.3 – Classification des attaques sur les systèmes informatiques

écriture) à l'intégralité de la mémoire (les registres du processeur inclus) d'un autre processus, c'est-à-dire un autre programme en cours d'exécution. Le fonctionnement de nombreux outils, tels que le debogueur `gdb`, les outils de trace de programmes `strace` et `ltrace` ainsi que plusieurs outils d'analyse structurelle de code reposent sur cet appel système. Malheureusement, il n'est pas nécessaire pour un attaquant d'acquérir des privilèges particuliers pour l'utiliser : les permissions requises sont identiques à celles nécessaires à l'envoi d'un signal à un autre processus. Il peut alors aisément mettre en œuvre cette technique contre un autre processus[7] et opérer des actions malveillantes diverses (injection de code, modification du contexte d'exécution, etc.). [Bareil 06] détaille, plusieurs exemples et codes-sources à l'appui, quelques unes de ces actions malveillantes. D'autres fonctionnalités logicielles, inhérentes à

[7] L'utilisation de cet appel système peut échouer sur certains processus. C'est le cas, en particulier, des processus issus de programmes exécutés avec les bits `suid` ou `sgid` positionnés. Pour les systèmes d'exploitation Linux, d'autres restrictions peuvent s'ajouter lorsque les *patchs* grsecurity [Spengler 12] ont été appliqués à son noyau.

l'éditeur de liens, peuvent également être détournées de leur utilisation légitime et impacter, par là-même, la sécurité d'autres programmes. Nous pensons, en particulier, au mécanisme de chargement de bibliothèques dynamiques de fonctions dont le comportement peut être modifié par la définition de variables d'environnement bien précises. Les variables d'environnement `LD_LIBRARY_PATH` et `LD_PRELOAD` précisent, par exemple, à l'éditeur de liens respectivement les emplacements du système où chercher les bibliothèques dynamiques de fonctions et celles qui doivent être chargées en priorité avant toutes les autres. Ces fonctionnalités logicielles sont particulièrement utiles lorsqu'il s'agit de développer et tester un système. Cependant, pour des raisons de sécurité évidentes, il est fortement recommandé de rendre ces fonctionnalités inopérantes dans tous les systèmes en production. Il suffirait alors pour un attaquant qu'une des bibliothèques dynamiques de fonctions chargées par ces variables d'environnement soit vulnérable ou qu'il puisse les modifier de façon à charger ses propres bibliothèques de fonctions pour porter atteinte à la sécurité du système informatique. Ces techniques d'attaques sont discutées dans le détail dans [halflife 97] et sont mises en œuvre dans le *rootkit* récent Jynx-Kit [ErrProne 12].

Le mécanisme de chargement de modules d'extension présent dans certains systèmes logiciels peut également être détourné à des fins d'attaques. Un module d'extension (en anglais, *plug-in*) désigne un composant logiciel qu'il est possible de charger dans un système logiciel de façon à étendre l'ensemble des services disponibles. Il s'adjoint alors à ce système en modifiant sa structure et son état au travers d'interfaces logicielles bien définies. Un attaquant peut alors complètement modifier le comportement du système logiciel dès lors que ces interfaces logicielles sont trop permissives. Un tel mécanisme est implémenté dans la majorité des noyaux de systèmes d'exploitation actuels afin de permettre le chargement de composants logiciels provenant de tiers tels que les pilotes de périphériques, les modules implémentant des fonctionnalités complémentaires, etc. Des personnes malintentionnées détournent naturellement cette fonctionnalité logicielle afin d'attaquer le noyau de système d'exploitation et, par là-même, l'ensemble des programmes qui en dépendent pour leur exécution. Dans les systèmes d'exploitation Linux, de nombreux *rootkits*, tels que [Creed 99, truff 03, TESO 04, styxˆ 12], s'insèrent par ce vecteur d'attaque. La majorité de ces maliciels reposent sur des techniques d'attaques publiées dans [Pragmatic et THC 99].

Une dernière fonctionnalité logicielle pouvant être détournée pour impacter la sécurité des systèmes matériels mérite d'être mentionnée. Les noyaux de système d'exploitation (pour être plus précis, les différents sous-systèmes logiciels qui pilotent chaque composant matériel) mettent souvent à disposition des programmes des interfaces logicielles de façon à faciliter les interactions entre les programmes et le matériel. Ces interfaces logicielles, que l'on nomme généralement des « périphériques virtuels » par abus de langage, permettent aux programmes de s'abstraire des spécificités de chaque composant matériel et réduisent d'autant la complexité de ces programmes. Les attaquants peuvent s'en servir comme d'un vecteur d'accès (en lecture, et parfois en écriture) au matériel. Les possibilités d'attaques dépendent alors des fonctionnalités implémentées dans ces périphériques virtuels. Dans les systèmes d'exploitation Linux, les périphériques virtuels `/dev/kmem`[8] et `/dev/mem`[9], lesquels offrent respectivement une abstraction de la mémoire virtuelle du noyau du système d'exploitation et de la mémoire physique, sont particulièrement usités par les maliciels. Les techniques d'attaques qui les mettent en œuvre sont discutées dans [Cesare 99, crazylord 02, Lineberry 09]. Heureusement, l'accès aux périphériques virtuels requiert des privilèges d'administrateur, tout comme le chargement

[8] Il est possible de désactiver ce périphérique virtuel depuis les versions de noyaux Linux 2.6.27 [van de Ven 08].
[9] Il est question de désactiver ce périphérique virtuel dans les prochaines versions du noyau Linux [ter Huurne 12].

de modules d'extension au sein du noyau de système d'exploitation. Cela limite fortement (mais n'empêche cependant pas) l'utilisation de ce vecteur d'attaque.

1.2.1.2 Exploitation d'une fonctionnalité logicielle vulnérable

La plupart des systèmes logiciels contiennent des vulnérabilités, c'est-à-dire des fautes de conception ou de configuration. Un attaquant peut alors exploiter ces vulnérabilités comme autant de vecteurs d'attaque pour effectuer différentes actions malveillantes dans un système informatique. Cette partie décrit quelques vulnérabilités courantes dans les systèmes logiciels. Le lecteur intéressé est invité à se référer à [Dowd *et al.* 06] pour une liste des vulnérabilités logicielles courantes illustrée par des exemples de programmes ayant été vulnérables.

Les débordements de tampon font partie des vulnérabilités logicielles les plus répandues et représentent près de 60% des annonces de sécurité des *Computer Emergency Response Team* (CERT) ces dernières années. Un débordement de tampon (en anglais, *buffer-overflow*) est une vulnérabilité dans laquelle les données copiées dans une zone de mémoire excèdent la taille allouée pour stocker ces données, écrasant, par là-même, les zones de mémoire contiguës qui peuvent contenir des données critiques. Ces débordements peuvent avoir lieu à deux emplacements de la mémoire : la pile ou le tas. Nous parlons alors plus spécifiquement de débordement dans la pile (en anglais, *stack-overflow*) ou de débordement dans le tas (en anglais, *heap-overflow*). L'exploitation d'une telle vulnérabilité implique nécessairement une modification de l'état du système logiciel : le débordement écrase alors soit des variables auxiliaires (variables locales, arguments de fonction, etc.), soit des variables de l'état d'exécution (adresse de retour d'une fonction, le pointeur de pile, etc.). Sur les systèmes informatiques respectant l'architecture de von Neumann [10] [von Neumann 45], dans laquelle les instructions et les données requises ou générées par le traitement sont stockées dans une même mémoire, un débordement de tampon peut conduire à la modification de la structure d'un programme en cours d'exécution. À cet effet, l'attaquant injecte dans la mémoire du processus un *shellcode*, autrement dit une chaîne de caractères malveillante contenant un code arbitraire qu'il souhaite exécuter, puis s'arrange (par exemple, en modifiant l'adresse de retour d'une fonction) pour détourner le chemin d'exécution du programme vers ce *shellcode* [Aleph One 96]. Le ver « *Inet Worm* » crée par Robert T. Morris Jr. [Eichin et Rochlis 89, Spafford 88] en novembre 1988 exploitait déjà ce vecteur d'attaque. Le ver envoyait aux démons `fingerd` une requête de 536 octets alors que seuls 512 octets étaient alloués dans la pile pour contenir cette requête. Le *shellcode*, injecté en mémoire par le débordement dans la pile, lui a ensuite permis d'obtenir une invite de commande (en anglais, *shell*) à distance.

Les débordements d'entiers forment une autre catégorie de vulnérabilités logicielles que nous retrouvons couramment dans les systèmes logiciels. Il s'agit d'une vulnérabilité qui survient lorsqu'un nombre à stocker en mémoire excède, par exemple, la plage de valeurs que le système informatique peut représenter. Cette vulnérabilité provient souvent soit d'un transtypage (par exemple, le système logiciel stocke un résultat dans une variable de trop petite taille), soit d'un calcul arithmétique (par exemple, la somme d'entiers positifs peut donner un nombre négatif en raison de la représentation des nombres en mémoire). Une telle vulnérabilité a été constatée par le passé sur le système logiciel de la fusée Ariane 5G V88/501, première de la série des fusées Ariane 5, qui embarquait la majorité du code produit pour la fusée Ariane 4. À cause de paramètres de vol différents, le code alors correct pour la fusée Ariane 4 ne l'était plus pour la fusée Ariane 5 et la conversion d'un réel sur 64 bits vers un entier signé sur 16

[10] Par opposition à l'architecture Harvard, dans laquelle les instructions et les données utilisées et générées par le traitement sont stockées dans des mémoires différentes qui sont connectées à deux bus distincts.

bits a provoqué une exception matérielle correspondant à un débordement arithmétique car le réel sur 64 bits contenait alors une valeur trop grande pour pouvoir être stockée dans un entier signé sur 16 bits. Or, pour des raisons de performance, la routine de traitement de cette exception avait justement été désactivée. Cette défaillance a provoqué, par la suite, une cascade de problèmes jusqu'à causer la défaillance des deux systèmes – primaire et secondaire – de guidage inertiel de la fusée qui s'est auto-détruite le 4 juin 1996, à peine 37 secondes après son lancement. Dans la plupart des cas, les débordements d'entiers ne sont pas exploitables. Cependant, ils peuvent mener à d'autres classes de vulnérabilités comme des débordements de tampons, par exemple, lorsque l'entier où se situe la vulnérabilité est utilisé pour allouer dynamiquement de la mémoire. Comme nous l'avons décrit précédemment, ces autres classes de vulnérabilités peuvent porter atteinte à la confidentialité, à l'intégrité et à la disponibilité des données et du service.

Les chaînes de format constituent une autre classe de vulnérabilités associée à l'utilisation inappropriée des familles de fonctions auxquelles appartiennent `printf()`, `err()` et `syslog()`. Le flux de caractères produit par ces familles de fonctions est construit à partir d'une chaîne de format. Celle-ci peut contenir des caractères ordinaires et des caractères spéciaux (appelés des « spécificateurs » de format) qui indiquent à la fonction comment manipuler les arguments qui lui sont passés en paramètres. Des problèmes de sécurité peuvent alors survenir lorsque l'attaquant contrôle cette chaîne de format. En faisant en sorte que les spécificateurs de format ne correspondent pas aux arguments fournis à la fonction, l'attaquant va le plus souvent provoquer une défaillance du système logiciel conduisant à un déni de service. Cependant, en construisant astucieusement la chaîne de format, il peut afficher le contenu de la pile, voire à faire exécuter au système logiciel des portions du code existant [Solar Designer 97, Nergal 01, Krahmer 05, Shacham 07].

1.2.2 Attaques agissant au niveau des systèmes matériels

Les systèmes matériels constituent un second niveau d'abstraction à partir duquel il est possible de nuire à la sécurité d'un système informatique. Bien qu'il soit plus simple pour un attaquant de cibler directement les systèmes logiciels, nous observons qu'actuellement de plus en plus d'attaques s'appuient sur les systèmes matériels pour impacter indirectement le système logiciel. Nous discernons deux raisons principales à cela. La première est que le fonctionnement des systèmes logiciels repose sur les systèmes matériels. Ainsi, corrompre un système matériel signifie potentiellement corrompre tous les systèmes logiciels qui en dépendent pour leur exécution. Le fait que les systèmes matériels sont souvent soumis à des restrictions moins drastiques que les systèmes logiciels constitue une seconde raison. En effet, au niveau du matériel, il n'existe plus de notion de privilèges, d'isolation de processus, etc. Une attaque au niveau des systèmes matériels agit alors soit sur les fonctionnalités matérielles qui ont été implémentées en logique câblée, soit sur leur configuration matérielle. Une telle attaque peut être mise en œuvre de différentes façons, présentées dans les trois sous-sections suivantes.

1.2.2.1 Abus d'une fonctionnalité matérielle légitime

Les systèmes matériels, étant les supports d'exécution des systèmes logiciels, ont automatiquement accès aux ressources utilisées par ces derniers. À l'instar du mécanisme d'accès direct à la mémoire dans les contrôleurs d'entrées-sorties [11], il est possible de détourner les fonctionnalités matérielles qui accèdent à ces ressources dans le but de perpétrer des attaques contre

[11] Les attaques qui exploitent le mécanisme d'accès direct à la mémoire seront discutées en détail dans le chapitre 3.

l'intégrité ou la confidentialité. Étant donné que ces accès sont autorisés et que leur mise en œuvre est relativement bien documentée dans les spécifications, il peut alors s'avérer difficile de distinguer un accès malveillant d'un accès légitime.

Un autre cas qu'il est important de considérer est celui des fonctionnalités matérielles non-documentées. Depuis de nombreuses années, on remarque, par exemple, que certaines instructions du processeur, qui ne semblent pas définies, ne provoquent pas d'exceptions matérielles lors de leur exécution [Collins 12b]. C'est le cas en particulier de l'*opcode* `0xd6` dans les processeurs Intel x86 qui, pendant longtemps a été considéré comme une instruction `nop` (une absence d'opération). En réalité, il correspondait à une opération qui consiste à mettre à 0 le registre `AL` du processeur lorsque le bit de retenue (en anglais, *carry*) du processeur est à 1 [Collins 12c]. Celui-ci n'a été documenté qu'à partir de la 6e génération de Pentium, soit à partir de l'année 2004. L'existence de ces fonctionnalités matérielles non-documentées est préoccupante et fait peser un risque important sur la sécurité du système. En effet, on voit sans peine l'avantage que pourrait obtenir un attaquant bien informé alors que les utilisateurs n'en soupçonnent même pas l'existence.

1.2.2.2 Exploitation d'une fonctionnalité matérielle vulnérable

De la même façon que les systèmes logiciels, les systèmes matériels peuvent contenir des fonctionnalités matérielles vulnérables. Toutefois, celles-ci sont moins courantes que les vulnérabilités logicielles et, dans la majorité des cas, ne sont pas exploitables. Pour se convaincre de l'existence de telles vulnérabilités matérielles, il suffit de remarquer que les principaux constructeurs de processeurs compatibles x86 publient, de façon régulière, les bogues matériels dans leurs composants. Ces bogues, pour lesquels une liste est disponible [Collins 12a], sont relativement nombreux et certains ne seront sans doute jamais corrigés. En particulier, le bogue matériel découvert dans les processeurs Pentium en décembre 1997 [Collins 97d] aurait pu être utilisé pour perpétrer des attaques contre la disponibilité. En effet, lorsqu'un processeur Pentium exécutait la séquence d'instructions `lock cmpxchg8b eax` (correspondant à la séquence d'octets `0xF00FC7C8`), celui-ci se bloquait aussitôt. L'instruction `cmpxchg8b eax` n'étant pas valide, l'exécution de celle-ci déclenchait une exception matérielle. Cette instruction préfixée par l'instruction `lock` formait également une autre instruction invalide et provoquait aussi une exception matérielle. À cause d'une inversion de priorité dans le traitement de ces deux exceptions matérielles, le processeur entrait dans un état d'interblocage. Une personne malintentionnée pouvait profiter de ce bogue matériel pour provoquer des déni de service simplement en exécutant un programme contenant cette séquence d'instructions. Il n'est pas clair que ce bogue matériel ait été corrigé sur les processeurs actuels. La plupart des systèmes d'exploitation masquent cependant ce problème par une configuration particulière du processeur. Il est intéressant de remarquer que tous les bogues matériels ne conduisent pas nécessairement à un déni de service. Dans certains cas, une escalade de privilèges [Wojtczuk et Rutkowska 11b, Gruskovnjak 12] est possible lorsque l'attaque est menée habilement.

1.2.2.3 Modification de la configuration matérielle

Le fonctionnement d'un système matériel repose en partie sur sa configuration matérielle. Un attaquant peut chercher à altérer la configuration d'un système matériel de façon à reconfigurer ou à désactiver certaines fonctionnalités matérielles. Par exemple, la plupart des systèmes d'exploitation actuels configurent les zones de la mémoire centrale correspondant à des don-

nées comme non-exécutables dans le but de rendre plus difficile l'exploitation des vulnérabilités logicielles. Pour faire en sorte que ces zones de données soient à nouveau exécutables, une attaque simple peut être envisagée. Elle consiste à positionner les attributs des pages contenant ces zones de données à une valeur appropriée. Ces attributs des pages configurent le contrôle d'accès effectué par l'unité de gestion mémoire (*Memory Management Unit*, ou MMU) dans le processeur. En occurrence, pour cet exemple précis, l'attribut `NX` (pour *Never-eXecute*) de chacune de ces pages doit être mis à zéro. Dans la majorité des cas, un attaquant va altérer la configuration matérielle de manière valide. Toutefois, il arrive parfois qu'un attaquant expérimente des modifications invalides de façon à perturber des fonctionnalités matérielles, à révéler des vulnérabilités matérielles [Rutkowska et Wojtczuk 08, Wojtczuk et Rutkowska 11b] ou à influer sur (voire exploiter) un système logiciel au moyen des valeurs contenues dans la configuration matérielle [Wojtczuk *et al.* 09]. Il peut être intéressant de positionner, par exemple, les champs réservés pour un usage futur, qui sont généralement mis à zéro, à des valeurs aléatoires.

1.2.3 Attaques agissant au niveau des canaux de communication

Les canaux de communication constituent un autre niveau d'abstraction à partir duquel un attaquant peut mettre en défaut la sécurité d'un système informatique. La notion de *canal de communication* désigne ici tout type de médium de transmission d'information dont le rôle est d'acheminer des messages entre des systèmes qui interagissent, et couvre aussi bien les canaux de communication physiques que les canaux de communication logiques (par exemple, les mémoires partagées, les fichiers partagés, etc.). Une attaque, à ce niveau d'abstraction, repose alors sur des vecteurs d'attaque liés aux messages échangés entre les systèmes logiciels ou matériels qui interagissent. Nous distinguons alors quatre vecteurs d'attaque possibles qui nécessitent pour l'attaquant un accès au canal de transmission : la destruction, la modification, la captation et l'insertion de messages, ces messages pouvant être soit cohérents, soit incohérents par rapport au protocole de communication. La suite de cette sous-section présente de façon succincte ces vecteurs d'attaque.

1.2.3.1 Destruction de messages

Ce vecteur d'attaque est probablement le plus simple à mettre en œuvre. Il nécessite que l'attaquant obtienne un accès en émission sur le canal de communication. Une fois cet accès acquis, il peut chercher à détruire les messages échangés entre les systèmes qui interagissent et perturber ainsi les communications, voire provoquer une atteinte contre la disponibilité. Une manière simple (et efficace) pour atteindre cet objectif revient à ne pas respecter les règles d'accès au médium et de le parasiter en continu de façon à empêcher les systèmes de communiquer. Cette technique d'attaque s'apparente au *brouillage* (en anglais, *jamming*). Lorsque les messages échangés transitent par un composant sous le contrôle de l'attaquant (par exemple, une passerelle compromise), une autre façon de détruire les messages consiste à les bloquer au niveau de ce composant. Dans ce cas, nous parlons plutôt d'interruption de messages.

1.2.3.2 Modification de messages

Ce vecteur d'attaque consiste à modifier les messages échangés entre les systèmes qui communiquent. Sa mise en œuvre nécessite, par conséquent, un accès en écoute et en émission sur le canal de communication. La modification des messages est opérée soit directement sur les

canaux de communication soit à l'aide d'un système intermédiaire, sous le contrôle de l'attaquant, par lequel transitent les échanges. Les objectifs de l'attaquant guident généralement la nature des modifications apportées. Une modification incohérente va impacter, dans la plupart des cas, la disponibilité et va conduire à un déni de service à moins qu'un mécanisme de tolérance aux fautes efficace contre ce type d'incohérence ne soit mis en œuvre. En revanche, une modification cohérente peut conduire à des attaques différentes. Une attaque courante est le déguisement (en anglais, *masquerade*) qui consiste à tromper les mécanismes d'authentification pour se faire passer pour un utilisateur autorisé, de façon à obtenir des droits d'accès illégitimes et ainsi compromettre la confidentialité, l'intégrité ou la disponibilité. Un cas particulier de déguisement consiste pour un attaquant à « forger » de fausses adresses d'origine pour les messages qu'il émet. Dans ce cas, nous parlons alors plus particulièrement de *spoofing* (littéralement, parodie) d'adresse. Ce déguisement est, par exemple, très facile à réaliser sur le réseau Internet, puisqu'il n'y a pas d'authentification sur les adresses IP.

1.2.3.3 Captation de messages

Ce vecteur d'attaque consiste à effectuer un accès, sans modification, aux messages qui sont générés, transmis ou stockés sur les systèmes matériels et logiciels vulnérables. Il cible plus particulièrement les propriétés de confidentialité. Les attaques par captation de messages sont très souvent difficiles à détecter. En effet, étant entièrement passives, elles ne produisent pas d'effets observables sur les canaux de communication. Il existe de nombreuses variantes de captation de messages (plus ou moins) passives, souvent baptisées de noms imagés : le *sniffing* (action de renifler), le *snooping* (action de fouiner), l'*eavesdropping* (action d'écouter aux portes), *wire-tapping* (branchement sur une liaison filaire, installation d'une « bretelle »). Les techniques de captation de messages se sont largement développées avec la connexion de nombreux réseaux locaux sur l'Internet. Une fois qu'un intrus a pris le contrôle d'une machine d'un réseau local, il peut installer un *sniffer*, qui est un logiciel capable de configurer la connexion réseau de cette machine (par exemple, le contrôleur Ethernet ou le contrôleur de réseau sans fil) en mode d'écoute (en anglais, *promiscuous mode*) puis analyser les paquets qui circulent sur le réseau, en particulier pour récupérer les noms d'utilisateurs et les mots de passe, parfois transmis en clair lors des phases de connexion d'utilisateurs. Il convient de remarquer que les informations (utiles) dans les messages ne sont pas toujours directement accessibles. Elles peuvent, par exemple, être dissimulées dans un ou plusieurs messages à l'aide de techniques de stéganographie.

1.2.3.4 Insertion de messages

Ce vecteur d'attaque consiste à insérer des messages dans un canal de communication. La mise en œuvre d'un tel vecteur d'attaque nécessite alors un accès au canal de communication en émission. Un attaquant peut, en particulier, rejouer des séquences de messages qu'il aura captées précédemment, par exemple pour se faire passer pour un autre utilisateur (répétition d'une séquence d'authentification) ou pour faire exécuter plusieurs fois une même action (répétition d'un transfert électronique de fonds), etc. Lorsque l'insertion consiste à émettre des messages, valides ou invalides, en très grand nombre, il s'agit plutôt d'une attaque contre la disponibilité qui peut aller jusqu'au déni de service. Nous parlons alors d'inondation (en anglais, *flooding*). Un cas particulier d'inondation est l'éblouissement, qui consiste à saturer les canaux de communication par des messages incohérents de façon à mettre temporairement hors ligne un ou plusieurs systèmes (par exemple, des systèmes de détection d'intrusions) de

façon à masquer, à ces systèmes, d'autres activités malveillantes (par exemple, l'installation d'une porte dérobée).

Il convient de remarquer qu'il est possible de définir des techniques d'attaques plus complexes à partir des quatre vecteurs d'attaque que nous venons de présenter. À titre d'exemple, une attaque de type *man-in-the-middle* (littéralement, l'homme au milieu), laquelle consiste à intercepter une communication entre deux systèmes puis à falsifier les échanges afin de se faire passer pour l'une des parties, peut être définie par la combinaison d'une captation et plusieurs insertions de messages. Les premières insertions permettent à l'attaquant de se faire passer pour l'une des parties et d'autres insertions sont nécessaires pour relayer les messages captés.

1.2.4 Attaques agissant au niveau des canaux auxiliaires

L'environnement d'un système informatique constitue un dernier niveau d'abstraction à partir duquel un attaquant peut porter atteinte à la sécurité de ce système. En particulier, un attaquant peut agir au niveau des canaux auxiliaires, c'est-à-dire les canaux autres que ceux généralement utilisés pour la transmission d'information. Une attaque, à ce niveau d'abstraction, peut alors consister à analyser les canaux de fuite ou à injecter des fautes dans le système informatique via les canaux auxiliaires. La suite de cette sous-section explore ces vecteurs d'attaque. Toutefois, notre intention n'est pas de faire un examen exhaustif des techniques existantes, mais plutôt de mettre en évidence la philosophie qui sous-tend les principales attaques. Aussi, seuls les canaux auxiliaires les plus usités seront évoqués ici.

1.2.4.1 Analyse des canaux de fuite

Ce vecteur d'attaque repose sur l'existence d'une corrélation entre des mesures physiques (consommation d'énergie, temps de calcul, rayonnement électromagnétique, etc.) et les traitements effectués par un système. Cette corrélation est parfois utilisée afin d'extraire des informations du système analysé et constitue, par conséquent, un *canal de fuite*. Les attaques sur ces canaux de fuite se déroulent généralement en deux temps. Dans une première phase dite d'interaction, l'attaquant mesure une ou plusieurs caractéristiques physiques du système qu'il souhaite attaquer. Au cours d'une seconde phase, appelée phase d'exploitation, l'attaquant analyse ces observations afin d'inférer l'information recherchée. Dans le cas de terminaux cryptographiques, cette information peut être, par exemple, une clé secrète. Plusieurs caractéristiques physiques peuvent être exploitées pour perpétrer ce type d'attaques.

Attaque par analyse temporelle. Dès lors qu'un attaquant connaît l'algorithme implémenté dans un système, les variations du temps d'exécution d'un traitement peuvent lui fournir de précieux renseignements sur les paramètres qui sont impliqués. Nous parlons alors d'attaques par analyse temporelle (en anglais, *timing attack*). Une telle attaque a été montrée contre le protocole *Secure Shell* (SSH) par [Song *et al.* 01] lors de la 10^{e} édition de l'*Unix Symposium Security*. En appliquant une analyse temporelle sur les paquets correspondant aux frappes de clavier, ils ont montré qu'il était possible d'inférer une quantité d'informations suffisante sur le contenu d'une communication chiffrée et localiser celles correspondant à la saisie d'un mot de passe. Ils ont également constaté qu'une simple analyse de trafic sur les paquets réseau révélait la longueur des données échangées (et donc la longueur des noms d'utilisateurs et des mots de passe associés). En recoupant ces informations, Song *et al.* ont été capables de réduire de moitié l'effort nécessaire pour retrouver le mot de passe d'un utilisateur.

Attaque par analyse de la consommation électrique. La consommation électrique d'un système électronique varie en fonction des traitements qu'il effectue. Dans le cas d'un processeur, par exemple, celle-ci est corrélée à l'instruction exécutée (voire aux instructions suivantes dans le cas d'un processeur avec *pipeline*), à ses opérandes (registres, adresses, valeurs, etc.) et aux valeurs des cellules mémoire et des bus. Un attaquant peut alors déterminer précisément les opérations effectuées en analysant cette grandeur physique. Nous parlons alors d'attaques par analyse de la consommation électrique (en anglais, *power attack*). Celles-ci se divisent en deux catégories : *Simple Power Attack* (SPA) et *Differential Power Attack* (DPA) [Kocher *et al.* 99]. La différence entre ces techniques réside dans le fait que les SPA reposent sur une simple analyse de consommation électrique alors que les DPA nécessitent de soumettre plusieurs exécutions similaires (par exemple, en changeant un bit dans le message à chiffrer) et à analyser les différences entres les traces de consommation.

Attaque par analyse du rayonnement électromagnétique. Dans un circuit électronique, le déplacement de charges électriques produit un champ électromagnétique. Par là-même, les variations du courant électrique dans un système électronique s'accompagnent d'une variation de son champ électromagnétique. Les deux grandeurs physiques étant étroitement liées, les techniques d'analyse que nous avons détaillées précédemment pour la consommation électrique peuvent être appliquées indifféremment au rayonnement électromagnétique. Dans la littérature, ce type de rayonnement est parfois appelé « effet van Eck », du nom d'un chercheur des PTT néerlandais qui a montré qu'avec un matériel de faible coût, il était possible d'afficher sur un écran de télévision le contenu de l'écran d'un terminal distant de plusieurs centaines de mètres [van Eck 85]. Une attaque similaire sur des claviers d'ordinateur a été récemment présentée par [Vuagnoux et Pasini 09]. En mesurant les ondes électromagnétiques émises par les claviers, ils ont réussi à retrouver chaque touche saisie jusqu'à une distance de 20 mètres. Chaque touche du clavier avait, en effet, une signature électromagnétique caractéristique.

Attaque par analyse acoustique. Les composants mécaniques et électroniques émettent un bruit dont la fréquence et l'intensité varient en fonction des opérations qu'ils effectuent[12]. Il est alors possible de reconstituer fidèlement le traitement effectué à partir des bruits mesurés. Peter Wright, alors scientifique pour l'*UK Government Communications Headquarters* (GCHQ) en 1965, raconte dans son autobiographie [Wright 87] que le MI5, service de renseignements britannique, tenta de casser le chiffrement utilisé par l'ambassade égyptienne à Londres durant la crise du canal de Suez. Limités par les capacités de leurs calculateurs, P. Wright suggéra de placer un microphone près des dispositifs (mécaniques) de chiffrement utilisés par les Égyptiens dans le but de capter le bruit produit. Avec le concours des services postaux britanniques, qui étaient à l'époque en charge des lignes téléphoniques, ils ont délibérément provoqué des dysfonctionnements dans les équipements téléphoniques de l'ambassade égyptienne. Ils ont ensuite envoyé des agents du MI5 dans l'ambassade, déguisés en agents de maintenance de la compagnie téléphonique, qui ont réussi à poser les microphones grâce à ce subterfuge. L'opération, du nom de code ENGULF, s'est ensuite couronnée de succès car en écoutant les clics produits par les dispositifs de chiffrement, ils ont pu déduire la position de 2 ou 3 de ses rotors. Cette information supplémentaire a réduit l'effort de cryptanalyse, et le MI5 a pu espionner, par ce biais, les communications de l'ambassade égyptienne à Londres pendant plusieurs années.

[12] Dans les composants électroniques, ce bruit est dû en particulier aux condensateurs qui émettent un claquement lorsqu'ils se chargent ou se déchargent.

Au prime abord, cette technique d'attaque peut sembler anecdotique et dépassée avec les technologies actuelles. Elle ne doit cependant pas être négligée. En 2004, [Shamir et Tromer 04] ont démontré qu'il est possible de s'aider des variations de bruits ultrasoniques émanant de composants électroniques dans le processeur ou dans la carte-mère pour identifier, à l'aide d'une analyse temporelle, les opérations ou les accès à la mémoire centrale effectués par le processeur.

1.2.4.2 Injection de fautes par les canaux auxiliaires.

Un attaquant peut ne pas se contenter d'analyser les canaux de fuite et utiliser ces mêmes canaux pour perturber le fonctionnement de ces systèmes et ainsi porter atteinte à leur sécurité (par exemple, reconstitution d'un secret, validation d'une fausse signature, etc.). Nous parlons alors d'injection de fautes par les canaux auxiliaires. La plupart des injections de fautes sont faites de façon plus ou moins aléatoire. L'objectif poursuivi est, en premier lieu, d'essayer de ne générer qu'une modification ponctuelle et de comparer les exécutions du système sans injection de fautes et avec injection de fautes. Les premières fautes injectées dans les systèmes informatiques (dans les années 1970) étaient accidentelles [13] : les particules radioactives alpha produites par des éléments présents dans les matériaux d'emballage causaient des fautes dans les puces. Ces particules créaient une charge dans les parties de la puce qui étaient critiques provoquant ainsi des inversions de bits (en anglais, *bit-flips*). Même si ces éléments n'étaient présents qu'en petite quantité, leur concentration était suffisante pour influer sur le comportement de la puce [May et Woods 79]. Aujourd'hui, il existe de nombreuses façons de perturber le fonctionnement d'un système informatique en agissant sur son environnement. Nous présentons dans la suite plusieurs de ces techniques.

Modification de la tension électrique. Les circuits électroniques sont tous conçus pour fonctionner pour une tension d'alimentation constante et bien définie. Des variations soudaines de cette tension (en anglais, *voltage spikes*) peuvent perturber son fonctionnement. Dans les systèmes informatiques, ce vecteur d'attaque est utilisé pour induire des erreurs dans les mémoires ou pour modifier les chemins d'exécution du code qu'il exécute (par exemple, modification d'un compteur de boucle, des condition de sauts, etc.). La littérature fait état de nombreuses attaques utilisant cette technique sur des cartes à puces [Boneh *et al.* 97b, Biham et Shamir 97, Boneh *et al.* 01a, Aumüller *et al.* 03, Yen *et al.* 03].

Modification de la fréquence d'horloge. De la même façon que pour la tension électrique, les circuits électroniques fonctionnent correctement uniquement dans une plage de fréquences d'horloge définie par son constructeur. L'utilisation temporaire d'une fréquence anormalement élevée ou basse peut entraîner des erreurs dans le traitement effectué. Nous parlons alors d'attaques par dérive de la fréquence d'horloge (en anglais, *frequency glitch*). Dans le cas d'un processeur, l'injection d'une anomalie dans sa fréquence d'horloge de référence peut l'amener à omettre plusieurs instructions et à complètement changer son comportement. Comme la compréhension de ce phénomène physique ne fait pas l'objet de cette partie, le lecteur intéressé est invité à consulter [Anderson et Kuhn 98] pour plus de détails. Il est intéressant de noter que cette technique, relativement simple dans son principe, a permis de contourner les mécanismes de sécurité d'au moins deux consoles de jeux vidéos réputées pour être difficilement

[13] Ce type de fautes accidentelles est de plus en plus fréquent, en particulier dans les systèmes aéronautiques et spatiaux : avec la réduction de taille des circuits intégrés, est aussi réduite l'énergie nécessaire pour perturber leur fonctionnement. Nous parlons généralement de *Single Event Upset* (SEU).

violables. En août 2007, des *hackers* ont remarqué sur la Xbox 360 que l'envoi d'une impulsion de remise à zéro à un processeur sous-cadencé ne le réinitialisait pas et changeait la façon dont le code était interprété. L'exploitation habile de ce comportement du processeur au moment où les condensats sont comparés lors de la vérification de la chaîne de confiance avait permis aux *hackers* d'exécuter du code arbitraire et, en particulier, de charger une version antérieure du noyau de système d'exploitation qui était vulnérable et pour laquelle des attaques avaient déjà été développées. En janvier 2010, une attaque utilisant quasiment le même mode opératoire a été menée sur la Playstation 3 [EurAsia 10] par George Francis Hotz, connu sous le pseudonyme GeoHot. Pour cette attaque, GeoHot avait programmé une puce FPGA pour envoyer une impulsion de 40 ns, à la pression d'un bouton, sur une ligne du contrôleur de bus mémoire de sa Playstation 3. Cette impulsion avait pour but d'empêcher le processeur d'écrire en mémoire (en mémoire cache pour être plus précis), et donc de donner une vision faussée de la mémoire à l'hyperviseur de la Playstation 3 : celui-ci croyait alors qu'une zone de mémoire était désallouée alors que l'attaquant y avait encore accès en lecture et en écriture, car la configuration de l'unité de gestion mémoire n'avait pas pu être modifiée. En faisant en sorte que l'hyperviseur utilise cette zone de mémoire pour l'une de ses structures internes, il a pu détourner son flot d'exécution et y installer deux appels systèmes pour permettre à du code moins privilégié de lire et d'écrire à une adresse mémoire quelconque.

Modification de la température. Les constructeurs de composants électroniques définissent également les températures extrêmes (minimales et maximales) jusqu'auxquelles les composants fonctionnent correctement. En faisant fonctionner un composant en dehors de cette plage de température nominale, il est possible de perturber son fonctionnement. Sur les cartes à puce (en anglais, *smart-cards*), il est possible d'observer deux effets lorsque ce vecteur d'attaque est utilisé : la modification aléatoire des mémoires RAM dues à l'échauffement des composants et une incohérence entre ce qui est écrit et ce qui est lu de la mémoire. Ce dernier phénomène est dû en particulier au fait que les températures extrêmes de fonctionnement des mémoires différent pour les opérations de lecture et d'écriture.

Injection de faute par rayonnement. Récemment, [Skorobogatov et Anderson 03] ont observé qu'éclairer un transistor pouvait le rendre conducteur, induisant alors une faute. Ainsi, en appliquant une source de lumière intense (produite à l'aide du flash d'un appareil photo puis concentrée à l'aide d'un microscope), ils ont réussi à modifier les valeurs de bits choisis dans une mémoire de type SRAM. Il convient de noter qu'il est souvent nécessaire que la puce soit au préalable décapsulée afin que le rayon lumineux atteigne la zone d'intérêt. Un tel vecteur d'attaque est particulièrement intéressant pour changer les bits relatifs à une instruction de saut afin de faire prendre au processeur la mauvaise branche (et valider par exemple un calcul faux). Les rayonnements lumineux ne sont pas les seuls qui peuvent être utilisés pour injecter des fautes. Il est également possible d'utiliser les rayonnements de particules et les rayonnements électromagnétiques.

Nous avons présenté, dans cette section, une classification possible des attaques impactant la sécurité des systèmes informatiques. Cette classification hiérarchique est issue d'un travail de caractérisation de ces attaques selon trois axes : le niveau d'abstraction du système où se situe l'attaque, l'objet de l'attaque à ce niveau d'abstraction et enfin le vecteur d'attaque que peut utiliser un attaquant sur cet objet. Afin de révéler rapidement les fautes dans leurs systèmes, nous constatons, aujourd'hui, que de plus en plus d'entreprises, conscientes de l'importance de

la sécurité informatique, intègrent des méthodes et des techniques d'élimination des fautes à leur cycle de développement. La section qui suit dresse un état de l'art de ces méthodes et de ces techniques qui s'adaptent aussi bien aux composants logiciels qu'aux composants matériels.

1.3 Méthodes et techniques pour l'élimination des fautes

Comme précédemment indiqué (cf. sous-section 1.1.2), la conception et la réalisation d'un système informatique sûr de fonctionnement passent par l'utilisation combinée d'un ensemble de méthodes qui sont classées en quatre moyens : la prévention, l'élimination, la tolérance et la prévision des fautes. La présente section s'intéresse plus particulièrement à l'élimination des fautes, qui vise à réduire le nombre et la sévérité des fautes. Elle est réalisée pendant la phase de développement d'un système par vérification, diagnostic et correction, ou pendant sa phase opérationnelle par maintenance [Laprie *et al.* 96].

La vérification consiste à déterminer si le système satisfait des propriétés, appelées conditions de vérification [Cheheyl *et al.* 81] ; si ce n'est pas le cas, deux autres étapes doivent être entreprises : diagnostiquer la ou les fautes qui ont empêché les conditions de vérification d'être remplies, puis apporter les corrections nécessaires. Après correction, le processus doit être recommencé afin de s'assurer que l'élimination des fautes n'a pas eu de conséquences indésirables.

Les fautes présentes dans un système peuvent être révélées par différentes techniques qui sont classées selon qu'elles impliquent ou non l'activation du système (cf. figure 1.4). La *vérification* d'un système sans activation réelle est dite *statique*, par opposition à la *vérification dynamique* qui nécessite son activation. Les sous-sections suivantes détaillent ces différentes techniques.

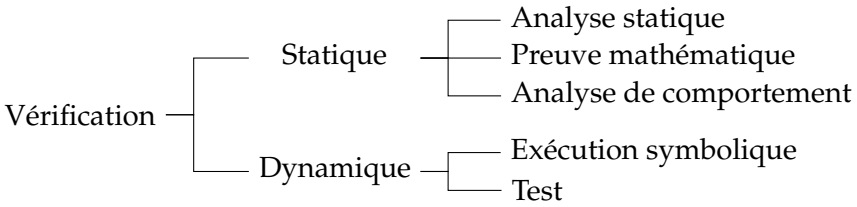

FIGURE 1.4 – Classification des techniques de vérification

1.3.1 Analyse statique

L'analyse statique peut être manuelle, ou automatique. L'*analyse statique manuelle* correspond aux techniques de revue ou d'inspection prévues tout au long du cycle de développement du système [Fagan 76, Dyer 92, van Emden 92, Ebenau et Strauss 94, Myers *et al.* 04]. Une équipe de personnes se réunit pour analyser en détail les documents (par exemple, spécification, ou dossier de conception générale ou détaillée, ou code-source) relatifs à la phase en cours. L'examen de ces documents est guidé par des listes de contrôle (en anglais, *check-list*) (contenant des questions à se poser, des standards à respecter, etc.) et donne lieu à des questions ou remarques qui sont discutées avec les auteurs du document. L'*analyse statique automatique* regroupe tous les contrôles effectués à l'aide d'outils informatiques – des compilateurs aux analyseurs les plus évolués. Ces outils fournissent des caractéristiques du code (mesures de complexité, références croisées, chemins d'exécution, etc.) qui facilitent le travail d'analyse

statique manuelle et signalent certaines anomalies structurelles (variables non-initialisées, interfaces incompatibles entre composants, utilisations incohérentes de variables globales, code mort, etc.). À cet effet, les algorithmes mis en œuvre dans ces outils peuvent se baser sur des heuristiques simples ou sur des modèles ou théories plus complexes tels que les graphes de contrôle ou l'interprétation abstraite [Cousot et Cousot 77, Cousot 01].

1.3.2 Preuve mathématique

Une *preuve mathématique* est une suite finie d'inférences dans un système formel, c'est-à-dire dans un formalisme ayant une syntaxe et une sémantique s'appuyant sur des fondements mathématiques associés à des règles de manipulation permettant d'effectuer des transformations et des vérifications. La preuve du programme dont est issu un système logiciel ou matériel par rapport à sa spécification nécessite une formalisation du problème de vérification : la description du système, les hypothèses formulées sur son environnement et les propriétés attendues doivent être exprimées dans un langage formel qui s'appuie sur un cadre logique bien défini.

La plupart des méthodes qui concernent les programmes séquentiels ont recours à une approche basée sur la sémantique axiomatique. Cette sémantique, dont les fondements remontent à [Floyd 67, Hoare 69, Dijkstra 76], consiste à définir une logique mathématique permettant de prouver des propriétés sur des programmes écrits dans un langage de programmation donné. Les logiques mathématiques utilisées peuvent être basées sur :

— la *logique des prédicats*, représentées par les méthodes Larch [Guttag et Horning 93] ou EHDM (*Enhanced Hierarchical Development Methodology*) [Rushby *et al.* 91]) ;
— la *théorie des ensembles*, représentées pincipalement par la méthode VDM (*Vienna Development Method*) [Bjørner et Jones 78, Jones 90], puis la méthode Z [Spivey 88a, Spivey 92b] et la méthode B [Abrial *et al.* 91], qui occupent une place de choix de par le fait qu'elles sont actuellement très utilisées par l'industrie informatique ;

Il convient de noter que la preuve mathématique d'un système requiert un investissement important, et n'est souvent applicable qu'à des systèmes de taille restreinte. Elle permet néanmoins de garantir le respect de certaines propriétés cruciales pour la sécurité, exprimées par exemple sous forme d'invariants [Glory et Bergerand 90],sous réserve que la preuve soit elle-même correcte et que le système formel construit soit cohérent.

1.3.3 Analyse de comportement

L'*analyse de comportement* est basée sur des modèles comportementaux du système, déduits de la spécification, ou des dossiers de conception, ou des codes-sources [Diaz 82, Davis 88, Harel *et al.* 90].Cette analyse met en jeu des simulations de son comportement, modélisées à partir de graphes, d'automates à états finis, de tables de décision, de réseaux de Petri, etc. Il est possible de vérifier des propriétés de cohérence, complétude, vivacité, temps de réponse, etc. dès lors qu'une sémantique formelle est associée au modèle utilisé. La vérification de modèles (en anglais, *model checking*), proposée par [Clarke et Emerson 08, Queille et Sifakis 82], constitue un cas particulier des techniques d'analyse de comportement. Il s'agit d'un ensemble de techniques de vérification automatique de propriétés temporelles sur des systèmes réactifs (par exemple, protocoles de communication, composants électroniques, etc.), c'est-à-dire un système qui est en interaction permanente avec son environnement et dont le rythme est imposé par cet environnement. Elle nécessite de formuler les spécifications dans une logique temporelle propositionnelle et de modéliser le système sous la forme d'un graphe orienté, constitué d'états et de transitions. L'outil de *model checking* (ou *model checker*) parcourt ensuite de façon

exhaustive le graphe en s'assurant qu'une propriété est vérifiée pour chacun de ses états et retourne un contre-exemple si la propriété a été violée dans au moins un de ses états.

1.3.4 Exécution symbolique

L'exécution symbolique de programme consiste à exécuter un programme en lui soumettant des valeurs symboliques en entrée : par exemple, si une entrée X est un nombre entier positif, nous pouvons lui affecter une valeur symbolique a qui représente l'ensemble des valeurs entières supérieures à 100. Une exécution symbolique consiste alors à propager les symboles sous forme de formules au fur et à mesure de l'exécution des instructions. Elle fournit comme résultats les expressions symboliques obtenues pour les sorties. En pratique, les limites auxquelles se heurte cette technique sont nombreuses : dans l'exemple précédent, comment connaître le résultat d'une condition de branchement portant sur une valeur précise de X (par exemple, $X = 200$), résultat qui conditionne la suite des instructions à exécuter ? Que représente l'élément d'indice X dans un tableau lorsque X a la valeur a ? Comment maîtriser l'explosion de la taille et de la complexité des formules obtenues ?

1.3.5 Test

Le *test* est certainement la technique de vérification la plus largement répandue [Roper 92]. Il consiste à exécuter un programme en lui fournissant des entrées valuées, les entrées de test, et à vérifier la conformité des sorties par rapport au comportement attendu. Sa mise en œuvre nécessite de résoudre deux problèmes : le problème de la sélection d'entrées de test, et le problème de l'oracle [Weyuker 82], autrement dit comment décider de l'exactitude des résultats observés. Sauf cas trivial, le test exhaustif est généralement impossible et on est amené à sélectionner de manière pertinente un (petit) sous-ensemble du domaine d'entrée. Cette sélection peut s'effectuer à l'aide de critères de test liés à un modèle de la structure du système. Nous parlons alors de *test structurel*. Elle peut également s'effectuer à l'aide de critères de test liés à un modèle des fonctions que doit réaliser le système. On parle alors plutôt de *test fonctionnel*.

Quel que soit le critère de sélection, la génération des entrées peut être déterministe ou probabiliste. Dans le premier cas qui définit le test déterministe, les entrées sont déterminées par un choix sélectif de manière à satisfaire le critère de test retenu. Dans le deuxième cas qui définit le test statistique ou aléatoire, les entrées sont générées de manière aléatoire selon une distribution probabiliste sur le domaine d'entrée, la distribution et le nombre des données de test étant déterminés à partir du critère retenu [Waeselynck 93]. Ces deux types de génération des entrées de test, déterministe et aléatoire, sont complémentaires [Duran et Ntafos 84, Thevenod-Fosse 91], et devraient être utilisés à toutes les étapes de test. Il est important de rappeler le rôle déterminant du choix de la distribution des probabilités sur le domaine d'entrée, dont dépend fortement l'efficacité du test statistique : la distribution doit être choisie en fonction du critère de test pour permettre une bonne couverture, structurelle ou fonctionnelle, du système.

En ce qui concerne le problème de l'oracle, un dépouillement automatisé des résultats de test est toujours souhaitable, et devient indispensable lorsqu'un grand nombre d'entrées ont été sélectionnées. Les solutions les plus satisfaisantes sont basées sur l'existence d'une spécification formelle du programme sous test. La spécification est alors utilisée pour déterminer les résultats attendus, soit lors de la sélection des entrées de test, soit *a posteriori*. D'autres solutions d'oracle partiel peuvent être déterminées au cas par cas, en réalisant des contrôles de vraisemblance sur les résultats de test (contrôle de cohérence entre différentes données, contrôle d'appartenance à une plage de valeurs, etc.).

Dans cette section, nous avons présenté des exemples de méthodes issues du domaine de la sûreté de fonctionnement, mais elles peuvent aussi s'appliquer à la sécurité informatique en les adaptant éventuellement à d'autres propriétés (par exemple, la confidentialité). Il convient de remarquer qu'aucune de ces techniques ne garantit une élimination exhaustive des fautes. Aussi paraît-il essentiel de combiner ces différentes techniques afin d'en révéler un maximum.

1.4 Conclusion

Ce chapitre a introduit une partie des connaissances nécessaires à la compréhension de nos travaux. Nous avons commencé par rappeler les concepts de base et la terminologie associée à la sûreté de fonctionnement des systèmes. Nous avons ensuite situé, dans ce contexte, le sujet principal de nos travaux, à savoir la sécurité des systèmes informatiques et précisé les trois propriétés fondamentales – confidentialité, intégrité et disponibilité – qui la définissent. À partir de là, nous nous sommes concentrés sur les actions qui cherchent à mettre en défaut ces propriétés. La caractérisation de ces attaques a abouti à une classification de ces actions malveillantes selon trois axes complémentaires : le niveau d'abstraction du système auquel s'effectue l'attaque, l'objet de l'attaque à ce niveau d'abstraction et le vecteur d'attaque utilisé sur cet objet. Remarquons que cette classification peut être adaptée à tout type de système informatique. Elle nécessite cependant, afin de lever toute ambiguïté, de définir précisément la frontière entre le système considéré et son environnement. Finalement, puisque ces attaques exploitent des fautes de conception ou de configuration introduites de manière intentionnelle ou accidentelle, nous avons présenté les méthodes et les techniques qui visent à éliminer ces fautes, aussi bien lors du développement du système informatique que lors de son service en vie opérationnelle.

Dans le prochain chapitre, nous proposons un modèle général d'attaques sur les systèmes informatiques. Celui-ci s'appuie, entre autres, sur la classification des actions malveillantes que nous venons de présenter. L'objectif que nous avons poursuivi lors de l'élaboration de ce modèle d'attaques a été d'identifier de façon exhaustive, puis d'étudier, les attaques impactant la sécurité des systèmes informatiques. Une attention particulière est ensuite prêtée, dans les chapitres 3 et 4, aux attaques qui reposent sur l'utilisation des mécanismes d'entrées-sorties auxquels nous nous intéressons dans les travaux décrits dans ce manuscrit.

Chapitre 2

Élaboration d'un modèle d'attaques

L'élaboration d'un modèle d'attaques est essentielle pour étudier de manière méthodique les attaques impactant les systèmes informatiques. Celui que nous avons établi pour notre étude s'inspire fortement des graphes d'attaque [Sheyner 04] et couvre des scénarios d'attaque qui impliquent éventuellement différents niveaux d'abstraction d'un système informatique. Notre démarche repose sur le fait qu'une attaque quelconque ciblant un système informatique peut être décomposée en une séquence d'actions élémentaires dont certaines d'entres elles – les attaques élémentaires – ont l'intention de nuire aux propriétés de sécurité de ce système ou d'un système ayant une quelconque relation avec lui. Naturellement, cette approche nous a amenés, d'une part, à identifier l'ensemble des actions élémentaires possibles dans un système informatique et, d'autre part, à déterminer les enchaînements de celles-ci qui donnent lieu à des attaques. Ce travail est l'objet de ce chapitre.

Nous commençons, en section 2.1, par introduire quelques éléments sur l'infrastructure matérielle d'un système informatique. En particulier, nous caractérisons fonctionnellement les composants matériels qui la constituent et nous rappelons succinctement les architectures de communication au travers desquelles ils interagissent. Ces éléments nous permettent alors d'avoir une vision macroscopique des interactions entre les composants matériels. Nous détaillons ensuite, en section 2.2, la logique que nous avons suivie pour élaborer notre modèle d'attaques. Dans celle-ci, nous considérons les composants matériels comme des systèmes à part entière qui interagissent avec d'autres composants matériels, donc d'autres systèmes et nous modélisons leur structure interne sous la forme d'un graphe de flots de données. En reportant les vecteurs d'attaque que nous avons précédemment identifiés (cf. section 1.2) sur ce modèle, nous déterminons l'ensemble des actions élémentaires (et des attaques élémentaires) au sein d'un composant matériel. La combinaison du graphe d'actions élémentaires associé à chaque composant matériel pris isolément et des interactions possibles entre ces différents composants nous permet alors de déduire de manière aussi exhaustive que possible l'ensemble des scénarios d'attaque dans un système informatique. Nous illustrons le modèle obtenu, en section 2.3, par plusieurs exemples réels d'attaque. Finalement, la section 2.4 discute de quelques limites de notre modèle d'attaques et propose quelques pistes d'amélioration.

2.1 Infrastructure matérielle d'un système informatique

Nous avons précédemment défini un système informatique comme la composition de deux systèmes complexes, l'un matériel et l'autre logiciel, organisés dans le but de remplir une fonction ou une mission informatique dans un environnement donné (cf. section 1.2). Dans la pré-

sente section, nous nous concentrons spécifiquement sur le système matériel et, en particulier, sur son infrastructure. Celle-ci est constituée de plusieurs composants matériels hétérogènes tels que des processeurs, des mémoires, des contrôleurs d'entrées-sorties, des périphériques, etc. qui interagissent au travers d'une architecture de communication dans le but d'accomplir des tâches spécifiques. Cette section traite de façon générale de différents aspects de l'infrastructure matérielle d'un système informatique. Nous commençons par rappeler la terminologie associée aux bus informatiques. Ensuite, nous nous intéressons aux aspects fonctionnels des composants matériels et nous décrivons ceux que nous retrouvons typiquement dans un système informatique.

2.1.1 Architectures de communication

Dans une infrastructure matérielle, les composants matériels peuvent endosser plusieurs rôles sur les bus informatiques. Ceux qui initient et contrôlent les transferts de données sont appelés *composants maîtres*. Ils se connectent aux bus informatiques au travers de *ports maîtres*. Typiquement, les processeurs se comportent en maîtres car ils initient des opérations de lecture et d'écriture vers d'autres composants matériels du système informatique (par exemple, la mémoire centrale, les contrôleurs, etc.). Les *composants esclaves* répondent simplement aux demandes de transfert de données des composants maîtres sans pouvoir en initier eux-mêmes. Ceux-ci sont reliés aux bus informatiques par des *ports esclaves*. La plupart des périphériques sont des composants esclaves. Ils répondent aux opérations de lecture et d'écriture d'autres composants maîtres tels que les processeurs. Il convient de remarquer que certains composants peuvent aussi jouer les deux rôles de maître et d'esclave. C'est le cas, en particulier, des contrôleurs de mémoire, des contrôleurs de périphériques et, plus généralement, des ponts. Nous rappelons qu'un *pont* est un composant matériel qui permet d'interconnecter deux bus qui n'implémentent pas nécessairement les mêmes protocoles et, éventuellement, ne fonctionnent pas aux mêmes fréquences d'horloge. À ce titre, il endosse alternativement le rôle de composant maître (pour émettre) et de composant esclave (pour recevoir) sur les bus qu'il interconnecte.

Chaque composant esclave se voit généralement attribuer une plage d'adresses. Lorsqu'un composant maître effectue un transfert de données, il diffuse sur les bus l'adresse de destination et les données à transférer. Le *décodeur* se charge alors d'interpréter cette adresse et de sélectionner le composant esclave destiné à recevoir ces données. Cette opération de décodage d'adresse peut être mise en œuvre de façon centralisée ou de façon distribuée. Dans le cas d'une mise en œuvre centralisée, le décodeur prend en entrée l'adresse du transfert de données et émet ensuite un signal pour activer le composant esclave auquel est destiné ce transfert. À l'opposé, tous les esclaves ont leurs propres décodeurs dans le cas d'une mise en œuvre distribuée. Lors d'un transfert de données, les décodeurs associés à chaque esclave décodent individuellement l'adresse transmise sur les bus afin de déterminer si le transfert leur est destiné.

Lorsque plusieurs composants maîtres sont connectés à un même bus, il est possible qu'ils initient simultanément un transfert de données. Étant donné que seul un transfert de données peut avoir lieu sur les bus à un instant donné, un *arbitre* est nécessaire pour désigner le composant maître prioritaire pour procéder à un transfert de données. Les critères retenus pour déterminer le composant maître qui a accès au bus forment le *schéma d'arbitrage*. De la même façon que pour un décodeur, un arbitre peut être mis en œuvre de façon centralisée ou de façon distribuée. La figure 2.1 illustre ces notions de composant maître et de composant esclave, d'arbitre et de décodeur que nous venons d'aborder.

Les bus informatiques peuvent être organisés de différentes manières au sein d'un système informatique. Le choix d'une organisation par rapport à une autre dépend généralement de

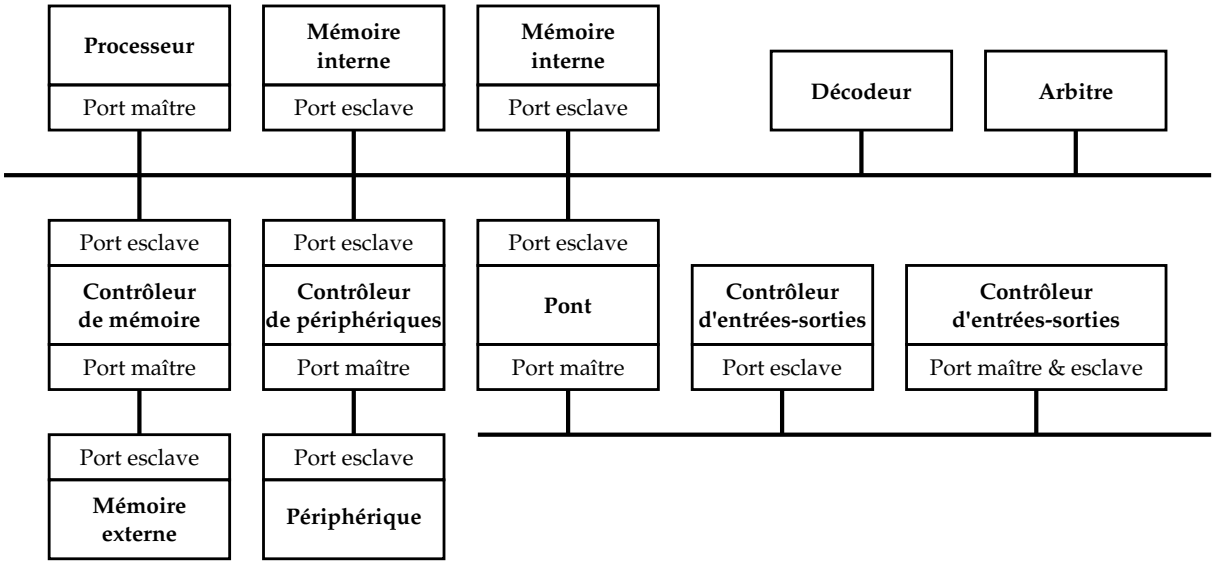

FIGURE 2.1 – Illustration de la terminologie des bus informatiques

plusieurs critères parmi lesquels les plus récurrents sont le coût, la complexité, la puissance et les performances. Différentes topologies de bus existent dans un système informatique : le bus partagé, le bus en étoile, le bus en anneau, etc. Celles-ci peuvent être combinées pour former l'*architecture de communication* dont le rôle est d'acheminer correctement et de manière fiable les flots de données à partir des composants sources aux composants destinataires, tout en garantissant, éventuellement, une certaine qualité de service (latence, bande passante, etc.).

Les composants matériels peuvent être décrits selon différents points de vue. Les ingénieurs électroniciens les décrivent par exemple comme une combinaison de composants électroniques (puces, transistors, résistances, condensateurs, etc.) interconnectés par des fils. Les ingénieurs logiciel les perçoivent plutôt comme des entités qui accomplissent une ou plusieurs fonctions et avec lesquelles il est possible d'interagir par des commandes. Jusqu'à présent, nous nous sommes attachés à décrire les multiples rôles (arbitre, décodeur, maître ou esclave) qu'ils peuvent endosser sur les bus auxquels ils sont connectés. La prochaine sous-section (§2.1.2) complète cette vision des composants matériels en considérant leurs aspects fonctionnels.

2.1.2 Composants matériels

Conceptuellement, l'infrastructure matérielle du système informatique peut être décrite par un modèle similaire à celui représenté sur la figure 2.2. Les processeurs, le répartiteur, le contrôleur de mémoire et ses mémoires, les contrôleurs d'entrées-sorties, les périphériques, le contrôleur d'interruptions, et le gestionnaire de la plate-forme sont interconnectés au travers d'une architecture de communication. Dans ce modèle, certains composants matériels sont indispensables au fonctionnement du système informatique et d'autres sont optionnels. La présente sous-section décrit succinctement les principales fonctions de ces composants.

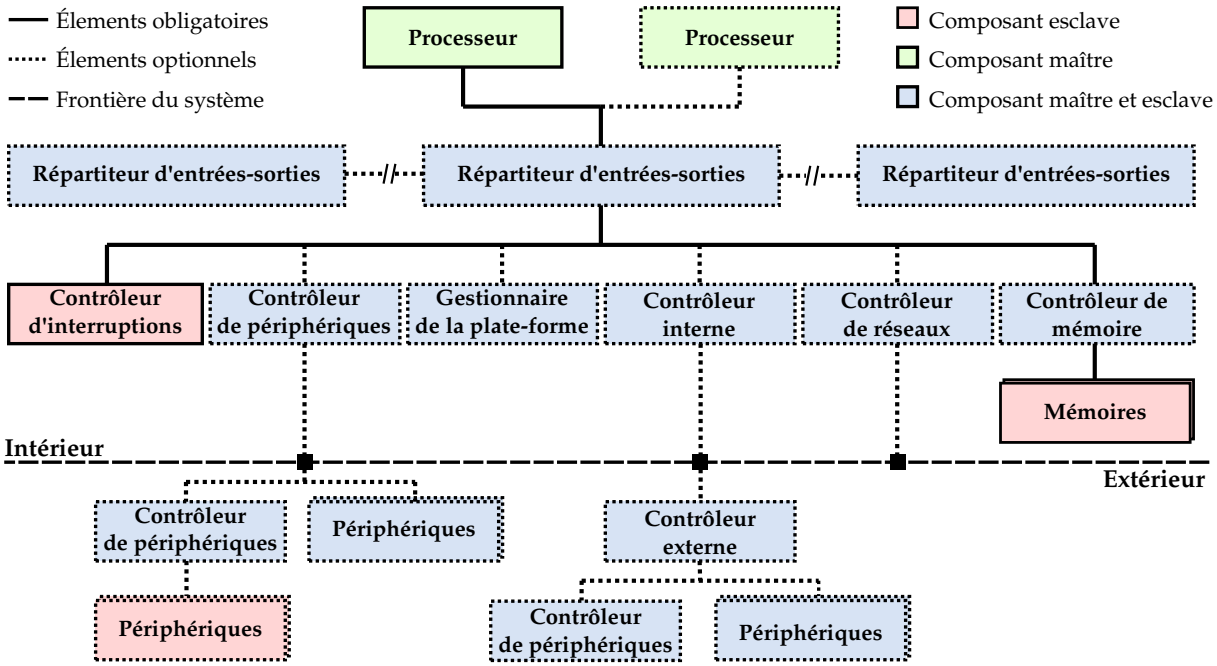

FIGURE 2.2 – Modèle d'infrastructure matérielle pour un système informatique

2.1.2.1 Processeur

Le processeur est un élément essentiel dans un système informatique. En effet, il constitue l'unité centrale de traitement (en anglais, *central processing unit*) du système. Les traitements opérés par ce composant matériel sont décrits par une séquence d'instructions qu'il récupère de la mémoire centrale, décode afin de déterminer leur type et leurs opérandes, puis exécute. Ces opérations sont effectuées de manière cyclique jusqu'à ce que le traitement se termine. Un système informatique peut contenir un ou plusieurs processeurs. Ils accèdent à la mémoire centrale au travers du répartiteur auquel ils sont reliés par un bus système. Ils se comportent généralement en composants maîtres sur ces bus.

2.1.2.2 Répartiteur

Le répartiteur désigne un composant matériel qui interconnecte les bus systèmes avec les bus locaux. Ainsi, il se comporte à la fois en composant maître et en composant esclave sur les bus. Dans son rôle d'esclave, il décode les transferts de données initiés par le processeur ou par les contrôleurs d'entrées-sorties et, dans son rôle de maître, les relaie à leurs destinataires. Il est chargé d'aiguiller correctement ces accès dans l'architecture de communication et il définit, par là-même, les différents espaces d'adressage qui permettent aux composants matériels d'interagir entre eux.

2.1.2.3 Contrôleur de mémoire et mémoires associées

La mémoire centrale stocke toutes les instructions et les données des programmes qui sont exécutés par le processeur. Elle résulte de l'association, d'une part, du contrôleur de mémoire et, d'autre part, des circuits électroniques qui forment la mémoire et sur lesquels sont physiquement stockées les données. Le contrôleur de mémoire agit comme un pont entre les bus

systèmes et les bus mémoires. En effet, il est chargé de traduire des requêtes de lecture ou d'écriture en mémoire en provenance d'un processeur ou des contrôleurs d'entrées-sorties vers le protocole de bus (par exemple, SDRAM, DDR SDRAM, etc.) implémenté par les barettes de mémoire utilisées dans le système informatique. Ainsi, le contrôleur de mémoire joue à la fois le rôle d'esclave sur les bus systèmes qui le relient au répartiteur et le rôle de maître sur le bus mémoire qui le connecte aux barettes de mémoire, agissant alors en esclaves.

2.1.2.4 Contrôleurs d'entrées-sorties

Les contrôleurs d'entrées-sorties désignent de façon générale tous les composants matériels de type pont. Leur dénomination tient de leurs fonctions : ils contrôlent différents bus sur lesquels transitent les entrées-sorties, c'est-à-dire les échanges entre le processeur et des systèmes externes tels que des périphériques ou d'autres systèmes informatiques. Nous distinguons plusieurs types de bus :

— les bus locaux (ou bus d'entrées-sorties) relient les contrôleurs d'entrées-sorties au répartiteur. Cette connexion au répartiteur est souvent directe. Il arrive, cependant, que celle-ci soit indirecte et se fasse au travers d'un autre contrôleur d'entrées-sorties ;
— les bus d'extensions sont chargés de connecter le système informatique à des systèmes matériels externes, en particulier à des périphériques.

Les contrôleurs d'entrées-sorties se distinguent principalement par les fonctions spécifiques qu'ils remplissent au sein du système informatique. Par exemple, lorsque leur rôle consiste à connecter les bus locaux (respectivement, les périphériques aux bus locaux), nous parlons spécifiquement de contrôleurs de bus locaux (respectivement de contrôleurs de périphériques) ; lorsque celui-ci est utilisé pour communiquer avec un autre système informatique au travers du réseau, nous parlons alors de contrôleur réseau ; etc. Ils sont généralement internes au système informatique. Dans ce cas, ils sont directement intégrés à la carte-mère par son fabricant ou ajoutés au système informatique par des connecteurs dédiés. À l'instar des contrôleurs embarqués dans les cartes PCMCIA, Express Card ou Thunderbolt, d'autres contrôleurs d'entrées-sorties sont externes au système informatique.

2.1.2.5 Périphériques

La littérature désigne les périphériques comme des systèmes matériels auxiliaires (tels que les claviers, les souris, les imprimantes, les *webcams*, etc.) qu'il est possible de connecter au système informatique dans le but d'étendre ses fonctionnalités. Malheureusement, cette définition n'est pas suffisamment précise. En effet, les cartes d'extensions telles que les cartes réseau, les cartes graphiques peuvent être considérées comme des périphériques au même titre que les imprimantes, les scanners, les *webcams*, etc. Pour les dissocier, nous allons considérer dans la suite de ce manuscrit que la connexion d'un périphérique à un système informatique se fait nécessairement au travers d'un bus d'extension issu d'un contrôleur de périphérique. Nous distinguons généralement deux catégories de périphériques : ceux qui sont internes et ceux qui sont externes au système informatique. Les périphériques tels que les claviers, les souris et les imprimantes sont dits externes car ils se situent, en général, à l'extérieur du système informatique. À l'opposé, les périphériques tels que les disques dur internes ou les lecteurs de disques internes sont dits internes car ils sont situés à l'intérieur de l'ordinateur (serveur, ordinateur de bureau, ordinateur portable, etc.). La plupart des périphériques se comportent en composants esclaves et répondent aux commandes initiées par les contrôleurs de périphériques. Cependant, certains périphériques peuvent se comporter à la fois en composants maîtres et en

composants esclaves sur leurs bus. C'est le cas notamment des périphériques qui se connectent aux bus USB *On-The-Go* (OTG) et aux bus FireWire.

2.1.2.6 Contrôleur d'interruptions

Le contrôleur d'interruptions se charge de mettre en attente les demandes d'interruptions provenant des différents contrôleurs d'entrées-sorties et de les notifier aux processeurs qui vont immédiatement exécuter une routine d'interruption pour les traiter. Pour traiter les demandes d'interruptions qui ont lieu simultanément, il est possible de programmer le contrôleur d'interruptions pour affecter des priorités différentes à chaque contrôleur d'entrées-sorties. L'interruption de priorité supérieure est alors traitée la première pendant que l'interruption de priorité plus faible est mise en attente. Du point de vue de l'architecture de communication, le contrôleur d'interruptions agit comme un pont qui interconnecte d'une part les lignes d'interruptions qui transportent les demandes d'interruptions des contrôleurs d'entrées-sorties et d'autre part la ligne d'interruption qui notifie le processeur de ces demandes. Aujourd'hui, ces lignes d'interruptions sont intégrées aux bus locaux et aux bus systèmes.

2.1.2.7 Gestionnaire de la plate-forme

Le gestionnaire de la plate-forme regroupe un ensemble de composants matériels en charge de la configuration et la maintenance de la plate-forme matérielle. Sur les systèmes informatiques compatibles PC, les mémoires qui stockent le *Basic Input/Output System* (BIOS) et l'*Unified Extensible Firmware Interface* (UEFI) sont des exemples de tels composants matériels. D'autres technologies matérielles plus spécifiques aux plates-formes telle qu'Intel *Active Management Technology* (AMT) [14] dans les *chipsets* Intel récents sont également considérées comme faisant partie du gestionnaire de la plate-forme. Il interagit avec les autres composants matériels du système informatique au travers du bus local sur lequel il agit soit en composant maître soit en composant esclave en fonction des composants matériels qui y sont intégrés.

Avec la conjonction de l'évolution des technologies de fabrication des circuits intégrés et de la nature du marché des systèmes électroniques, il est aujourd'hui possible d'intégrer ces composants matériels sur un ou plusieurs circuits intégrés relativement petits. Nous parlons alors spécifiquement de systèmes sur puce (en anglais, *system on-chip*). Dans cette sous-section, nous avons présenté les caractéristiques fonctionnelles de chacun de ces composants matériels indifféremment de leur degré d'intégration. Ce travail de caractérisation est à l'origine du modèle d'attaques que nous décrivons dans la prochaine section (§2.2).

2.2 Modèle d'attaques basé sur les interactions entre les composants

Cette section présente le modèle d'attaques que nous avons établi pour identifier les attaques ciblant les systèmes informatiques. La construction de ce modèle repose principalement sur l'idée qu'une attaque peut être décomposée en une séquence d'actions élémentaires parmi lesquelles certaines sont légitimes et l'intention d'autres est de nuire aux propriétés de sécurité

[14] Intel AMT désigne une technologie d'administration à distance de plates-formes. Utilisée avec des applications d'administration et de sécurisation des systèmes informatiques, cette technologie facilite la maintenance (c'est-à-dire, la détection, la réparation et la protection) des ressources informatiques en réseau.

de ce système ou d'un système ayant une quelconque relation avec lui. Ainsi, une attaque peut être formalisée comme suit.

Soit les ensembles suivants :

$\mathcal{A}$ l'ensemble des attaques dans un système informatique,
$\mathcal{E}$ l'ensemble des actions élémentaires,
$\mathcal{L}$ l'ensemble des actions élémentaires légitimes et
$\mathcal{M}$ l'ensemble des attaques élémentaires.

L'ensemble des actions élémentaires $\mathcal{E}$ est formé de l'ensemble des actions élémentaires légitimes auquel est adjoint l'ensemble des attaques élémentaires ($\mathcal{E} = \mathcal{L} \cup \mathcal{M}$). Pour simplifier notre formalisme, nous allons considérer que ces deux ensembles sont disjoints ($\mathcal{L} \cap \mathcal{M} = \emptyset$). En réalité, il peut parfois être difficile de différencier les actions élémentaires légitimes des attaques élémentaires. En effet, en fonction des hypothèses effectuées sur le système étudié et du contexte dans lequel une attaque est menée, elle peut être perçue comme appartenant à l'un des deux ensembles. Bien qu'il s'agisse d'une attaque en confidentialité, l'accès à une ressource à laquelle un utilisateur ne devrait pas avoir accès, dans la mesure où il y a un défaut dans la politique de sécurité et que celle-ci permet l'accès, pourrait être perçu comme un accès légitime par exemple. La simplification que nous avons faite au sujet des ensembles $\mathcal{L}$ (actions élémentaires légitimes) et $\mathcal{M}$ (attaques élémentaires) est, par conséquent, une hypothèse forte mais acceptons, pour des raisons de clarté, qu'elle soit valide.

Lors d'une attaque, plusieurs actions élémentaires sont exécutées en séquence. Une action élémentaire antérieure prépare la mise en œuvre d'une autre action élémentaire qui lui est postérieure. Pour exprimer cette notion de séquence et d'interdépendance entre actions élémentaires, nous introduisons la relation de précédence notée $\prec$:

Soit $e_1, e_2 \in \mathcal{E}$ deux actions élémentaires,
$e_1 \prec e_2$ signifie que l'action élémentaire e_1 doit nécessairement avoir eu lieu et que celle-ci ait, suite à son déroulement, satisfait les conditions pour la mise en œuvre de l'action élémentaire e_2 qui pourra se dérouler à son tour.

Si $e_1 \prec e_2$, alors e_1 et e_2 peuvent constituer une séquence d'actions élémentaires notée $e_1 \bullet e_2$. Par ailleurs, si $e_1 \prec e_2$ et si $e_2 \prec e_3$ alors e_1, e_2 et e_3 peuvent être concaténées pour former une seule séquence d'actions élémentaires notée $e_1 \bullet e_2 \bullet e_3$. Une séquence d'actions élémentaires est considérée comme une attaque lorsqu'elle contient au moins une attaque élémentaire. L'ensemble des attaques $\mathcal{A}$ peut alors être défini par la relation suivante :

$$\mathcal{A} = \{e_1 \bullet e_2 \bullet \ldots \bullet e_i \in \mathcal{E}^+ : \forall i \in [1, n-1], e_i \prec e_{i+1} \text{ et } \exists j \in [1, n], e_j \in \mathcal{M}\}$$

Cette formalisation mathématique du concept d'attaque peut être appliquée pour différents niveaux d'abstraction d'un système informatique. Seuls le modèle et le séquencement des actions élémentaires doivent alors être adaptés de façon à pouvoir représenter les attaques auxquelles nous nous intéressons. Ces deux éléments sont traités dans les sous-sections 2.2.1 et 2.2.2. Nous décrivons, pour commencer, le modèle des actions (et des attaques) élémentaires que nous avons établi pour notre étude.

2.2.1 Modèle des actions élémentaires et des attaques élémentaires

Le modèle que nous souhaitons établir cherche à couvrir de manière aussi complète que possible les attaques dans un système informatique et doit satisfaire, à cet effet, plusieurs ca-

ractéristiques. Nous avons affirmé, lors de notre travail de caractérisation des attaques élémentaires (cf. section 1.2), que celles-ci peuvent agir à différents niveaux d'abstraction du système informatique. Aussi est-il important, dans notre modèle, de couvrir ces différents niveaux d'abstraction. Par ailleurs, une attaque peut impliquer plusieurs systèmes. Dans le cadre de nos travaux, un type de système nous intéresse en particulier : les composants matériels et, plus spécifiquement, les contrôleurs d'entrées-sorties. Ainsi, notre modèle d'attaques doit également prendre en compte les attaques entre les différents composants matériels et doit pouvoir les représenter.

Afin de concilier ces caractéristiques au sein de notre modèle d'attaques, nous proposons, dans un premier temps, d'établir un modèle générique de composant matériel sur lequel nous pouvons nous appuyer pour identifier les actions élémentaires ainsi que les attaques élémentaires spécifiques à tout composant matériel.

Un composant matériel peut être modélisé comme un système à part entière au sein duquel plusieurs sous-systèmes, organisés en couches, interagissent. Nous discernons trois principaux sous-systèmes – communication, configuration et logique – auxquels s'associe optionnellement un sous-système logiciel lorsque le composant matériel dispose d'une partie programmable, un *firmware* par exemple. Le sous-système de communication appartient à la couche la plus basse. Il se charge des communications avec les autres composants matériels en offrant aux autres sous-systèmes du composant une abstraction du médium de communication, autrement dit les bus. Les communications sont initiées (dans le cadre d'une émission) et traitées (dans le cadre d'une réception) par le sous-système logique qui implémente les principaux services fournis par le composant matériel. Ces services peuvent éventuellement être étendues par d'autres services implémentés par le sous-système logiciel. Les accès à ces services depuis un autre composant matériel se font au moyen du sous-système de configuration. Les flots d'informations entre ces sous-systèmes sont décrits sur la figure 2.3. Les arcs reliant les sous-systèmes représentent les actions élémentaires.

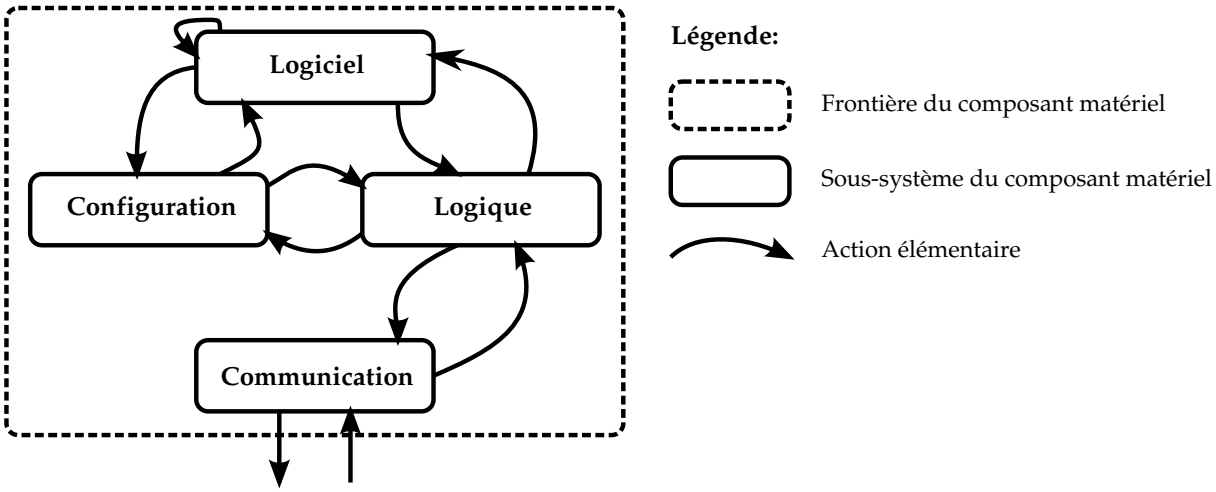

FIGURE 2.3 – Actions élémentaires dans un composant matériel

Grâce au travail de caractérisation des attaques élémentaires que nous avons présenté au chapitre 1 (cf. section 1.2), il est immédiat d'identifier l'ensemble des attaques élémentaires sur notre modèle générique de composant matériel. Ces attaques élémentaires sont illustrées sur la figure 2.4. Cette représentation sous la forme d'un graphe orienté rend compte de la causalité entre les actions élémentaires nécessaires à leur enchaînement. Dans ce formalisme graphique,

un arc représente une action élémentaire qui peut être réalisée par un sous-système source sur un sous-système cible. Les attaques élémentaires sont représentées par des arcs étiquetés d'un numéro indiquant le type d'attaque. Pour ce formalisme, nous avons délibérément choisi de ne pas représenter, par exemple, les vecteurs d'attaques. La granularité de représentation que nous avons choisie est amplement suffisante pour les besoins de notre étude.

Il est opportun de remarquer que nous avons simplifié la représentation de certaines classes d'attaques, en particulier les attaques ciblant les canaux auxiliaires. En toute rigueur, il serait nécessaire de tracer un arc depuis le dispositif externe vers chaque sous-système du composant matériel ainsi que vers chaque médium de communication au travers desquels ils communiquent (également représentés par les arcs). Pour alléger cette représentation, nous avons tracé un unique arc depuis le dispositif externe vers la frontière du système cible.

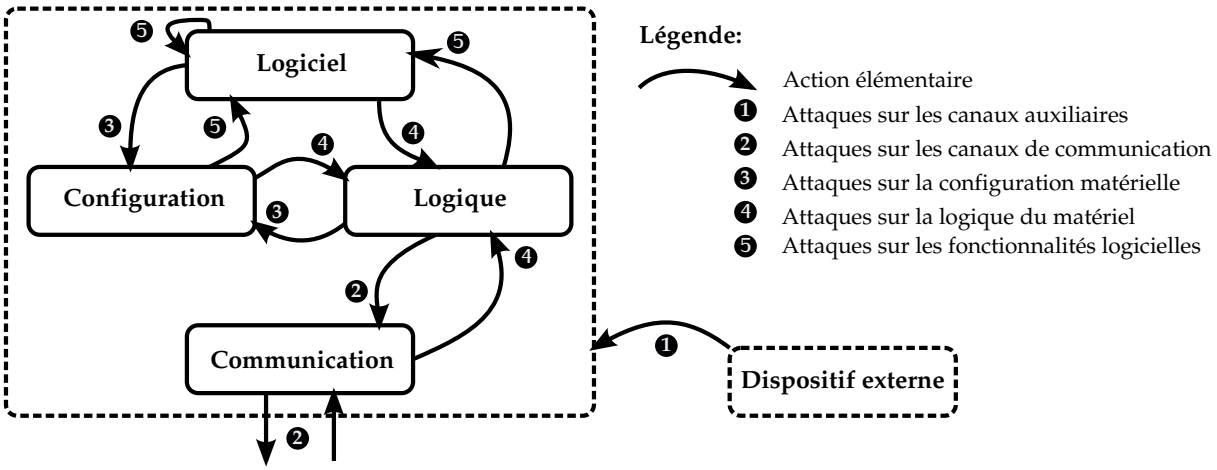

FIGURE 2.4 – Attaques élémentaires dans un composant matériel

Le modèle d'attaques élémentaires que nous venons d'établir s'applique à tout composant matériel. Il suffit alors d'instancier ce modèle pour chacun des composants matériels du système informatique étudié afin d'identifier les actions élémentaires légitimes et les attaques élémentaires qui leur sont spécifiques. Bien évidemment, cette instanciation est fonction des hypothèses d'attaques considérées pour chaque composant matériel. Par ailleurs, nous pouvons combiner ces instances construites individuellement en considérant les flots d'informations possibles entre les composants matériels pour obtenir l'ensemble des actions élémentaires légitimes et des attaques élémentaires pour le système informatique considéré.

2.2.2 Séquences d'actions et d'attaques élémentaires

Il est possible de déterminer l'ensemble des attaques pour un système informatique par un parcours exhaustif des chemins du graphe des actions élémentaires. Bien que fastidieux et sujet à de nombreuses erreurs, ce traitement peut être opéré de façon manuelle. Il peut également être assisté par des outils automatiques implémentant des algorithmes de parcours de graphes [Cormen *et al.* 09]. Nous donnons, dans la suite, quelques éléments d'implémentations d'un tel outil qui contribueraient à accélérer considérablement ses calculs.

En raison de la caractérisation (parfois trop) fine des actions élémentaires, il peut apparaître au sein du graphe certaines actions élémentaires qui ne contribuent pas directement au processus d'attaque et qui sont par là-même superflues. Nous pouvons alors envisager de réduire la

taille du graphe en fusionnant les sommets (et les arcs associés) qui n'apportent pas d'informations significatives aux attaques. Aussi, pour réduire la taille du graphe et réduire en conséquence le temps nécessaire aux calculs de l'ensemble des chemins possibles, nous proposons, dans un premier temps, de fusionner les sommets appartenant à une même classe d'équivalence, c'est-à-dire les sommets qui sont fortement connexes et dont les chemins pour aller d'un sommet à un autre sont composés uniquement d'actions élémentaires légitimes (et donc d'arcs non-étiquetés). Nous rappelons qu'en théorie des graphes [Gross et Yellen 03], un couple de sommets (u, v) sont fortement connexes lorsqu'il existe un chemin de u à v. Afin de garder la structure générale du graphe et de ne pas perdre d'informations significatives, il est important d'appliquer ces réductions localement à chaque composant matériel. En effet, en les appliquant à l'ensemble des sommets du graphe, nous pouvons être confrontés parfois à une situation où des sous-systèmes de composants différents sont fusionnés. Il devient alors difficile d'identifier précisément les sources et les destinations des attaques.

Afin d'accélérer les calculs de l'ensemble des chemins du graphe, il est également possible de calculer au préalable l'ensemble des chemins possibles au sein de chaque composant matériel pris isolément. Nous pouvons ensuite considérer les interactions qui existent entre les composants matériels. En ayant pré-calculé les chemins internes à chaque composant, il est possible de calculer rapidement l'ensemble des chemins correspondant à des séquences d'attaques couvrant l'ensemble des composants du système informatique considéré.

2.2.3 Application à un modèle de système informatique

La présente sous-section illustre la mise en œuvre de notre modèle d'attaques en l'instanciant à un système informatique particulier représenté sur la figure 2.5. Il s'agit d'une instance (très) simplifiée du modèle générique de système informatique que nous avons précédemment décrit en section 2.1 (cf. figure 2.2). Pour les besoins du manuscrit, cette instance omet volontairement plusieurs composants matériels (tels que le contrôleur d'interruption, le gestionnaire de la plate-forme, etc.) que nous n'allons pas considérer pour notre étude. Il convient également de remarquer que nous avons regroupé plusieurs composants matériels implémentant des fonctions similaires (les contrôleurs d'entrées-sorties, les périphériques, etc.) au sein d'une même entité afin de réduire la taille du graphe des actions élémentaires construit et, par là-même, gagner en lisibilité. Bien évidemment, un système informatique réel sera très rarement aussi simple. Ainsi, le système informatique que nous allons considérer est composé d'un seul processeur, d'une mémoire centrale et de contrôleurs d'entrées-sorties, interconnectés au moyen de bus distincts reliés à un unique répartiteur. Les contrôleurs d'entrées-sorties étant ici génériques, ils pourront être considérés comme des contrôleurs internes, comme des contrôleurs externes ou comme des contrôleurs de périphériques selon le contexte.

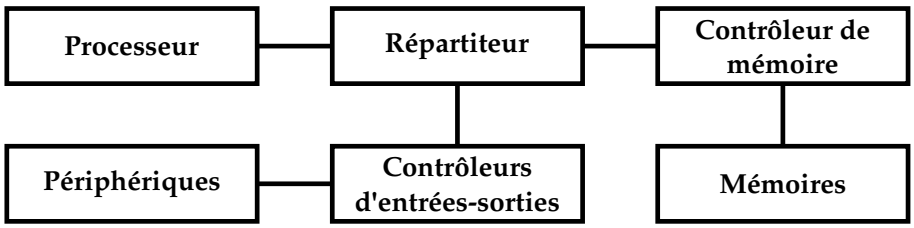

FIGURE 2.5 – Exemple de système informatique

Pour identifier l'ensemble des attaques élémentaires dans le système informatique, nous

avons formulé les hypothèses d'attaque suivantes pour chacun des composants matériels.

Le processeur. Étant donné le rôle actif joué par le processeur dans un système informatique, il peut être utilisé pour initier des attaques ciblant d'autres composants matériels. Toutefois, les sous-systèmes qui le composent peuvent également faire l'objet d'attaques.

— Les composants logiciels qui s'exécutent au dessus de ce processeur sont potentiellement vulnérables. De par leur complexité, ils sont rarement exempts de vulnérabilités qui sont souvent exploitées par les attaquants. Nous allons représenter ces attaques élémentaires par des arcs étiquetés de ❺ au sein du processeur.

— Le sous-système logique dans le processeur est susceptible de contenir des bogues ou des fonctionnalités cachées qui peuvent être mises à profit par les attaquants pour perpétrer des attaques. Plusieurs exemples de ces bogues matériels ont été présentés au chapitre 1. Nous allons représenter les attaques élémentaires ciblant le sous-système logique par des arcs étiquetés de ❹ au sein du processeur.

— Le sous-système logique et le sous-système logiciel reposent, pour leur fonctionnement, sur le sous-système de configuration. Il est sans doute possible d'impacter indirectement le fonctionnement de ces deux sous-systèmes par une configuration inappropriée du processeur. Nous allons représenter les attaques élémentaires ciblant le sous-système de configuration par des arcs étiquetés de ❸ au sein du processeur.

Le répartiteur. Ce composant matériel a un rôle passif dans le système informatique. Ainsi, nous considérons que ses sous-systèmes peuvent être les cibles d'attaques.

— Le sous-système logique du répartiteur peut contenir des vulnérabilités matérielles. Notamment, ceci a été démontré par la preuve de concept d'attaque publiée dans [Rutkowska et Wojtczuk 08]. Pour cela, les auteurs exploitent une fonctionnalité de traduction d'adresses (précisément, la fonctionnalité de *memory reclaiming*) au sein du *chipset* (que nous assimilons dans notre modèle au répartiteur) de façon à contourner tous les mécanismes de contrôle d'accès à la mémoire centrale y compris ceux mis en place par les unités de gestion de la mémoire dans les processeurs pour isoler les processus entre eux et ceux mis en place par le répartiteur lui-même pour protéger les sous-systèmes logiciels dans le gestionnaire de la plate-forme. Il semble que la vulnérabilité soit due initialement à une erreur de configuration du répartiteur car les fabricants de BIOS ont publié, peu de temps après la divulgation de la vulnérabilité matérielle, un correctif logiciel empêchant son exploitation. Nous allons représenter ces attaques élémentaires par des arcs étiquetés de ❹ au sein du répartiteur.

— Afin d'activer, de désactiver ou de perturber les services fournis par le répartiteur, un attaquant peut modifier de manière inappropriée la configuration du répartiteur d'entrées-sorties. Nous allons représenter ces attaques élémentaires par des arcs étiquetés de ❸ au sein du répartiteur.

Le contrôleur de mémoire et les mémoires associées. Ces deux composants matériels combinés forment la mémoire centrale. Étant donné le rôle crucial joué par la mémoire centrale dans le fonctionnement d'un système informatique, les sous-systèmes de ces composants peuvent être les cibles d'attaques.

— De nombreux composants matériels, notamment le processeur et certains contrôleurs d'entrées-sorties, reposent sur des structures de contrôle stockées dans la mémoire centrale pour leur fonctionnement. Toute modification inappropriée de ces structures peut potentiellement perturber le fonctionnement de ces composants matériels. Nous assimilons ces modifications inappropriées à des attaques élémentaires

sur le sous-système logique des mémoires. Nous allons représenter ces attaques élémentaires par des arcs étiquetés de ❹ au sein des mémoires.
— Le contrôleur de mémoire peut être reconfiguré de façon à activer, désactiver ou perturber certaines fonctions. Une attaque élémentaire qui peut être envisagée sur ce composant consiste, par exemple, à désactiver les accès à une mémoire. Nous allons représenter ces attaques élémentaires par des arcs étiquetés de ❸ au sein du contrôleur de mémoire.

Les contrôleurs d'entrées-sorties internes. Ces composants peuvent initier des attaques ou, au contraire, en être la cible.
— Les contrôleurs d'entrées-sorties qui contiennent un sous-système logiciel présentent potentiellement des vulnérabilités logicielles qui permettent de reprogrammer leurs fonctions. Des publications récentes ont présenté des attaques qui reprogramment un contrôleur de clavier [Gazet 11] afin d'exploiter une vulnérabilité dans la routine de traitement des System Management Interrupts (SMI) exécutée par le processeur lorsqu'il est dans le mode System Management (SMM) ou qui modifient à distance le comportement d'une carte réseau [Duflot *et al.* 10] afin de mener des attaques de type *Direct Memory Access* (DMA). Nous représentons les attaques élémentaires associées par des arcs étiquetés de ❺ dans les contrôleurs d'entrées-sorties.
— Certains contrôleurs d'entrées-sorties ont la faculté de gérer eux-mêmes leurs transferts de données. Un attaquant peut chercher à modifier la configuration de ces contrôleurs d'entrées-sorties afin de détourner ces transferts à des fins malveillantes (par exemple, pour modifier le contenu de la mémoire centrale). Nous allons représenter les attaques élémentaires utilisés dans ces scénarios d'attaque par des arcs étiquetés de ❷ au sein des contrôleurs d'entrées-sorties.

Les contrôleurs d'entrées-sorties externes et les périphériques De la même façon que pour les contrôleurs d'entrées-sorties internes, les contrôleurs d'entrées-sorties externes ainsi que les périphériques peuvent initier des attaques ou en être les cibles.
— Étant donné qu'ils sont externes au système informatique, il est possible d'effectuer des attaques élémentaires à tous les niveaux d'abstraction de ces composants. Grâce aux micro-contrôleurs de plus en plus puissants et aux technologies de logique programmable, nous pensons en effet qu'un attaquant peut créer son propre composant matériel et modeler les parties du composant matériel à sa convenance pour perpétrer des attaques.

Les hypothèses d'attaque que nous avons formulées ont été établies de façon empirique et subjective, à partir d'attaques référencées dans la littérature et à partir d'attaques élémentaires que nous avons considérées plausibles. Évidemment, pour couvrir davantage d'attaques à partir de notre instance de modèle, il pourrait être souhaitable d'élargir notre spectre d'attaques élémentaires. Cependant, l'enrichissement d'un tel modèle impacte obligatoirement sa lisibilité. Afin de garder une instance de modèle suffisamment intelligible et qui permette de représenter les attaques auxquelles nous nous intéressons, nous avons délibérément restreint le spectre des attaques élémentaires possibles. Par ailleurs, pour simplifier la représentation de notre modèle, nous introduisons la notion d'arc étiqueté epsilon (ϵ). Les arcs de ce type représentent des opérations effectuées de façon transparente par la plate-forme matérielle ou une suite d'actions élémentaires légitimes susceptibles d'être utilisées dans une attaque et que nous souhaitons mettre en relief pour des raisons de lisibilité. Le mécanisme qui contribue à la cohérence des mémoires caches dans le répartiteur illustre typiquement ces opérations automatiques. Pour maintenir à jour les mémoires caches dans le processeur, le répartiteur écoute

continuellement les accès qui transitent sur les bus systèmes et compare les adresses des transferts de données avec celles contenues dans les mémoires caches. Lorsqu'une mémoire cache requiert la donnée transférée, le répartiteur peut automatiquement mettre à jour, et cela de façon transparente, le contenu de cette mémoire. L'ajout de ces arcs nous permet alors d'affiner notre modèle d'attaques en précisant des interactions complémentaires entre composants matériels issus de notre connaissance de la plate-forme matérielle. L'arc étiqueté epsilon (ϵ) reliant le contrôleur d'entrées-sorties et les mémoires met en exergue, quant à lui, les communications impliquées la mise en œuvre du mécanisme d'accès direct à la mémoire. Compte tenu des hypothèses que nous avons formulées et des simplifications que nous avons opérées, nous obtenons le modèle d'attaques élémentaires représenté sur la figure 2.6. Puisque nous n'avons pas considérées les attaques de type ❶ dans notre modèle de menaces, il est logique que celles-ci n'apparaissent pas sur cette figure.

2.3 Illustration du modèle par plusieurs exemples réels d'attaque

Dans cette section, nous illustrons le modèle d'attaques que nous venons d'établir par plusieurs preuves de concept d'attaques qui ont été démontrées pour des systèmes informatiques compatibles PC. Cette section va nous permettre, d'une part, de confronter notre modèle à différentes attaques réelles et, par là-même, de le valider ; et, d'autre part, permettre au lecteur d'appréhender facilement notre modèle. Pour chacun des trois exemples d'attaques que nous détaillons dans la suite, nous commençons par rappeler quelques considérations techniques permettant de mieux les comprendre. En nous appuyant sur le modèle d'attaques que nous avons établi, nous donnons ensuite les grandes lignes de l'attaque. Les séquences d'actions élémentaires sont représentées sur la figure 2.7

2.3.1 Usurpation d'identité de périphériques USB

L'*Universal Serial Bus* (USB) est un standard de bus série conçu pour connecter « à chaud » des périphériques externes à des plates-formes matérielles. Ceux-ci sont alors automatiquement reconnus par le système d'exploitation qui leur affecte dynamiquement des ressources et charge les pilotes de périphériques correspondants sans qu'un redémarrage ne soit nécessaire. De par la multiplicité de leurs utilisations, les périphériques USB sont omniprésents dans les environnements informatiques actuels. Ceux-ci séduisent les utilisateurs par les nombreuses fonctionnalités attractives qu'ils offrent : le transport et le stockage de données à moindre coût, l'amorce d'un système, l'exécution automatique d'un programme, etc.

Ces fonctionnalités sont parfois détournées à des fins malveillantes. En effet, il n'est pas rare qu'un périphérique USB contienne des logiciels malveillants placés intentionnellement par le concepteur ou le propriétaire du périphérique, mais à l'insu de son utilisateur. Ces *maliciels* exploitent ces fonctionnalités pour s'exécuter et se propager sur les systèmes auxquels ils sont connectés. Les systèmes informatiques au cœur de l'installation nucléaire Iranienne de Natanz étant isolés du réseau Internet, l'*Advanced Persistent Threat* (APT) Stuxnet a pleinement profité de ces fonctionnalités (et en particulier de l'exécution automatique de programmes) des périphériques USB pour infecter ces systèmes [Falliere *et al.* 11]. Cette sous-section traite plus particulièrement des attaques qui usurpent l'identité de périphériques USB de type *Human Interface Device* (HID) tels que les claviers USB.

Du point de vue de l'électronique, un périphérique USB est relativement simple. Il communique avec le contrôleur de périphériques USB au travers de deux paires de fils. Une pre-

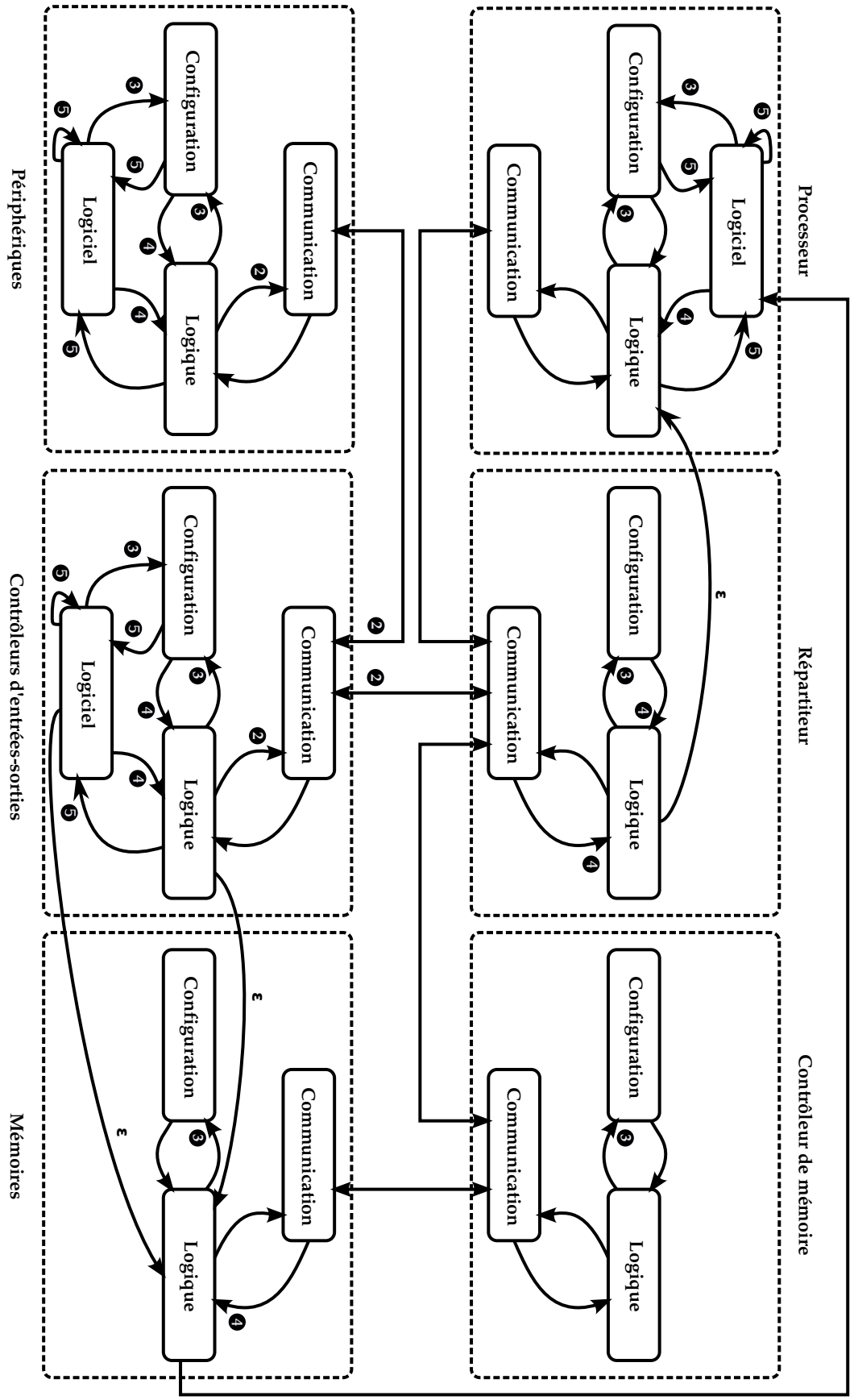

FIGURE 2.6 – Graphe des actions élémentaires pour un système informatique

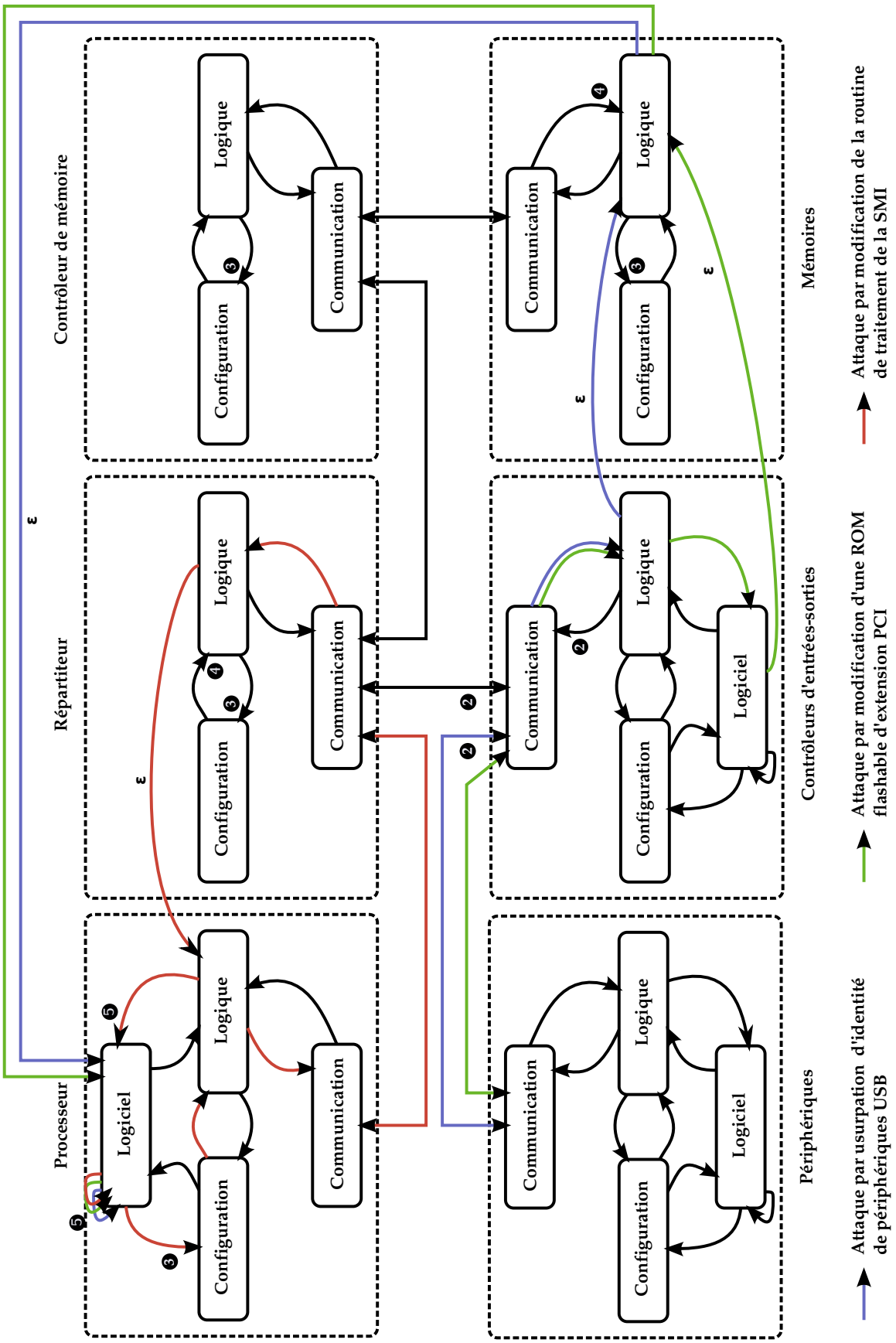

FIGURE 2.7 – Illustration de plusieurs exemples réels d'attaques

mière paire est utilisée pour l'alimentation du périphérique USB et une autre paire est dédiée à la transmission en série des informations selon une signalisation différentielle. Un attaquant disposant un tant soit peu de connaissances en électronique pourrait, sans trop de peine, construire son propre périphérique USB [Crenshaw 10, Pisani *et al.* 10, Kennedy 10] ou modifier un périphérique USB existant de façon à effectuer des attaques ciblées : enregistrement de frappes au clavier, infection par un maliciel, exploitation d'une vulnérabilité particulière du système d'exploitation, etc. Le développement de ces périphériques malveillants est aujourd'hui, entre autres, facilité par l'existence de périphériques USB programmables via des micro-contrôleurs toujours plus miniaturisés. En particulier, il est possible de programmer ce type de périphérique pour agir comme un clavier USB malveillant qui enverrait, par exemple, des frappes pré-définies par l'attaquant. Ces frappes pourraient, par exemple, ajouter automatiquement un utilisateur au système ou télécharger un logiciel malveillant depuis le réseau Internet avant de l'exécuter. Afin de ne pas révéler ses activités et de ne pas éveiller les soupçons des utilisateurs, l'attaquant ne programme généralement pas l'envoi de ces frappes à la connexion du périphérique mais attend des événements particuliers. En connectant un capteur de luminosité au périphérique USB programmable, cet événement pourrait être, par exemple, une chute brutale de la luminosité ambiante que le micro-contrôleur dans le périphérique malveillant interpréterait comme l'extinction d'une lampe lorsqu'on quitte une pièce éclairée. Finalement, pour convaincre l'utilisateur de connecter ce périphérique programmable, il est important de l'encapsuler dans un boitier de périphérique en apparence inoffensif tel qu'un périphérique de stockage.

Ainsi, les attaques par usurpation d'identité de périphériques USB se déroulent généralement en plusieurs étapes. L'attaque débute dès qu'un utilisateur connecte le périphérique USB malveillant au système informatique ciblé. Pour les besoins de notre exemple, nous considérons que l'identité d'un clavier est ici usurpée. Ce périphérique USB malveillant insère alors sur les bus USB (transition ❷) des trames correspondant à des frappes au clavier pré-définies par l'attaquant. Ces trames sont reçues par le contrôleur de périphériques USB qui les transfère automatiquement à la mémoire centrale (transition ϵ). Celles-ci attendent d'être traitées par un sous-système logiciel (le système d'exploitation par exemple) exécuté par le processeur principal (transition ϵ suivie d'une transition ❺).

Un tel scénario d'attaque a été mis en œuvre par Netragard lors d'une opération d'audit de sécurité chez l'un de ses clients [Desautels 11]. L'objectif de cet audit était alors d'obtenir l'accès au parc informatique de son client sans faire usage des moyens d'attaques habituels tels que l'ingénierie sociale par prise de contact direct avec la victime (par exemple, se faire passer pour l'adminstrateur du parc informatique au téléphone), les attaques réseau, les exploitations de vulnérabilités non-publiées (en anglais, *zero-day*), etc. Leur démarche a alors consisté à modifier une souris USB de marque Logitech de façon à pouvoir y embarquer un micro-contrôleur émulant un clavier USB. Afin que cette souris se comporte comme une souris standard, les *pen-testers* de Netragard ont également embarqué un concentrateur USB dans la souris modifiée auquel étaient connectés les composants électroniques de la souris et le clavier malveillant. Du point de vue fonctionnel, le périphérique se comporte alors comme une souris classique. Celui-ci renferme cependant également un clavier malveillant. Une fois mis au point, ils ont usé d'ingénierie sociale pour infiltrer le périphérique malveillant chez le client. Pour cela, ils ont soigneusement emballé la souris dans son boitier d'origine et l'ont envoyée à un employé de l'entreprise cliente en lui laissant croire que c'était un cadeau promotionnel. Ne sachant pas que la souris contenait un clavier malveillant et la croyant inoffensive, l'employé la connecte sans hésitation à son poste. Une minute après la connexion du périphérique, le clavier malveillant commence à envoyer des frappes pour télécharger une porte dérobée depuis l'Internet

et l'installer sur le poste victime. Cette porte dérobée est ensuite utilisée par l'attaquant pour accéder au parc informatique de l'entreprise cliente.

2.3.2 Modification d'une ROM flashable d'extension PCI

Lorsqu'un ordinateur est mis sous tension, une procédure appelée *Power-On Self Test* (POST) est chargée d'initialiser la plate-forme matérielle. Au cours de cette procédure, le BIOS est copié dans la mémoire centrale depuis la ROM sur laquelle il était stockée. Cette image du BIOS est alors exécutée par le processeur, entre autres, pour détecter les différents contrôleurs d'entrées-sorties connectés au système informatique et préparer leur initialisation. Étant donné que chaque modèle de contrôleur d'entrées-sorties dispose d'une procédure d'initialisation qui lui est spécifique, les constructeurs implémentent généralement cette procédure dans une mémoire ROM additionnelle embarquée dans le contrôleur d'entrées-sorties. C'est le cas, en particulier, des contrôleurs d'entrées-sorties de type PCI ou PCI Express qui stockent cette procédure dans leurs ROM éventuellement sous la forme d'un code qui est exécuté par le BIOS. Ce code est alors directement écrit en langage assembleur pour les systèmes informatiques compatibles x86 ou dans un langage interprété par le BIOS (par exemple, *OpenBoot*) dans d'autres types de systèmes informatiques. Ce code est également responsable de l'auto-test de la carte et de l'ajout de routines d'interruption qui permettent au BIOS d'interagir avec des fonctions spécifiques au contrôleur d'entrées-sorties.

La plupart des ROM d'extension dans les contrôleurs d'entrées-sorties sont aujourd'hui reprogrammables. La procédure pour modifier leur contenu dépend de la technologie utilisée pour les mémoires. Les mémoires EPROM, par exemple, nécessitent en premier lieu que la puce soit effacée par une exposition prolongée à des rayonnements ultra-violets avant de pouvoir être reprogrammée. Étant donnée que chaque puce doit individuellement être retirée de sa carte pour être reprogrammée, la mise à jour des mémoires EPROM est lourde à mettre en place. C'est la raison pour laquelle de plus en plus de contrôleurs d'entrées-sorties utilisent des mémoires EEPROM qui, contrairement aux mémoires EPROM, peuvent être reprogrammées électriquement sans que la puce ne soit retirée de la carte. Dans ces circonstances, des outils sont mis à disposition par les constructeurs de contrôleurs PCI ou PCI Express pour reprogrammer la ROM d'extension du contrôleur d'entrées-sorties depuis le système d'exploitation. Il suffit donc que l'utilisateur dispose des privilèges d'un administrateur du système pour dialoguer directement avec le contrôleur d'entrées-sorties et reprogrammer sa ROM d'extension pour y insérer du code malveillant. Ce code malveillant sera alors exécuté par le BIOS à l'initialisation de la plate-forme et à chaque fois qu'une routine d'interruption du BIOS est appelée.

Typiquement, lors d'une attaque, l'attaquant commence par reprogrammer la ROM d'extension utilisée par le contrôleur d'entrées-sorties PCI ou PCI Express. En fonction du type de mémoire, cette reprogrammation est effectuée depuis le processeur principal via des applications dédiées (transition ❺) ou nécessite un dispositif externe (transition ❷). Lors de l'initialisation du système, le code malveillant inséré dans la ROM d'extension est recopiée automatiquement par le BIOS en mémoire centrale (transition ϵ). Le processeur exécute ce code malveillant qui peut alors altérer d'autres éléments du sous-système logiciel, en particulier le système d'exploitation (transition ϵ suivie de transitions ❺). Le principe de cette attaque a été publié dans [Heasman 07].

2.3.3 Modification de la routine de traitement de la SMI

Les processeurs x86 supportent plusieurs modes d'opération. Le mode d'opération nominal correspond au mode protégé pour les processeurs exploités en 32-bits et mode IA-32e pour les processeurs 64-bits. Ils fournissent un ensemble de fonctionnalités tels que les anneaux (en anglais, *rings*) qui répartissent les fonctions du processeur sur différents niveaux de privilège, les unités de segmentation et de pagination qui assurent la protection de la mémoire, etc. Le mode réel, quant à lui, permet une compatibilité descendante avec les processeurs Intel 8086. Il s'agit du mode d'opération initial des processeurs x86 et il incombe au système d'exploitation de passer ensuite dans le mode protégé. Le mode 8086 virtuel (*virtual-8086 mode*) contribue également à la compatibilité avec les processeurs Intel 8086. Il rend possible l'exécution de logiciels écrits pour ce type de processeur dans un environnement où chaque tâche dispose d'un espace d'adressage isolé. Finalement, le mode de gestion système (*System Management Mode* ou SMM) est une fonctionnalité architecturale standard de tous les processeurs x86. Il est dédié à l'exécution d'une routine d'interruption qui gère différents événements spécifiques à la plate-forme matérielle. La présente sous-section s'intéresse à ce dernier mode d'opération.

Le processeur entre dans le mode SMM à la réception d'une SMI (*System Management Interrupt*). Il s'agit d'une interruption matérielle générée par le *chipset* lorsque certains événements se produisent : le réveil d'un périphérique, la surchauffe du processeur, les erreurs physiques de la mémoire, le contrôle des différents ventilateurs, etc. Lorsque le processeur reçoit une SMI, il bascule automatiquement dans le mode SMM et sauvegarde en mémoire le contexte de la tâche qui s'exécutait jusqu'alors (en particulier l'état des registres de données et de contrôle du processeur), dans une région de la mémoire centrale appelée SMRAM (*System Management RAM*) mise en place par le BIOS à l'initialisation de la machine. Suite à cela, la routine de traitement de la SMI est alors exécutée. Celle-ci s'exécute de façon transparente et a accès à l'ensemble de la plate-forme matérielle : la mémoire physique de la machine (dans une limite de 4 Go), les registres de configuration ou de l'espace de mémoire propre des périphériques, etc. Un attaquant qui serait capable d'injecter du code dans cette routine de traitement de la SMI pourrait alors simplement prendre le contrôle de la machine.

Les accès à la SMRAM sont contrôlés par le *chipset*. La configuration de ce mécanisme repose sur le positionnement de certains bits des registres SMRAMC (*SMRAM Control*) et ESMRAMC (*Extended SMRAM Control*). Lorsque le bit `D_OPEN` du registre SMRAMC est positionné à 0, alors l'accès à la SMRAM n'est autorisé que depuis le mode SMM. S'il est à 1, la SMRAM est accessible depuis n'importe quel mode d'opération du processeur. Le BIOS est censé, après avoir copié la routine SMI dans cette région, positionner le bit `D_LCK` à 1. Cela provoque automatiquement la mise à 0 de `D_OPEN` et empêche toute modification des registres SMRAMC et ESMRAMC avant le prochain redémarrage de la machine. Depuis 2006, la plupart des concepteurs de BIOS utilisent à bon escient ce mécanisme de contrôle d'accès à la SMRAM. Aussi, il est aujourd'hui difficile de remplacer le code de cette routine d'interruption par du code malveillant et il est nécessaire, sur les plates-formes matérielles récentes, de trouver une autre technique pour contourner ce mécanisme de contrôle d'accès à la SMRAM.

En 2009, plusieurs auteurs [Wojtczuk et Rutkowska 09a, Duflot et Levillain 09] ont proposé, de manière indépendante, une technique d'attaque qui repose sur l'empoisonnement du cache des processeurs. Pour remplacer la routine de traitement de la SMI, l'attaquant commence par configurer la région de mémoire centrale où la SMRAM est située en *Write-Back* (WB) afin que toute écriture à cette région de mémoire soit automatiquement mise en cache (transition ❸). L'attaquant écrit ensuite du code malveillant dans cette région de mémoire dans le but de remplacer la routine de traitement de la SMI dans le cache (transition ❺ puis ϵ). Il ne reste plus,

pour l'attaquant, qu'à déclencher une SMI (transition ❸) afin que le processeur bascule en mode SMM et exécute la routine de traitement de la SMI. Le processeur récupère alors cette routine de traitement de la SMI à partir du cache et exécute le code malveillant fourni par l'attaquant (transition ❺).

2.4 Limites du modèle d'attaques

Notre travail de réflexion sur un modèle d'attaques capable de représenter les attaques élémentaires pour chaque niveau d'abstraction d'un système informatique a abouti à un nouveau formalisme de modélisation graphique d'attaques. La modélisation que nous proposons ne prétend aucunement remplacer les formalismes classiquement utilisés dans le domaine de la sécurité, tels que les arbres d'attaque [Schneier 99], les arbres de défaillances dynamiques (en anglais, *Dynamic Fault Trees* ou DFT) [Bechta Dugan *et al.* 90], les graphes d'attaques [Sheyner 04] ou les *Boolean logic Driven Markov Processes* (BDMP) [Bouissou et Bon 03]. Comparé aux autres formalismes, notre modèle nous a paru plus simple à mettre en œuvre afin de représenter et analyser différents scénarios envisageables par un attaquant, impliquant différents niveaux d'abstraction d'un système informatique, pour atteindre un ou plusieurs objectifs.

L'élaboration d'un modèle d'attaques est unanimement reconnue comme une problématique ardue et mériterait à elle seule que nous lui consacrions une thèse. Ainsi, le modèle que nous présentons dans ce chapitre est nécessairement incomplet et présente quelques limites.

Une des premières limites que nous pouvons signaler découle de la formalisation mathématique que nous avons établie pour définir une attaque. Dans celle-ci, nous avons considéré une attaque comme une succession d'actions élémentaires dont le séquencement suit un ordre total. Or, certaines actions élémentaires dans une attaque peuvent suivre un ordre partiel. Une attaque par éblouissement (cf. chapitre 1) noie, par exemple, les actions élémentaires constituant une attaque dans un nombre important d'actions élémentaires sans réelle relation avec la dite attaque. Ces dernières peuvent s'entrelacer de façon arbitraire avec les actions élémentaires constituant l'attaque dans la mesure où leurs effets n'impactent pas l'état du système ciblé. Considérer un ordre partiel sur les actions élémentaires n'est pas trivial dans notre modèle d'attaques car cela nécessite de définir quelles pré-conditions permettent d'effectuer une action élémentaire. À ce propos, nous gagnerions probablement en lisibilité et en puissance de modélisation si le modèle d'attaque était représenté sous la forme d'un réseau de Petri [Petri 62]. En effet, les réseaux de Petri sont des formalismes puissants qui permettent la spécification graphique d'interactions complexes tels que les conflits, la synchronisation, les boucles, les exclusions mutelles ou encore le partage de ressources. Entre autres, leur puissance de modélisation permettrait de rendre compte plus simplement de l'aspect séquentiel des attaques, des actions concurrentes, etc.

Une autre limite que nous avons identifiée est son passage à l'échelle. Le graphe de flots de données peut rapidement devenir complexe dès lors que nous ajoutons des composants matériels. Le modèle correspondant sera alors difficilement lisible et l'explosion combinatoire des chemins à explorer pour identifier les attaques deviendra rapidement problématique. Jusqu'à présent, nous n'avons pas de pistes intéressantes pour pallier ce problème.

Il convient de noter que nous aurions pu approfondir notre modèle en travaillant sur les limites mentionnées précédemment. Cependant, dans son état actuel, celui-ci est à un stade suffisament avancé pour nous permettre de nous focaliser sur l'étude des attaques par entrées-sorties qui constitue l'objectif principal de cette thèse. Nous avons envisagé d'élaborer davantage notre modèle d'attaques pour nos travaux futurs.

2.5 Conclusion

Dans ce chapitre, nous avons expliqué la démarche que nous avons adoptée pour élaborer un modèle d'attaques sur lequel il est possible de s'appuyer pour étudier les malveillances dans un système informatique. Le modèle que nous avons obtenu couvre des scénarios d'attaques qui impliquent éventuellement différents niveaux d'abstraction d'un système informatique. Dans notre démarche, nous avons considéré les composants matériels comme des systèmes à part entière qui interagissent avec d'autres composants matériels, donc d'autres systèmes, au travers d'une architecture de communication. Naturellement, nous avons modélisé ensuite leur structure interne sous la forme d'un graphe de flots de données et nous avons déduit, à partir des vecteurs d'attaques que nous avons identifiés au chapitre 1 (cf. section 1.2), l'ensemble des actions élémentaires (et des attaques élémentaires) possibles au sein de ceux-ci. En combinant cette vision microscopique avec les interactions entre les différents composants matériels, nous avons pu déduire de manière exhaustive l'ensemble des scénarios d'attaques qui sont possibles dans un système informatique. Nous avons appliqué cette démarche à un modèle de système informatique proche de ceux auxquels nos travaux s'intéressent, à savoir les systèmes informatiques compatibles PC. Enfin, afin de mieux appréhender le modèle d'attaques que nous avons obtenu, nous l'avons illustré par plusieurs exemples concrets d'attaques.

Au cours de l'élaboration de notre modèle d'attaque, nous avons privilégié une approche de construction du graphe d'attaque qui est manuelle. Cette approche est fastidieuse et potentiellement source d'erreurs. Pour ces raisons, nous avons envisagé de travailler sur un outil qui puisse générer automatiquement le modèle d'attaques d'un système informatique en se basant sur un moteur d'inférence écrit en Prolog [Deransart *et al.* 96]. Il sera alors possible de l'instancier pour le modèle de système informatique que nous avons considéré afin de compléter éventuellement le modèle d'attaques que nous avons établi en incluant des scénarios d'attaques que nous avons omis dans notre analyse manuelle.

Dans les deux chapitres qui suivent, nous allons mettre en œuvre plusieurs attaques issues de notre graphe d'attaque sur des systèmes informatiques compatibles PC. Nous allons nous concentrer en particulier sur les attaques par entrées-sorties sur lesquelles relativement peu de travaux existent dans la littérature. Ces attaques nous permettent alors d'introduire les contre-mesures matérielles possibles pour s'en protéger et de discuter de certaines de leurs limites que nous avons identifiées.

Chapitre 3

Mise en œuvre d'attaques par entrées-sorties sur une architecture PC

Dans le chapitre précédent, nous avons établi un modèle d'attaques qui peut s'appliquer à tout système informatique. Dans le présent chapitre, nous appliquons ce modèle à une architecture matérielle particulière, celle des PC, ce qui nous permettra, d'une part, d'étudier concrètement les différentes manières de mettre en défaut sa sécurité et, d'autre part, d'envisager plusieurs contre-mesures à celles-ci. L'architecture PC, créée en 1981 par la société *International Business Machines* (IBM), s'est imposée dans la majorité des systèmes informatiques grâce aux principes qui ont guidé sa conception : elle est standardisée et ouverte, relativement simple (du moins, elle l'était à ses débuts), bien documentée et conçue pour équiper spécifiquement des systèmes informatiques sur étagère appelés « ordinateurs personnels » (en anglais, *Personal Computers*). Aujourd'hui, le fabricant mondial de semi-conducteurs Intel, acteur majeur [15] sur le marché des processeurs et des *chipsets*, continue à faire évoluer cette architecture. Nous parlons alors plus spécifiquement d'architecture matérielle Intel-PC ou, plus simplement, d'architecture Intel-PC. Elle constitue, de nos jours, une des références pour les systèmes informatiques dits compatibles PC. C'est pour cette raison que ce chapitre se concentre spécifiquement sur celle-ci.

L'approche que nous présentons dans ce chapitre pour étudier les attaques issues de notre modèle s'apparente à l'approche traditionnelle, basée sur une analyse de vulnérabilités plutôt empirique : rechercher puis identifier une vulnérabilité, développer une preuve de concept qui permette de l'exploiter et enfin proposer des contre-mesures. Dans le prochain chapitre, nous proposerons une autre approche, plus systématique, de recherche des vulnérabilités.

Nous nous intéressons, en particulier, aux attaques par entrées-sorties, principalement parce qu'elles sont très difficiles à détecter et à contrer par logiciel, dans la mesure où elles ne mettent en jeu ni les processeurs principaux ni même, au moins dans certains cas, la mémoire centrale. Pour bien comprendre comment il est possible de mener des attaques par entrées-sorties sur une architecture PC, nous commençons par présenter, en section 3.1, quelques considérations techniques sur cette architecture matérielle. Nous rappelons, en particulier, les différents espaces d'adressage et les modes d'entrée-sortie qui sont supportés. À partir de là, nous nous intéressons spécifiquement à la capacité qu'ont les contrôleurs d'entrées-sorties à gérer eux-mêmes leurs transferts. Ce mode de communication n'étant pas sans risque sur la sécurité

[15] Une analyse [Trefis 12] publiée dans le magazine économique Forbes [Trefis Team 12] estime, en effet, qu'Intel a occupé en 2011 près de 84.3%, 73.8% et 94.5% de parts de marché respectivement des ordinateurs portables, des ordinateurs de bureau et des serveurs. Cette domination se maintient depuis près de vingt ans.

du système informatique, nous pointons la faiblesse structurelle à l'origine des attaques par entrées-sorties dans les architectures PC, à savoir le manque de contrôle d'accès sur les entrées-sorties. Nous nous mettons ensuite dans la peau d'un attaquant et nous décrivons plusieurs attaques de ce type que nous avons identifiées à l'aide de notre modèle d'attaques. Cela fait l'objet des sections 3.2 et 3.3. Pour chacune d'elles, nous présentons plusieurs contre-mesures – logicielles mais aussi et surtout matérielles – possibles et nous discutons de leurs limitations respectives. Les réflexions autour desquelles s'articulent ce chapitre ont été présentées à la journée Sécurité des Systèmes Logiciels & Sûreté des Logiciels (3SL) [Lone Sang *et al.* 11a] et au séminaire SysSec [Lone Sang *et al.* 11c] en 2011.

3.1 Aperçu de l'architecture Intel-PC

Dans la littérature, nous retrouvons plusieurs définitions différentes associées au terme *architecture matérielle*. Aussi nous paraît-il essentiel de le préciser. Dans un système informatique, le terme *architecture matérielle* couvre trois aspects du système. Historiquement, ce terme désignait uniquement le jeu d'instructions (en anglais, *instruction set*) des processeurs. Nous parlions alors, par exemple, d'architecture matérielle SPARC, PowerPC, ARM ou MIPS. Des évolutions architecturales ont ensuite été apportées pour que les processeurs puissent atteindre des fréquences plus élevées. La définition de ce terme a suivi cette évolution et s'est étendue pour y adjoindre la micro-architecture des processeurs. Le même jeu d'instructions Intel pouvait alors être implémenté par plusieurs (micro-)architectures matérielles de processeurs : Pentium, Core, Nehalem, Sandy Bridge, etc. Récemment, la définition associée à ce terme s'est davantage élargie et couvre aujourd'hui également l'infrastructure matérielle du système informatique.

Puisque ce chapitre est dédié aux attaques reposant sur les contrôleurs d'entrées-sorties, c'est tout d'abord à ce dernier aspect auquel nous nous intéressons. Puis, nous détaillons les différents espaces d'adressage ainsi que les modes d'entrée-sortie supportés par l'architecture Intel-PC. Le lecteur intéressé par d'autres aspects de cette architecture pourra consulter, en complément, le dossier que nous avons publié dans la revue de vulgarisation scientifique Multi-System & Internet Security Cookbook (MISC) [Chifflier *et al.* 11].

3.1.1 Infrastructure matérielle

La structure matérielle d'un système informatique de type PC se compose d'un circuit imprimé principal appelé carte-mère qui relie électriquement plusieurs composants matériels :

- un ou plusieurs processeurs ;
- la mémoire centrale, qui se matérialise physiquement sous la forme d'une ou de plusieurs barrettes composées de circuits intégrés qui constituent la mémoire ;
- des contrôleurs d'entrées-sorties internes, dont le rôle est d'étendre les bus d'entrées-sorties ou de piloter d'autres composants internes ou externes tels que les disques ;
- une ou plusieurs cartes d'extension (ou cartes-fille), qui embarquent un ou plusieurs contrôleurs d'entrées-sorties externes. Elles sont enfichées sur des connecteurs spécifiques reliés à un ou plusieurs bus d'extension tels quels les bus PCI et les bus PCI Express ;
- plusieurs circuits intégrés, qui sont directement soudés sur la carte-mère pour réaliser des fonctions annexes tels que le stockage de la configuration du matériel et l'horloge ;
- un ou plusieurs périphériques, qui se connectent aux contrôleurs de périphériques à l'aide de connecteurs spécifiques reliés à des bus de périphériques tels que les bus USB

et les bus FireWire.

Au sein de la carte-mère, les différents composants communiquent au travers d'un ensemble de puces électroniques appelé *chipset*. Il interconnecte le ou les processeurs à d'autres composants matériels tels que la mémoire centrale, la carte graphique, la carte réseau, les contrôleurs de disques, les périphériques, etc. La plupart des *chipsets* d'Intel (et, jusqu'à récemment, ceux de ses concurrents) reposent sur une infrastructure à plusieurs niveaux incorporant une partie *northbridge* (littéralement, pont du nord) et une partie *southbridge* (littéralement, pont du sud) reliés entre eux par un bus propriétaire parfois appelé *Direct Media Interface* (DMI). La figure 3.1 présente une organisation possible de ces composants matériels.

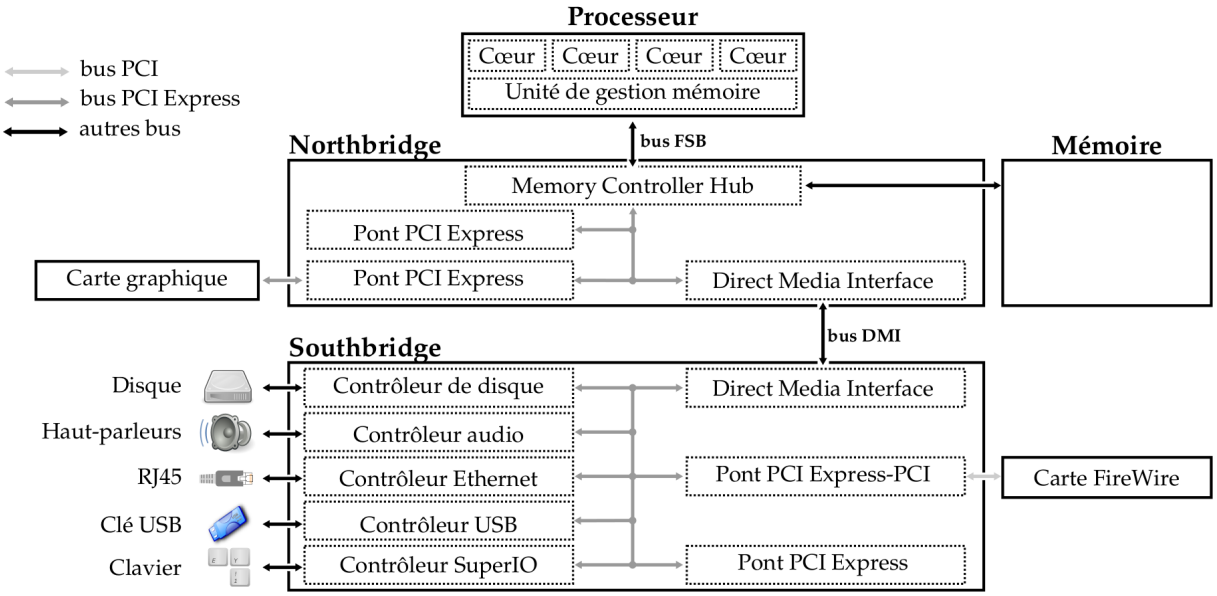

FIGURE 3.1 – Exemple d'architecture matérielle Intel-PC (*chipset* Q45)

Le *northbridge* relie le ou les processeurs et la mémoire centrale aux différents contrôleurs d'entrées-sorties. 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 (typiquement, les contrôleurs graphiques), ou au travers du *southbridge*, lorsque ceux-ci nécessitent moins de ressources. Sur les anciennes cartes-mère, le *northbridge* pouvait comprendre jusqu'à trois puces. Aujourd'hui, il n'est constitué que d'une seule puce qui tend même à être directement intégrée dans les processeurs. Par exemple, les processeurs Intel (à partir de Nehalem et surtout Sandy Bridge) et les processeurs AMD (Fusion) actuels intègrent désormais le contrôleur de mémoire et quelques contrôleurs d'entrées-sorties rapides tels que les contrôleurs graphiques.

Le *southbridge* assure la connexion entre la plupart des contrôleurs d'entrées-sorties et le *northbridge*. Il fournit plusieurs interfaces qui permettent de communiquer sur différents types de bus tels que les bus USB, les bus FireWire ou les bus PCI auxquels peuvent être connectés d'autres contrôleurs d'entrées-sorties (internes ou externes) et des périphériques. La communication avec ces bus s'effectue au travers de ponts (par exemple, un pont PCI Express-PCI) ou au travers de contrôleurs de bus (tels que des contrôleurs USB, des contrôleurs FireWire). Le *southbridge* a toujours été constitué d'une seule puce et il est, d'une certaine manière, interchangeable. En effet, un même modèle de puce pour le *northbridge* est compatible avec différents modèles de puce pour le *southbridge*, permettant ainsi aux fabricants de cartes-mère de diversi-

fier leurs offres. Pour cette raison, la puce utilisée pour le *northbridge* donne souvent son nom au *chipset*. Ainsi, le *chipset* Q45 représenté sur la figure 3.1 correspond à la puce de *northbridge* 82Q45.

Les sous-sections suivantes se concentrent sur la manière dont ces composants matériels interagissent. Nous décrivons les espaces d'adressage définis dans l'architecture PC (§3.1.2) et nous rappelons les principaux modes d'entrée-sortie dans les systèmes informatiques (§3.1.3).

3.1.2 Espaces d'adressage

Les architectures PC définissent trois espaces d'adressage distincts (cf. figure 3.2) : l'espace mémoire, l'espace d'entrée-sortie et l'espace de configuration des contrôleurs PCI et PCI Express. Chaque espace d'adressage dispose d'un mécanisme d'accès qui lui est spécifique.

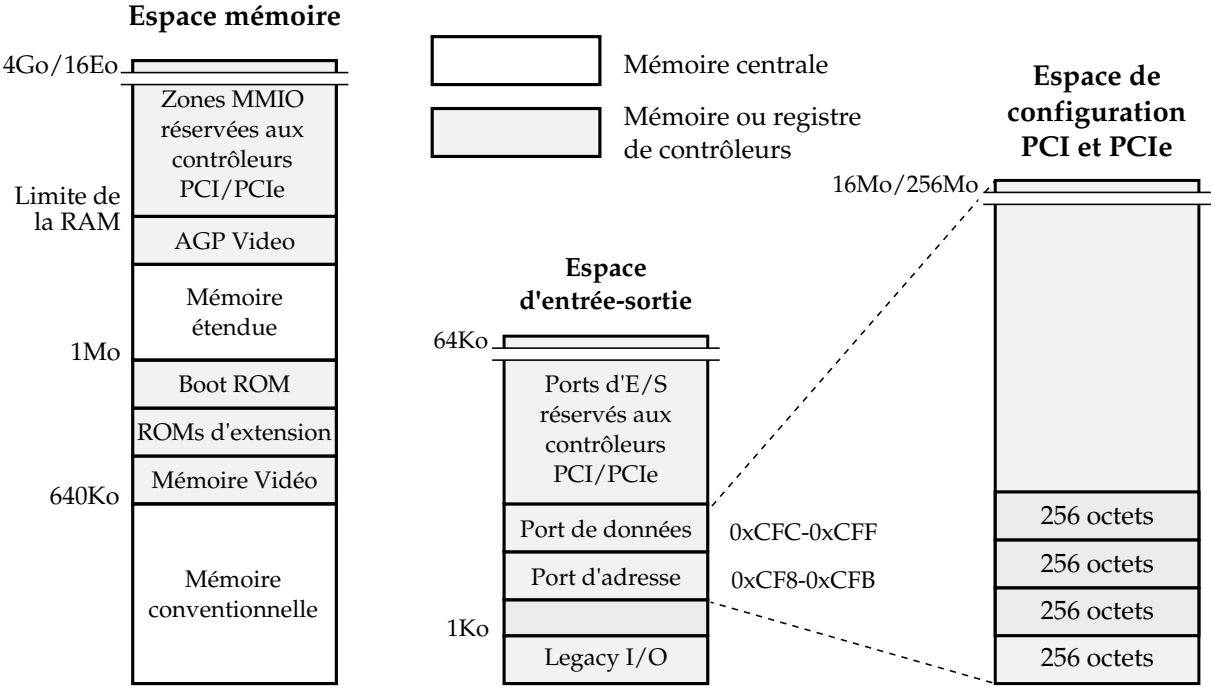

FIGURE 3.2 – Espaces d'adressage définis dans les architectures PC

3.1.2.1 Espace mémoire

L'espace mémoire (en anglais, *memory space*) peut s'étendre jusqu'à 4 gigaoctets dans les systèmes informatiques qui supportent un mode d'adressage de la mémoire sur 32 bits, et jusqu'à 16 exaoctets sur ceux qui supportent un mode d'adressage de la mémoire sur 64 bits. Les accès à cet espace d'adressage utilisent des adresses physiques [16] et sont, pour la plupart, aiguillés vers le contrôleur de mémoire (*Memory Controller Hub*). Elles concernent, par conséquent, principalement les accès à la mémoire centrale. Cependant, plusieurs contrôleurs d'entrées-sorties

[16] Il est important ici de distinguer les adresses virtuelles et les adresses physiques. Les adresses manipulées par les programmes correspondent à des adresses virtuelles et sont utilisées pour accéder à une mémoire virtuelle. Les unités de gestion de la mémoire situées dans les processeurs convertissent systématiquement ces adresses virtuelles en adresses physiques pouvant être manipulées par les contrôleurs d'entrées-sorties. Cette conversion nécessite des tables de traduction – les tables des pages (en anglais, *page tables*) – stockées en mémoire centrale.

peuvent nécessiter d'y projeter des registres ou de portions de leur mémoire interne (RAM, ROM, EEPROM, etc.) afin, d'une part, de faciliter les transferts d'entrées-sorties avec ces composants et, d'autre part, de les accélérer. Le *chipset* réserve alors à ces contrôleurs d'entrées-sorties des fragments de cet espace d'adressage et le contrôleur de mémoire dans le *northbridge*, dans son rôle de répartiteur, rend les accès à ces zones d'entrées-sorties transparents depuis les processeurs en les aiguillant vers les contrôleurs d'entrées-sorties concernés : les processeurs y accèdent comme ils accèderaient à la mémoire centrale. Il s'agit du mécanisme de *Memory Mapped I/O* (MMIO). Afin d'éviter tout conflit avec la mémoire centrale [17], ces zones d'entrées-sorties sont généralement positionnées par le gestionnaire de la plate-forme matérielle – plus précisément, le *Basic Input/Output System* (BIOS) – dans les adresses hautes de l'espace mémoire, au-delà des plages réservées à la mémoire centrale.

3.1.2.2 Espace d'entrée-sortie

L'espace d'entrée-sortie (en anglais, *I/O space*) est dédié aux entrées-sorties. Il s'agit d'un espace d'adressage logiquement indépendant de l'espace mémoire. Il peut s'étendre théoriquement jusqu'à 4 gigaoctets. Cependant, de nombreuses plates-formes matérielles restreignent l'accès à cet espace d'adressage uniquement aux 64 premiers kilooctets. Cette restriction était initialement due à la limite d'adressabilité de cet espace dans les premiers processeurs Intel 8086 (16 bits). Elle a été maintenue jusque dans les processeurs récents compatibles x86 malgré leur capacité d'adressage plus grande (32 ou 64 bits). Historiquement, cet espace d'adressage servait à interroger, au travers de registres, des composants matériels divers : les contrôleurs de disques *Advanced Technology Attachment* (ATA), les ports série et les ports parallèle, les claviers implémentant le standard PS/2 et les souris, les manettes de jeu vidéo, etc. Les registres projetés dans cet espace sont appelés plus spécifiquement des ports d'entrées-sorties justifiant ainsi le nom attribué à ce mécanisme : *Port Mapped I/O* (PMIO) ou plus simplement *Port I/O* (PIO). Aujourd'hui, cet espace d'adressage reste très peu usité à cause de la lenteur des transferts d'entrées-sorties qu'implique son utilisation et n'est gardé que pour des raisons de rétro-compatibilité. En effet, l'espace d'adressage étant de taille restreinte, les portions de mémoire interne des contrôleurs d'entrées-sorties ne peuvent pas être directement projetées dans cet espace. Pour pallier cette limite, les contrôleurs d'entrées-sorties définissent un mécanisme de fenêtrage qui, au lieu de projeter l'intégralité de la mémoire interne dans cet espace, projette uniquement une sous-partie spécifique de cette mémoire interne. Ce mécanisme consomme ainsi moins de resources, mais requiert plus de cycles du processeur (et par-là même, plus de temps) dans la mesure où la fenêtre définie par le contrôleur d'entrées-sorties est petite et que la quantité de données à transférer est importante. Les performances d'entrées-sorties moindres par rapport aux accès à l'espace mémoire s'expliquent également par les modes d'accès qu'ils supportent respectivement : l'espace d'entrée-sortie ne supporte que des accès par blocs de taille fixe (de 1, 2 ou 4 octets) alors que l'espace mémoire autorise des accès par blocs de taille variable pouvant aller théoriquement jusqu'à 4096 octets.

3.1.2.3 Espace de configuration des contrôleurs PCI et PCI Express

La configuration matérielle des premiers systèmes informatiques compatibles PC était une opération souvent délicate et source de problèmes importants. En effet, ils nécessitaient d'affec-

[17] Sur des architectures PC plus anciennes, il arrivait que le *chipset* réserve une partie des adresses utilisées pour la mémoire centrale au mécanisme de MMIO, empêchant ainsi son utilisation. Ce phénomène était dû à une limite d'adressabilité des processeurs de l'époque. Ce problème n'est plus d'actualité avec les processeurs 64 bits actuels.

ter manuellement, à chaque composant matériel, les ressources nécessaires à leur bon fonctionnement (par exemple, les plages d'adresses, les lignes d'interruption, etc.) et de résoudre les conflits qui pouvaient en résulter. L'espace de configuration des contrôleurs PCI et PCI Express a été introduit spécifiquement pour résoudre ce problème. Il s'agit d'un espace d'adressage dans lequel sont projetés les registres de configuration des contrôleurs PCI et PCI Express. Leur configuration, qui était auparavant manuelle, est maintenant laissée au BIOS à l'initialisation du système. Cet espace de configuration peut s'étendre jusqu'à 16 mégaoctets pour les architectures avec des bus PCI et jusqu'à 256 mégaoctets pour celles qui contiennent des bus PCI Express.

L'accès à cet espace d'adressage nécessite une adresse particulière formée par la concaténation, d'une part, de l'identifiant de bus associé à un contrôleur d'entrées-sorties et, d'autre part, de l'index du registre auquel on accède dans ce contrôleur. Cet identifiant de bus est nécessaire au *chipset*, entre autres, pour repérer physiquement le contrôleur sur les bus d'entrées-sorties et pour aiguiller l'accès de configuration correctement vers celui-ci. Il se compose de trois parties :

— un numéro de bus, qui est initialisé par le BIOS au démarrage du système et qui précise le bus d'entrées-sorties auquel le contrôleur est physiquement connecté ;
— un numéro de composant (pour le terme anglo-saxon *device*), qui localise le contrôleur dans un composant matériel (carte d'extension, circuit intégré, etc.) de ce bus ;
— et un numéro de fonction, qui identifie le contrôleur d'entrées-sorties dans ce composant. Il existe aujourd'hui des cartes d'extension qui embarquent plusieurs contrôleurs et qui fournissent des services différents. À titre d'exemple, une carte d'extension combo eSATA-USB embarque à la fois un contrôleur de bus USB et un contrôleur de bus eSATA. Le numéro de fonction joue, en quelque sorte, le rôle de multiplexeur et permet d'interroger l'un ou l'autre des deux contrôleurs.

Dans la littérature, cet identifiant est communément appelé identifiant BDF (en référence aux numéros de *bus*, de *device* et de *function*) d'un contrôleur d'entrées-sorties. Chaque contrôleur d'entrées-sorties prend connaissance de l'identifiant BDF qui lui est associé par les requêtes d'entrées-sorties qui lui parviennent, lesquelles transportent généralement cette information.

La figure 3.3 présente l'agencement de l'espace de configuration d'un contrôleur PCI. Les 64 premiers octets de cet espace sont organisés de manière standard et les octets restants sont à la discrétion du fabricant du contrôleur. Il convient de remarquer que l'organisation de ces 64 octets est identique pour un contrôleur PCI Express, ce qui contribue à la compatibilité logicielle entre les deux protocoles de bus. Les systèmes d'exploitation s'appuient sur ces octets pour identifier et charger les pilotes (en anglais, *device drivers*) de contrôleur d'entrées-sorties adéquats. Les deux premiers registres, `Vendor ID` et `Device ID`, sont essentiels au fonctionnement de ce processus. Ils indiquent respectivement le matricule attribué au fabricant du contrôleur d'entrées-sorties et le matricule du contrôleur d'entrées-sorties lui-même. Ces deux matricules combinés identifient de manière unique un modèle de contrôleur d'entrées-sorties. À présent, prêtons une attention particulière aux registres `Base Address Registers` (BAR) qui contribuent à la mise en œuvre des mécanismes de MMIO et de PMIO. Cet ensemble de six registres permettent d'affecter dynamiquement les ressources dans l'espace d'entrée-sortie ou dans l'espace mémoire nécessaires aux contrôleurs PCI ou PCI Express. À la mise sous tension du système informatique, chacun de ces registres indique au BIOS l'espace d'adressage dans lequel il souhaite projeter des registres, et la taille de mémoire ou de ports contigus qu'il nécessite. Une fois que les bus ont été cartographiés et que tous les contrôleurs ont été interrogés, le BIOS calcule un agencement possible des différents espaces d'adressage. Il attribue alors à chaque contrôleur les ressources souhaitées et les informe des plages d'adresses qui leur ont été attribuées en écrivant dans ces mêmes registres.

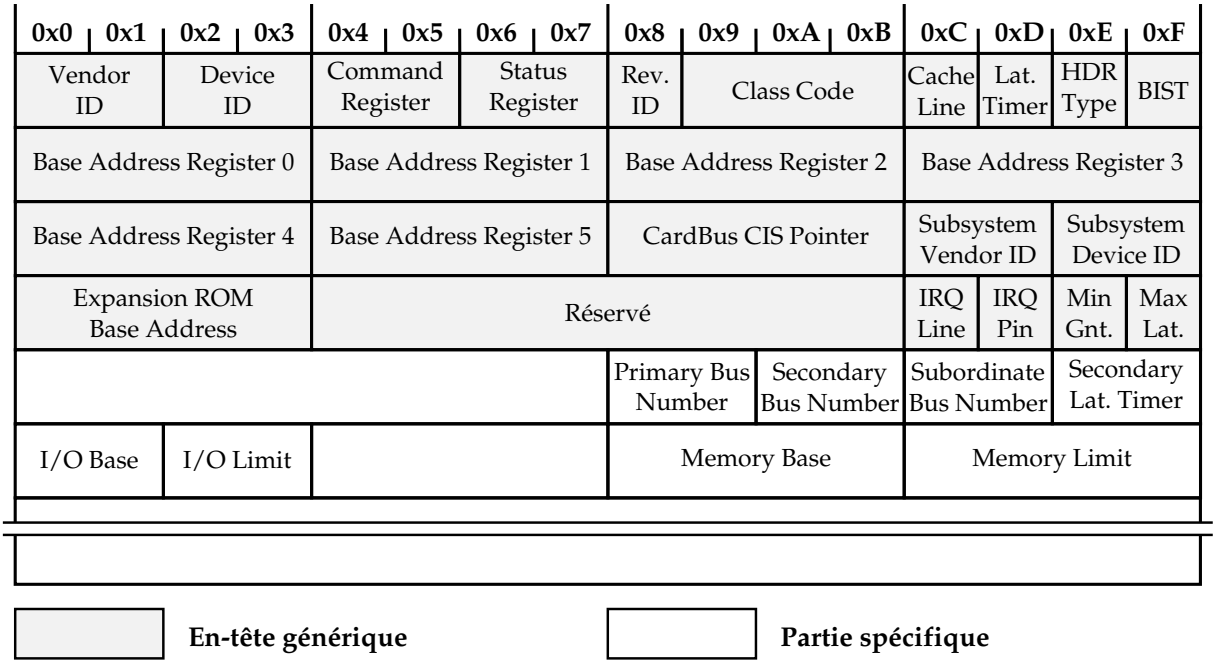

FIGURE 3.3 – Espace de configuration d'un contrôleur PCI

Notons que de nombreux ouvrages ne considèrent que deux espaces d'adressage : l'espace mémoire et l'espace d'entrée-sortie. L'espace de configuration des contrôleurs PCI et PCI Express est présenté comme un sous-espace des deux autres espaces d'adressage. Cette vision des espaces d'adressage est historique. En effet, rappelons que l'espace de configuration des contrôleurs PCI et PCI Express a été introduit pour permettre une configuration automatique des composants matériels. Pour maintenir une compatibilité ascendante entre les processeurs, il a été décidé de ne pas définir d'instructions dédiées aux accès à cet espace d'adressage. Les accès à celui-ci, depuis les processeurs, se font aujourd'hui soit par un mécanisme de fenêtrage utilisant deux ports d'entrées-sorties localisés aux adresses `0xCFC-0xCFF` (registre de données) et `0xCF8-0xCFB` (registre d'adresse), soit par un fragment de l'espace mémoire réservé à cet effet. Ces accès, tout comme les accès à une zone quelconque de l'espace mémoire ou de l'espace d'entrée-sortie, sont traités par le répartiteur dans le *northbridge*. Celui-ci se charge d'émettre sur les bus d'entrées-sorties les requêtes d'entrées-sorties appropriées. La vision proposée dans ces ouvrages sied, par conséquent, à celle des processeurs mais ne correspond pas à celle des bus d'entrées-sorties dans laquelle nous distinguons réellement trois espaces d'adressage.

3.1.3 Modes d'entrée-sortie

Nous identifions jusqu'à trois modes d'entrée-sortie dans tout système informatique : le mode des entrées-sorties programmées, le mode *bus-mastering I/O* et les interruptions. Cette sous-section présente succinctement leurs principales caractéristiques.

3.1.3.1 Entrées-sorties programmées

Les entrées-sorties programmées (en anglais, *programmed I/O* [18]) désignent un mode d'entrée-sortie synchrone dans lequel les processeurs initient et gèrent eux-mêmes les transferts de données avec les contrôleurs d'entrées-sorties [19]. Au sein d'un programme, ces transferts d'entrées-sorties sont mis en œuvre à l'aide d'instructions spécifiques des processeurs. À titre d'exemple, les transferts de données dans l'espace d'entrée-sortie nécessitent l'utilisation d'instructions dédiées : les instructions `in` (respectivement, `out`) en assembleur pour des opérations de lecture (respectivement, d'écriture). *A contrario*, les transferts de données vers l'espace mémoire ne requièrent pas d'instructions dédiées et se font simplement en utilisant les instructions `mov` en assembleur.

3.1.3.2 *Bus-mastering I/O*

Les transferts de données vers l'un des espaces d'adressage peuvent également être gérés par certains contrôleurs d'entrées-sorties. C'est le cas, en particulier, des contrôleurs dits *bus-masters* qui sont capables – à leur propre initiative, à l'initiative d'un processeur ou d'un périphérique – de prendre le contrôle des bus d'entrées-sorties et d'effectuer sur ces bus un transfert de données vers l'un des espaces d'adressage. Ce mode d'entrée-sortie asynchrone soulage les processeurs de ces transferts de données et leur permet d'effectuer d'autres tâches pendant que les transferts se déroulent. Actuellement, de nombreux contrôleurs d'entrées-sorties sont capables d'effectuer des accès vers l'espace mémoire, notamment pour échanger des données avec la mémoire centrale. Nous parlons alors d'accès direct à la mémoire (*Direct Memory Access*, ou DMA). Il convient de préciser que, même si en pratique cela est assez peu usité, certains contrôleurs d'entrées-sorties utilisent aussi ce mode d'entrée-sortie pour dialoguer avec d'autres contrôleurs d'entrées-sorties. Nous parlons alors d'accès pair-à-pair (en anglais, *peer-to-peer*). Certaines cartes graphiques récentes, utilisées pour des calculs à hautes performances, peuvent s'en servir pour dialoguer directement avec d'autres cartes graphiques sans utiliser les processeurs comme intermédiaires. Nous détaillons davantage ces deux modes de communication DMA et pair-à-pair dans les prochaines sections.

3.1.3.3 Interruption

Ce mode d'entrée-sortie n'est pas en soi un mécanisme de transfert de données. Néanmoins, il est intéressant de l'évoquer car il permet aux processeurs de répondre à des événements extérieurs de façon optimale. Une requête d'interruption peut être émise par un contrôleur d'entrées-sorties, par exemple, lorsqu'une donnée est disponible dans un périphérique. Lorsqu'une telle requête est reçue, le processeur concerné s'interrompt et sauvegarde le contexte d'exécution de la tâche qui vient d'être interrompue dans une pile. Il exécute ensuite une routine particulière chargée de traiter cette requête d'interruption. Finalement, il restaure le contexte d'exécution de la tâche interrompue pour reprendre son fil d'exécution là où il s'était arrêté.

Certaines attaques sont susceptibles de détourner ces modes d'entrée-sortie à des fins malveillantes. Nous parlons alors d'attaques par entrées-sorties. Quelques exemples de ce type

[18] Il est essentiel de ne pas confondre ce mode d'entrée-sortie avec le mécanisme d'entrées-sorties *Port I/O*.

[19] Il convient de remarquer que les transferts de données avec les périphériques ou d'autres composants matériels (par exemple, les mémoires RAM, les mémoires flash, etc.) se font nécessairement au travers d'un contrôleur d'entrées-sorties qui aura alors mis à disposition des mécanismes qui facilitent ces échanges.

d'attaques apparaissent dans la littérature. Ainsi, la preuve de concept d'attaque publiée dans [Wojtczuk *et al.* 09] détourne les entrées-sorties programmées dans le but de modifier la valeur d'un registre (le registre `MCHBAR` du *chipset*) sur lequel repose l'exécution d'un programme privilégié (le code SINIT nécessaire à la mise en œuvre de la technologie Intel TXT). L'exploit a alors consisté à positionner ce registre à une valeur bien particulière dans le but de modifier le flot d'exécution de ce programme privilégié. Dans [Duflot 07, Rutkowska et Wojtczuk 08], les auteurs présentent également un autre type d'attaque basé sur les entrées-sorties programmées. Avec les seuls privilèges d'entrées-sorties, ils exécutent un code malveillant chargé de paramétrer plusieurs mécanismes de traduction d'adresses (l'ouverture graphique AGP et le *memory reclaiming*) mis à disposition par le *chipset*. Ils les configurent alors de façon à rediriger les accès à une région de mémoire arbitraire, ne nécessitant pas de privilèges particuliers, vers des régions de mémoire privilégiées dont l'accès est contrôlé par le système d'exploitation par exemple. Puisque ces mécanismes de traduction d'adresses sont transparents pour le système d'exploitation (et pour les unités de gestion de la mémoire dans les processeurs), tout se passe comme si l'accès se faisait vers une région de mémoire arbitraire, alors qu'il s'effectue en réalité vers une région de mémoire privilégiée. Ce subterfuge leur a permis d'acquérir davantage de privilèges dans le système, notamment ceux du noyau de système d'exploitation.

La suite de ce chapitre se focalise plus particulièrement sur les attaques qui exploitent le mode d'entrée-sortie *bus-mastering I/O*. Dans ce mode d'entrée-sortie, certains transferts de données sont laissés aux contrôleurs d'entrées-sorties de façon à décharger les processeurs de ces opérations, et ceux-ci peuvent alors se concentrer sur d'autres traitements. Lorsqu'une telle responsabilité est laissée aux contrôleurs d'entrées-sorties, il paraît essentiel, pour la sécurité du système informatique, de s'assurer que ces transferts se déroulent correctement et que des transferts frauduleux ne sont pas susceptibles d'être mis en place à l'insu d'un utilisateur. Or, jusqu'à très récemment, ni les processeurs, ni le *chipset* n'étaient en mesure d'opérer ces vérifications. Ce manque de contrôle d'accès sur les bus d'entrées-sorties fait l'objet des deux prochaines sections. Nous présentons plusieurs attaques qu'il est possible de mener avec des contrôleurs d'entrées-sorties sur étagère et nous discutons de contre-mesures envisageables à chacune d'elles. Dans un souci de concision et de clarté, nous allons considérer principalement les attaques qui ciblent l'espace mémoire. Les concepts sur lesquels elles reposent étant génériques, ils pourront être transposés sans peine à d'autres espaces d'adressage. Ainsi, la section 3.2 est dédiée spécifiquement aux attaques par entrées-sorties qui ciblent la mémoire centrale. Le cas des attaques par entrées-sorties qui ciblent la mémoire interne d'autres contrôleurs d'entrées-sorties sera traité dans la section 3.3.

3.2 Contrôle des accès directs à la mémoire centrale

La plupart des attaques par entrées-sorties référencées dans la littérature ciblent la mémoire centrale. La présente section commence par rappeler le principe des accès directs à la mémoire centrale et détaille le fonctionnement des attaques qui en abusent. Nous présentons ensuite plusieurs contre-mesures matérielles qui ont été proposées et nous détaillons l'une d'entre elles pour les architectures PC récentes. Avec un exemple d'attaque à l'appui, nous montrons que cette contre-mesure, à elle seule, n'est pas suffisante pour contrôler l'ensemble des entrées-sorties et nous préconisons l'utilisation d'une autre contre-mesure qui lui est complémentaire.

3.2.1 Attaques par accès direct à la mémoire

L'accès direct à la mémoire ou DMA (*Direct Memory Access*) est un mécanisme où la gestion des transferts de données vers la mémoire centrale est laissée aux contrôleurs d'entrées-sorties. Les processeurs ne sont notifiés qu'*a posteriori* par une interruption qui les prévient de la fin du transfert. Pour accomplir ces transferts, un contrôleur d'entrées-sorties endosse temporairement le rôle de maître sur les bus d'entrées-sorties de façon à en prendre le contrôle et à émettre sur ceux-ci les requêtes d'accès correspondantes. Jusqu'à très récemment, ces transferts se réalisaient de manière transparente pour le système d'exploitation. Ni les processeurs, ni le *chipset* n'étaient alors en mesure de contrôler les adresses vers lesquelles s'effectuaient ces transferts et, encore moins, de bloquer les éventuels accès frauduleux. Des attaquants ont naturellement profité de cette faiblesse structurelle pour violer les propriétés de sécurité (intégrité et confidentialité) des composants logiciels chargés en mémoire centrale. L'accès à certaines régions de mémoire (par exemple, celles contenant le code du noyau de système d'exploitation) pouvant être restreint par les processeurs, il était ainsi plus simple, pour un attaquant, de mettre en place ces accès avec le concours d'un contrôleur d'entrées-sorties. En effet, de tels accès ne peuvent pas être contrôlés par un logiciel de détection ou de protection étant donné qu'ils ne nécessitent pas l'intervention des processeurs principaux. Nous parlons alors d'attaques par accès direct à la mémoire, en référence au type d'accès d'entrées-sorties sur lequel elles s'appuient. Chaque contrôleur d'entrées-sorties implémente une manière de configurer un transfert de données qui lui est spécifique. Toutefois, cette configuration s'effectue depuis un processeur, depuis un périphérique, voire depuis le contrôleur d'entrées-sorties lui-même.

3.2.1.1 Attaques DMA initiées par les processeurs

Les attaques par accès direct à la mémoire dont les transferts sont initiés par les processeurs nécessitent l'exécution d'un code malveillant par le processeur principal. Une attaque typique consiste à mettre en place, dans un premier temps, des structures de contrôle en mémoire centrale qui décrivent les transferts à effectuer : l'adresse, la direction [20] et la taille de chacun d'eux doivent notamment être précisées. Le logiciel malveillant précise ensuite au contrôleur d'entrées-sorties, en positionnant un registre prévu à cet effet, la localisation de la structure de contrôle concernée, et le contrôleur exécutera les transferts programmés à la réception d'un événement spécifique (par exemple, une interruption). Ces transferts peuvent cibler une adresse quelconque de la mémoire centrale. En particulier, il peut être intéressant, pour un attaquant, de cibler le noyau de système d'exploitation (code ou tables de configuration, en particulier les tables de gestion de la mémoire). En effet, corrompre le système d'exploitation signifie potentiellement corrompre l'ensemble des applications qui s'exécutent au dessus de ce noyau.

Les preuves de concept d'attaques qui reposent sur ce mode opératoire mettent en œuvre des contrôleurs d'entrées-sorties variés. Celle publiée dans [Duflot 07] décrit, par exemple, une escalade de privilèges à l'aide de contrôleurs de périphériques USB : l'attaquant met en place plusieurs transferts de données dans le contrôleur USB de façon à lui faire écraser des régions précises de la mémoire centrale (la région de données contenant la valeur du *securelevel* [21] dans un système OpenBSD) par les données fournies par les périphériques USB. Les transferts sont

[20] Nous distinguons deux directions de transfert de données possibles : un transfert de la mémoire centrale vers le contrôleur d'entrées-sorties ou l'inverse, c'est-à-dire du contrôleur d'entrées-sorties vers la mémoire centrale.

[21] Dans un système de type BSD, cette variable indique le niveau de sécurité mis en œuvre par le noyau de système d'exploitation. Des restrictions au sein du système (par exemple, accès aux périphériques virtuels, opérations autorisées sur les disques, etc.) sont instaurées en fonction de sa valeur. Plus celle-ci est élevée, plus elle implique de restrictions.

exécutés par le contrôleur de périphériques USB à la détection d'un événement sur l'un de ses bus (la ré-initialisation d'un bus USB, le branchement d'un périphérique USB, etc.).

Les contrôleurs réseau peuvent, de la même façon, être détournés à des fins d'attaques. Pour atteindre des bandes passantes de l'ordre du gigabit sur les liens réseau, ceux-ci reposent, en particulier, sur le mécanisme d'accès direct à la mémoire pour échanger des trames Ethernet (reçues ou à émettre) avec la mémoire centrale. Les attaques typiques impliquant ces contrôleurs nécessitent de reconfigurer leurs transferts de données de façon à modifier une région de mémoire avec des données reçues par le réseau ou, au contraire, à exfiltrer des données stockées dans la mémoire centrale vers le réseau. En fonction du contrôleur réseau qu'il utilise, un attaquant peut rencontrer quelques difficultés d'ordre technique. Un transfert de données dans un contrôleur réseau correspond soit à la trame qui vient d'être reçue depuis le réseau et que le contrôleur réseau copie dans la mémoire centrale, soit à une trame qui a été construite par le système d'exploitation et que le contrôleur réseau va lire en mémoire centrale puis émettre sur le réseau. Lorsqu'un attaquant détourne l'un de ces transferts pour modifier la mémoire centrale par exemple, il doit s'assurer que les données qu'il souhaite y inscrire correspondent à une trame valide, notamment pour que celle-ci puisse être commutée correctement au travers du réseau jusqu'au contrôleur réseau sur lequel repose l'attaque. Un problème analogue existe lorsqu'un attaquant s'en sert pour exfiltrer des données vers le réseau : les données lues en mémoire doivent également correspondre à une trame valide afin que celle-ci puisse être commutée au travers du réseau jusqu'à l'attaquant. Ces contraintes techniques rendent difficile l'exploitation de ces transferts depuis un contrôleur réseau car l'attaquant n'a pas une maîtrise sur l'intégralité des données.

Dans [Wojtczuk et Rutkowska 11b], les auteurs proposent une solution intéressante à ce problème. Elle consiste à détourner le mécanisme de *scatter-gather* embarqué dans les contrôleurs réseau récents dans le but de diviser les transferts de données en transferts plus petits. L'attaquant peut alors s'arranger, en configurant ce mécanisme, pour que les octets sur lesquels il n'a aucune maîtrise (par exemple, les entêtes de la trame Ethernet et du paquet IP) soient lus depuis (ou copiés vers) une région de mémoire arbitraire, et que les octets restants sur lesquels il a une maîtrise complète soient lus depuis (ou copiés vers) la région souhaitée de la mémoire centrale. Dans leur preuve de concept, ils mettent en œuvre cette solution pour modifier à leur guise les régions de la mémoire centrale, depuis un contrôleur réseau Intel e1000e. Ce modèle de contrôleur réseau est aujourd'hui présent dans la majorité des *chipsets* Intel. L'attaque a consisté à envoyer des paquets réseau *ping* malveillants : les octets correspondant aux en-têtes des trames Ethernet, des paquets IP et des paquets *ping* sont copiés par le contrôleur réseau dans les tampons de réception du système d'exploitation (afin de le leurrer et le laisser croire en la réception de paquets réseau normaux) alors que les octets restants sont copiés dans une région souhaitée de la mémoire centrale et écrasent par exemple des structures de contrôle du système d'exploitation.

3.2.1.2 Attaques DMA initiées par les périphériques

Les attaques par accès direct à la mémoire dont les transferts sont initiés par les périphériques requièrent au préalable un accès physique au système cible. En effet, elles nécessitent la modification d'un périphérique de façon à ce qu'il puisse se comporter à la fois en maître et en esclave sur son bus et non plus uniquement en esclave comme est supposé se comporter un périphérique classique. Un attaquant exploite ensuite la capacité de ce périphérique à prendre le contrôle de son bus afin de soumettre au contrôleur de périphérique des commandes pour mettre en place des transferts de données avec la mémoire centrale. Aujourd'hui, de nombreux

standards de bus tels que le FireWire [22] et certaines variantes de l'USB, motivés par des exigences de performances de plus en plus élevées, autorisent les périphériques à se comporter à la fois en tant que maître et en tant qu'esclave. Ceux-ci sont susceptibles d'être utilisés à des fins d'attaques. M. Dornseif a été pionnier dans ce domaine. Il a présenté, à l'occasion des conférences PacSec [Dornseif 04] et CanSecWest [Becher *et al.* 05], plusieurs preuves de concept d'attaques consistant à relier par un câble FireWire, un système BSD, Linux ou Mac OS à un iPod sur lequel avait été installé un système d'exploitation Linux modifié. L'exploit a consisté à utiliser cet iPod pour lire ou modifier l'intégralité de la mémoire centrale du système auquel il était connecté. Ses travaux ont été complétés par [Boileau 06], qui a proposé une technique consistant à usurper l'identité d'un autre périphérique FireWire, pour également faire fonctionner ces attaques sur les systèmes Windows. Ces travaux en ont inspiré d'autres : [Martin 07] présente plusieurs techniques de compromission de systèmes Linux alors que [Aumaitre 08] s'est focalisé sur les systèmes Windows. Des variantes de l'USB, en particulier l'USB *On-The-Go* (OTG), fournissent des fonctionnalités similaires au FireWire. [Maynor 05] montre ainsi qu'un téléphone portable peut être modifié de façon à injecter du code en mémoire centrale. Il semblerait également que les périphériques USB 3 puissent commander les contrôleurs de périphériques auxquels ils sont connectés. Cependant, à notre connaissance, aucune étude n'a encore confirmé cette assertion.

3.2.1.3 Attaques DMA initiées par un contrôleur d'entrées-sorties

Finalement, certains contrôleurs d'entrées-sorties peuvent initier eux-mêmes des transferts de données vers la mémoire centrale. Nous distinguons, parmi les contrôleurs d'entrées-sorties qui disposent d'une telle capacité, ceux qui ont été conçus délibérément pour cela et ceux qui ont été modifiés à cet effet.

La plupart des contrôleurs d'entrées-sorties conçus pour initier eux-mêmes des transferts de données vers la mémoire centrale reposent sur un micro-contrôleur pour piloter les bus d'entrées-sorties ou sont programmés à l'aide de technologies de logique programmable tels que les FPGA. De nombreuses preuves de concept d'attaques par entrées-sorties ont recours à ce type de contrôleur d'entrées-sorties malveillant [Aumaitre et Devine 10, Carrier et Grand 04, Devine et Vissian 09, Petroni *et al.* 04]. Puisque l'attaquant conçoit lui-même son propre contrôleur d'entrées-sorties, il a la liberté de définir son propre canal de commande et d'implémenter, au sein de son contrôleur d'entrées-sorties, des fonctionnalités qui ne seraient pas disponibles dans les contrôleurs d'entrées-sorties sur étagère.

Une autre possibilité, cependant moins permissive que l'approche précédente, consiste à modifier un contrôleur d'entrées-sorties existant. Ceci est possible parce que les contrôleurs d'entrées-sorties disposent de plus en plus de capacités de calcul pour décharger davantage les processeurs principaux de certains calculs et pour gagner en performances. À cet effet, ils embarquent un ou plusieurs processeurs sur lesquels s'exécute un *firmware* qui décrit les traitements à effectuer. Il est possible de modifier ce programme de façon à ce que le contrôleur d'entrées-sorties puisse, de lui même, initier des accès à la mémoire centrale. À notre connaissance, A. Triulzi a été le premier à utiliser cette technique : il décrit dans [Triulzi 08a] une modification (hors-ligne) du *firmware* de la carte réseau Broadcom Tigon consistant à trans-

[22] À l'origine, les bus FireWire ont été créés pour accueillir des périphériques externes qui nécessitent une bande passante élevée. Parmi ceux-ci, les périphériques multimédias tels que les appareils photo, les caméras numériques et les disques durs externes nécessitent des taux de transferts particulièrement élevés. Ces besoins expliquent la possibilité offerte à ces périphériques de commander le contrôleur de périphériques auquel ils sont reliés et d'initier des transferts avec sa mémoire. Les accès à cette mémoire sont généralement redirigés vers la mémoire centrale.

férer automatiquement les trames réseau reçues qui respectent un certain motif vers la mémoire centrale. Il a suffit qu'il envoie des trames réseau forgées d'une manière spécifique pour initier l'attaque. La modification d'un *firmware* peut également se faire en cours d'exécution, par exemple, par l'exploitation d'une vulnérabilité au sein de celui-ci. Récemment, l'équipe de L. Duflot [Duflot *et al.* 10] a démontré la faisabilité d'une telle attaque sur plusieurs cartes réseau Broadcom NetXtreme. L'un des *firmware* exécuté par ce modèle de cartes réseau, chargé d'implémenter la fonctionnalité d'administration ASF (*Alert Standard Format*), contenait une vulnérabilité de type débordement de tampon. Ils ont ainsi pu prendre le contrôle d'un des processeurs embarqués dans la carte réseau et ils lui ont fait exécuter du code arbitraire en lui envoyant de trames réseau malformées. Cette technique peut aussi être utilisée pour insérer une porte dérobée [Delugré 10].

Les attaques DMA précitées exploitent le manque de contrôle d'accès dans les architectures PC. Nous rappelons qu'il est extrêmement difficile de détecter de manière logicielle ces attaques, dans la mesure où leur mise en œuvre ne nécessite pas l'intervention des processeurs principaux. Comme nous allons le montrer dans les sous-sections 3.2.2 et 3.2.3, il est possible de s'en protéger au moyen de contre-mesures matérielles. Néanmoins, les contre-mesures matérielles elles-mêmes présentent des insuffisances, et d'autres attaques par entrées-sorties restent possibles. Ainsi, nous présentons en sous-section 3.2.4, un exploit consistant à injecter des trames réseau par des accès DMA initiés depuis un périphérique FireWire, et ceci en présence de mécanismes matériels visant à contrôler ces accès. Il nous est alors possible de mener tout type d'attaque réseau (par exemple, de l'*ARP spoofing*) sans qu'elles puissent être détectées puisqu'elles ne laissent aucune trace sur les réseaux.

3.2.2 Quelques contre-mesures spécifiques

Plusieurs solutions techniques ont été proposées pour parer les attaques par accès direct à la mémoire. Dans cette sous-section, nous analysons quelques-unes de ces solutions.

Une première solution, proposée par L. Duflot [Duflot 07], vise spécifiquement le problème des attaques par accès direct à la mémoire initiées par les processeurs. Elle consiste à filtrer, au niveau des processeurs, les accès d'entrées-sorties qui peuvent être considérés comme dangereux. Si l'accès est jugé peu dangereux (par exemple, une opération de lecture sur un registre dont la valeur n'influe pas sur la sécurité du système), le noyau de système d'exploitation laisse l'opération se dérouler. En revanche, si l'accès peut être utilisé pour mettre en place une attaque (et, en particulier, pour configurer des accès directs à la mémoire), le noyau de système d'exploitation bloque l'accès. La mise en place d'une telle solution de filtrage des entrées-sorties nécessite d'apporter des modifications conséquentes au noyau de système d'exploitation (notamment, les appels système) de façon à intercepter et vérifier tous les accès d'entrées-sorties. Il nécessite également d'intégrer au sein du noyau une infrastructure permettant d'appliquer des règles de filtrage qui peuvent évoluer en fonction des événements matériels (par exemple, ajout ou retrait d'un composant matériel). Cette solution étant inefficace vis-à-vis d'attaques initiées par un périphérique ou par un contrôleur d'entrées-sorties, elle n'apporte qu'une réponse partielle au manque de contrôle d'accès sur les bus d'entrées-sorties.

D'autres solutions ont été proposées pour détecter une attaque exécutée par un *firmware* malveillant sur un contrôleur d'entrées-sorties. Elles s'inspirent des techniques de protection d'applications logicielles qui sont aujourd'hui largement éprouvées et nécessitent, en pratique, d'altérer le code du *firmware*. Dans [Duflot *et al.* 11], les auteurs proposent, par exemple, de vérifier les chemins d'exécution du *firmware*. À cet effet, une copie de la pile est créée à chaque

appel de fonction. À chaque retour de fonction, la valeur de l'adresse de retour stockée dans la pile est comparée à sa copie. Il est ainsi possible de vérifier que les chemins d'exécution du *firmware* n'ont pas été modifiés, par exemple suite à une attaque par débordement de tampon.

Dans [Li *et al.* 11], les auteurs proposent une autre approche dite de *remote firmware attestation* qui permet, à tout instant, de déterminer si un contrôleur a été compromis. Il est alors nécessaire de modifier le *firmware* exécuté par le contrôleur de façon à implémenter le protocole proposé. Dans ce protocole, une entité *verifier* dans le système d'exploitation lance un défi au *firmware* exécuté par le contrôleur d'entrées-sorties. Si la réponse au défi est celle attendue par l'entité *verifier*, le contrôleur est sain. Autrement, il est sans doute compromis et, par précaution, il peut être envisagé de le désactiver. Dans un premier temps, un *nonce* est envoyé par le système d'exploitation au contrôleur d'entrées-sorties. Il est essentiel que celui-ci soit aléatoire pour éviter des attaques par rejeu. Ce *nonce* est utilisé par le contrôleur d'entrées-sorties comme graine pour un générateur de nombres pseudo-aléatoires. Le contrôleur calcule alors la somme de contrôle de sa mémoire, et renvoie le résultat du calcul au *verifier*. Ayant une copie de la mémoire du contrôleur d'entrées-sorties, l'entité *verifier* est en mesure de vérifier la réponse au défi et de déterminer si le contrôleur a été compromis.

Une dernière solution, proposée par J. Rutkowska [Rutkowska 07], concerne de façon générique à toutes les attaques par accès direct à la mémoire. Elle consiste à reconfigurer le *chipset* de façon à ce que le *northbridge* redirige automatiquement tous les accès depuis les contrôleurs d'entrées-sorties aux régions de mémoire que l'on souhaite protéger vers un autre emplacement, par exemple vers un autre bus d'entrées-sorties où un contrôleur va éventuellement pouvoir traiter l'accès. À cet effet, elle fait se chevaucher une portion des adresses de la mémoire centrale que l'on souhaite protéger par les adresses utilisées par les contrôleurs d'entrées-sorties. Cela crée un conflit, qui a un comportement différent en fonction de l'entité (processeur ou contrôleur) qui effectue l'accès. Lorsque les accès vers cette zone sont effectués par un contrôleur d'entrées-sorties, au lieu d'être aiguillés vers la mémoire centrale, ils sont aiguillés vers un autre bus d'entrées-sorties pour être traités par un contrôleur configuré à cet effet. En revanche, ils sont bien redirigés vers la mémoire centrale lorsqu'il s'agit d'un accès depuis les processeurs. Une tentative d'accès à ces régions de mémoire protégées depuis un contrôleur d'entrées-sorties peut provoquer, par exemple, un blocage complet du système lorsque l'accès est volontairement aiguillé par le *chipset* vers un emplacement invalide, typiquement un bus d'entrées-sorties où aucun contrôleur n'est configuré pour traiter cet accès. Cette solution n'est malheureusement efficace que pour des régions de mémoire contiguës et s'adapte difficilement aux régions de mémoire fragmentées.

Les solutions évoquées précédemment s'adressent de façon évidente à des attaques bien particulières et ne corrigent pas un problème d'ordre structurel : les accès vers la mémoire centrale ne sont pas suffisamment contrôlés. Même si les travaux publiés dans [Rutkowska 07] semblent ouvrir la voie vers une solution générique, il serait préférable d'effectuer le contrôle d'accès au niveau de l'architecture matérielle, en particulier par le *chipset*. C'est en tout cas la solution qui est adoptée dans les *chipsets* récents avec l'intégration d'une *Input/Output Memory Management Unit* (I/O MMU) qui est l'objet de la prochaine sous-section.

3.2.3 Contre-mesure générique : une I/O MMU

Les I/O MMUs désignent des unités de gestion de la mémoire centrale dédiées aux contrôleurs d'entrées-sorties. Elles ont été initialement conçues pour virtualiser cette mémoire pour les contrôleurs d'entrées-sorties, leur mise en œuvre permettant, entre autres, de simplifier les accès d'entrées-sorties. Il est possible, par exemple, d'utiliser une I/O MMU pour mettre en

place des mécanismes de *scatter-gather* (mécanisme permettant de transférer des données sur des zones de mémoire non-contiguës), ou de pallier certaines limites d'adressabilité de la mémoire physique [23]. Avec l'avènement de la virtualisation, les I/O MMUs ont été étendues afin d'assurer également des propriétés d'isolation. Celles-ci sont indispensables, par exemple, pour empêcher qu'un attaquant utilise un contrôleur d'entrées-sorties associé à un domaine virtualisé [24] pour accéder frauduleusement (par un accès direct à la mémoire) à un autre domaine. Aujourd'hui, AMD *Virtualization Technology* (Vi) et Intel *Virtualization Technology for directed I/O* (VT-d) constituent les implémentations d'I/O MMU les plus complètes pour les architectures PC [25]. En plus de virtualiser la mémoire physique et de vérifier les accès des contrôleurs à celle-ci, ces technologies ont récemment intégré (depuis l'apparition des *chipsets* SandyBridge) des fonctions de virtualisation d'interruptions de type *Message Signaled Interrupt* (MSI).

3.2.3.1 Intel VT-d : I/O MMU dédiée aux architectures Intel-PC

Dans ce manuscrit, nous allons spécifiquement nous intéresser à la technologie Intel VT-d, qui équipe la plupart des *chipsets* Intel actuels. Au niveau matériel, cette contre-mesure est composée d'unités appelées *DMA-Remapping Hardware Units* (DRHUs), chacune de ces unités étant chargée de virtualiser la mémoire centrale pour un ensemble de contrôleurs. Elles sont toutes localisées au sein du répartiteur (plus précisément, dans le *Memory Controller Hub* qui joue le double rôle de répartiteur et de contrôleur de mémoire), ce qui lui permet de bloquer efficacement tout accès frauduleux à la mémoire centrale.

L'activation de ces unités matérielles est laissée à la discrétion du système d'exploitation. Celui-ci les détecte (et les active) au moyen de structures de contrôle (la table ACPI *DMA-Remapping Reporting*, ou DMAR pour être plus précis) mises en place en mémoire centrale par le BIOS lors du démarrage du système (cf. figure 3.4). Une fois activées, les unités matérielles sont configurées au moyen de tables de configuration similaires aux tables des pages utilisées par les unités de gestion de la mémoire implantées dans les processeurs. La figure 3.5 rappelle la logique de parcours de ces tables de configuration par une unité matérielle. De par la position de l'I/O MMU dans l'architecture matérielle, tout accès à la mémoire centrale depuis un contrôleur d'entrées-sorties est obligatoirement intercepté. L'unité matérielle correspondante commence alors par identifier le contrôleur qui l'a initié à l'aide du champ `Requester ID` contenu dans l'en-tête de la requête. Avec cette méta-donnée, elle détermine ensuite, par une série d'indirections, où sont localisées les tables des pages qui lui sont associées. L'accès est bloqué par l'unité matérielle à partir du moment où le contrôleur ne dispose pas des permissions de lecture ou d'écriture requises. Ce principe de fonctionnement des unités matérielles est similaire à celui des unités de gestion de la mémoire dans les processeurs qui implémentent le mécanisme de pagination. Ainsi, par analogie avec ce mécanisme de pagination, les accès à la mémoire centrale s'effectuent par une adresse DMA virtuelle qui est traduite en adresse physique lors du transit dans l'une de ces unités matérielles.

[23] De telles limites existent typiquement lorsque des contrôleurs d'entrées-sorties 32 bits sont connectés à des systèmes utilisant plus de 4 gigaoctets de mémoire : les régions de mémoire situées au delà des 4 gigaoctets sont inaccessibles pour ces contrôleurs.

[24] Très schématiquement, un domaine correspond à un ensemble de régions de mémoire dédié à une entité. Dans le cas présent, l'entité est un contrôleur d'entrées-sorties.

[25] Il est intéressant de noter que des implémentations d'I/O MMU existent sur d'autres architectures. Citons, à titre d'exemples, la technologie *Calgary* disponible dans les serveurs IBM System X, la *DMA Address Relocation Table* (DART) des processeurs PowerPC ou les *Peripheral Access Management Units* (PAMU) des plate-formes QorIQ Freescale (architecture Power).

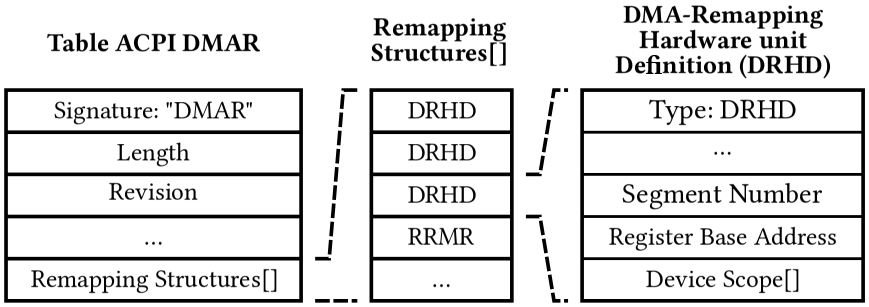

FIGURE 3.4 – Structure simplifiée de la table ACPI DMAR

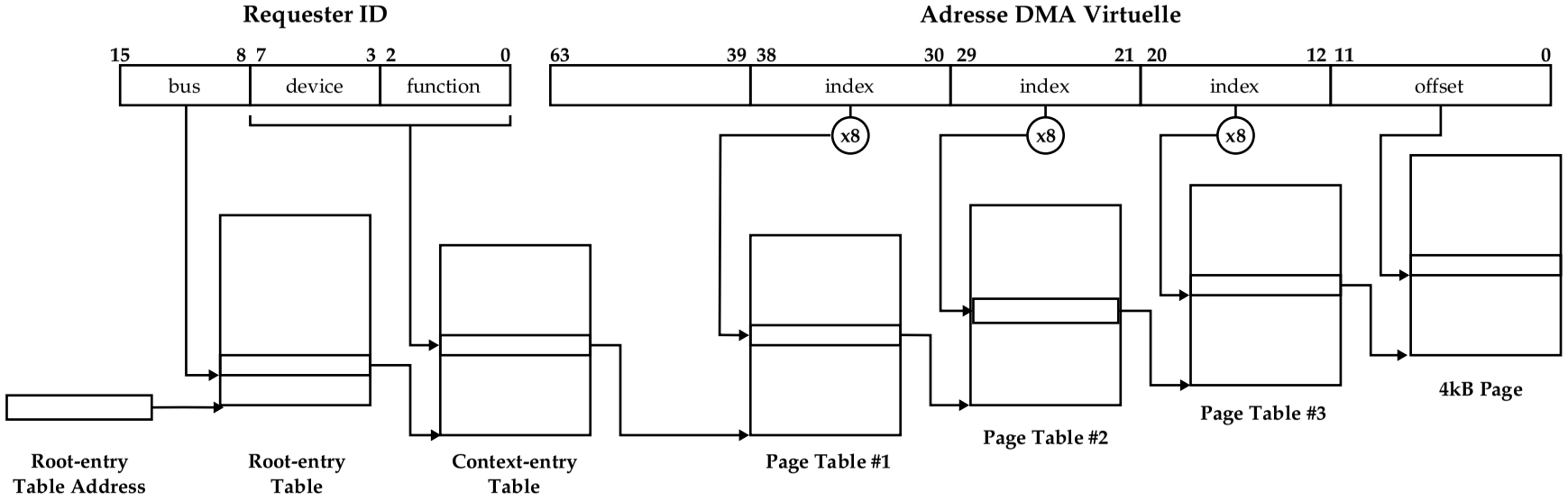

FIGURE 3.5 – Logique de parcours des tables de configuration pour Intel VT-d

3.2.3.2 Limitations d'Intel VT-d

De par sa position dans l'architecture matérielle, l'I/O MMU apporte indéniablement une solution aux attaques par accès direct à la mémoire centrale, en s'assurant notamment qu'il est impossible aux contrôleurs d'effectuer des transferts de données vers une adresse mémoire non autorisée. Cependant, il est important de vérifier que son implémentation est fiable et qu'elle reste incontournable pour que cette solution soit efficace. En effet, il serait problématique qu'un attaquant puisse la reconfigurer, la désactiver ou contourner son contrôle d'accès. Afin de nous en assurer, nous avons étudié en détail son fonctionnement. Cette analyse a notamment révélé plusieurs limitations qu'un attaquant peut utiliser pour contourner le contrôle d'accès mis en œuvre sur les bus d'entrées-sorties. Les résultats de cette analyse ont été publiés en 2010 au Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC) [Lone Sang *et al.* 10a] et à l'International Conference on Malicious and Unwanted Software (MALWARE) [Lone Sang *et al.* 10b].

Contrôle d'accès insuffisant La contre-mesure matérielle Intel VT-d ne permet de contrôler que partiellement les entrées-sorties. En effet, seuls les accès à la mémoire centrale sont filtrés. Il reste possible, par exemple, aux contrôleurs d'entrées-sorties de dialoguer directement entre eux, via l'un des espaces d'adressages. Nous traitons des attaques exploitant les communications pair-à-pair dans la section 3.3.

Contrôle d'accès configurable par du logiciel. Comme la plupart des composants de l'architecture PC (GDT, IDT, etc.), la configuration d'Intel VT-d repose sur des structures de contrôle stockées dans la mémoire centrale. Un jeu de tables de configuration à plusieurs niveaux d'indirection est en effet utilisé pour définir la politique de contrôle d'accès à appliquer à chaque contrôleur d'entrées-sorties. La protection de ces parties critiques contre d'éventuelles actions malveillantes est à la charge du noyau de système d'exploitation. Or, les systèmes d'exploitation 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. 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. [Lacombe *et al.* 09] proposent pour cela 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 des processeurs, du *chipset*, etc.).

Activation à la discrétion du système d'exploitation. Puisque l'activation de l'I/O MMU et, par là-même, la mise en œuvre de son contrôle d'accès sur les bus d'entrées-sorties est à la discrétion du système d'exploitation, un attaquant peut éventuellement forcer le système d'exploitation à ne pas activer cette contre-mesure matérielle, par exemple, suite à l'exploitation d'une vulnérabilité logicielle. Par ailleurs, la détection et la mise en œuvre de celle-ci dépend de tables ACPI chargées en mémoire. Comme le chargement du système d'exploitation vient après l'initialisation des contrôleurs d'entrées-sorties par le BIOS, une autre attaque qui peut être envisagée pour désactiver l'I/O MMU consiste à profiter de ce laps de temps pour effacer ces tables de la mémoire depuis un contrôleur malveillant initialisé. Cette attaque est cependant difficile à mettre en œuvre car il est nécessaire de changer la structure de l'ensemble des

tables ACPI afin qu'elle reste cohérente. Une solution plus simple consisterait à rendre incohérente uniquement la table DMAR correspondant à l'I/O MMU de façon à ce que le système d'exploitation ne puisse pas activer correctement cette contre-mesure matérielle.

Utilisation de méta-données fournies par les contrôleurs. Le fait d'utiliser des méta-données fournies par les contrôleurs d'entrées-sorties pour déterminer la politique de sécurité à appliquer à ce contrôleur peut être éventuellement utilisé pour contourner le contrôle d'accès d'Intel VT-d. Puisqu'aucun contrôle n'est effectué sur le champ `Requester ID`, il est sans doute envisageable, pour un contrôleur d'entrées-sorties, d'acquérir les mêmes droits d'accès à la mémoire qu'un autre contrôleur, par exemple, en usurpant son identité. Il suffit pour cela que le contrôleur malveillant remplace son identifiant BDF par celui d'un autre contrôleur dans le champ source (`Requester ID`) des requêtes qu'il émet (cf. sous-section 3.2.4).

Par ailleurs, Intel VT-d prévoit l'utilisation de *device-IOTLB*s implémentées au niveau des contrôleurs d'entrées-sorties. Les *device-IOTLB*s désignent des mémoires caches mises en œuvre dans les contrôleurs d'entrées-sorties. Ces mémoires caches servent à stocker, au sein des contrôleurs d'entrées-sorties, les traductions d'adresses DMA virtuelles que le DRHU effectue fréquemment pour ceux-ci. Il permet aux contrôleurs d'entrées-sorties d'effectuer eux-mêmes les traductions d'adresse pour leurs accès directs à la mémoire. Pour informer l'I/O MMU que l'adresse virtuelle a déjà été traduite, le contrôleur d'entrées-sorties renseigne un drapeau dédié dans l'en-tête de la requête d'accès direct à la mémoire qu'il effectue. À la réception d'un tel accès, l'unité matérielle correspondante la relaie directement au contrôleur auquel il est destiné (généralement, le contrôleur de mémoire) sans effectuer de vérification supplémentaire. Cette fonctionnalité peut alors être détournée par des contrôleurs d'entrées-sorties malveillants pour contourner le contrôle d'accès mis en place au travers des tables de traduction d'Intel VT-d. Heureusement, cette fonctionnalité peut être désactivée au niveau du *chipset*.

Quelques-unes des limitations que nous venons de présenter ont été exploitées pour des attaques. Dans [Wojtczuk et Rutkowska 11b], les auteurs se sont notamment focalisés sur celles qui peuvent être mises en œuvre par un *rootkit* s'exécutant sur les processeurs. *A contrario*, nous allons nous intéresser, dans la suite, aux attaques qui sont initiées depuis les contrôleurs d'entrées-sorties, en particulier au moyen des méta-données qu'ils fournissent à l'I/O MMU.

3.2.4 Attaques par partage d'identifiants

Le standard PCI (pour *Peripheral Component Interconnect*) 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 (de 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 comme bus système. Afin d'assurer la retro-compatibilité des contrôleurs d'entrées-sorties PCI sur la nouvelle architecture, des ponts PCI Express-PCI 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. Dans la pratique, le fonctionnement des ponts PCI Express-PCI se rapproche plus de celui d'un mandataire (en anglais, *proxy*) que d'un pont. Pour nous en convaincre, analysons comment s'effectue la traduction d'un accès direct à la mémoire depuis un contrôleur d'entrées-sorties PCI. Ces accès, contrairement à ceux effectués par les contrôleurs PCI Express, ne contiennent pas de champ `Requester ID`. Pour combler ce

manque, le pont PCI Express-PCI traduit les requêtes PCI au format PCI Express en insérant son identifiant BDF comme `Requester ID` avant de les relayer sur le bus PCI Express.

Du fait de ce mécanisme, tous les accès effectués par les contrôleurs PCI appartenant à un même bus sont vus sur les bus PCI Express comme effectués par le pont PCI Express-PCI. Puisque les accès de ces contrôleurs possèdent le même `Requester ID`, les DRHU les associent au même domaine de protection et donc aux mêmes régions de la mémoire centrale. On comprend dès lors que des problèmes de sécurité liés au partage de ressources (confidentialité, intégrité, disponibilité) peuvent s'en suivre : un contrôleur malveillant peut tirer partie de cette situation pour accéder et altérer les régions de la mémoire centrale utilisées par un autre contrôleur d'entrées-sorties. Nous parlons alors d'attaques par partage d'identifiants.

L'exploitation d'un tel vecteur d'attaque n'est cependant pas triviale et nécessite la résolution d'un certain nombre de contraintes, la première étant que le système sur lequel se déroule l'attaque doit posséder au moins un bus PCI. Il se trouve que beaucoup d'ordinateurs de bureau disposent encore aujourd'hui d'au moins deux connecteurs d'extension PCI [26]. Ces deux connecteurs sont généralement connectés au même bus. Dans la mesure où des contrôleurs sont branchés à ces connecteurs, l'attaque que nous venons de décrire peut être mise en œuvre sur cette population d'ordinateurs.

L'attaquant doit également localiser précisément les régions de mémoire utilisées par les contrôleurs d'entrées-sorties qui partagent la même identité sur les bus PCI Express. Ce type d'attaque peut s'envisager aisément dans une entreprise. En effet, 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.). Un employé malveillant pourrait alors mettre au point une attaque par partage d'identifiants sur sa propre machine. Les machines étant identiques, il pourra sans trop de peine la rejouer sur d'autres machines. Le scénario d'attaque suivant reste plausible : pour attaquer ses collègues (et en particulier son supérieur hiérarchique), un employé malveillant peut par exemple programmer l'attaque dans un périphérique malveillant (un iPod par exemple) puis prétendre que celui-ci contient un document présentant un intérêt pour la personne ciblée (par exemple, un rapport de réunion, une musique, etc.) afin que celle-ci le connecte, déclenchant ainsi l'attaque.

Afin d'illustrer cette vulnérabilité, nous avons développé une preuve de concept qui reprend le scénario d'attaque ci-dessus et une vidéo de démonstration de cette attaque est disponible [Lone Sang *et al.* 10]. Dans celle-ci, nous profitons du fait qu'une carte réseau et une carte FireWire sont connectées au même bus PCI. Nous utilisons alors la carte FireWire (pilotée par un iPod modifié pour initier des accès DMA) afin d'injecter des trames réseau au sein des régions de la mémoire centrale utilisées par la carte réseau pour délivrer ces trames au système d'exploitation. L'injection de paquets ARP malveillants nous a alors permis de détourner une communication, en l'occurrence un flux vidéo. Le lecteur curieux pourra trouver davantage de détails sur l'attaque que nous avons développée en annexe A.

3.2.5 Contre-mesures complémentaires

Pour se protéger contre le partage d'identité sur le bus PCI, une solution simple consiste à isoler chaque contrôleur d'entrées-sorties PCI sur un bus séparé. Ce faisant, la confusion entre les identifiants des contrôleurs PCI disparaîtrait. La fonctionnalité que nous avons utilisée pour notre étude ne serait plus exploitable car l'isolation entre les régions de mémoires associées

[26] Constat effectué sur les cartes-mère Intel dont les spécifications techniques sont disponibles à l'adresse http://ark.intel.com/ProductCollection.aspx?familyId=1125

aux différents contrôleurs d'entrées-sorties serait faite de manière stricte. Nous préconisons cette solution à court terme. Une solution à long terme consisterait à 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 Express-PCI et, par conséquent, à la suppression de la fonctionnalité que nous avons exploitée.

Le problème lié au partage d'identifiants sur les bus PCI soulève un autre problème plus large et plus préoccupant : les attaques qui consistent à usurper de manière délibérée l'identité d'un autre contrôleur. La technologie *Access Control Services* a été définie pour répondre à ce problème. Il s'agit d'une technologie de contrôle d'accès disponible uniquement sur certains contrôleurs de bus PCI Express (son implémentation étant optionnelle). Il est possible de mettre en œuvre les ACS pour vérifier systématiquement si l'identité du contrôleur qui effectue l'accès est correcte et bloquer son accès dès lors qu'une usurpation d'identité est détectée. Il est important de préciser que ce contrôle d'accès ne se focalise pas que sur les requêtes d'accès à la mémoire mais s'applique indifféremment à tout type de transaction PCI Express (c'est-à-dire, à tous les accès d'entrées-sorties). 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 situés dans le *northbridge*. Nous espérons que les prochaines générations de *chipsets* proposeront leur mise en œuvre au sein du *southbridge* afin de mieux contrôler les bus d'entrées-sorties. Le support de la technologie ACS par un contrôleur se manifeste par la présence d'un groupe de registres projeté dans son espace de configuration, nommé *ACS Extended Capability*. Le système d'exploitation repose sur les registres `ACS_CAPABILITY` et `ACS_CONTROL` respectivement pour lister l'ensemble des services implémentés par un contrôleur et pour les activer. Le tableau 3.1 présente l'ensemble des services prévus par le standard PCI Express.

Type de contrôle d'accès	**Description**
ACS Source Validation	Bloque toutes les requêtes ayant un numéro de bus n'appartenant pas à une plage attendue
ACS Translation Blocking	Bloque toutes les requêtes ayant le champ AT positionné
ACS P2P Request Redirect	Redirige toutes les requêtes directes entre contrôleurs (dites pair-à-pair) vers d'autres contrôleurs
ACS P2P Completion Redirect	Redirige toutes les réponses directes entre contrôleurs (dites pair-à-pair) vers d'autres contrôleurs
ACS P2P Upstream Forwarding	Redirige toutes les requêtes directes entre contrôleurs (dites pair-à-pair) vers d'autres contrôleurs
ACS P2P Egress Control	Bloque toutes les requêtes pair-à-pair
ACS Direct Translated P2P	Lorsque le champ AT est positionné, transmet la requête vers le contrôleur de destination sans vérification supplémentaire

TABLE 3.1 – *Access Control Services*

En faisant l'hypothèse que l'ensemble des contre-mesures matérielles que nous venons de décrire soient activées et mises en œuvre par des systèmes d'exploitation fiables, il sera sans doute difficile qu'un attaquant réussisse encore à mettre en place des attaques par accès direct à la mémoire centrale sans qu'elles ne soient détectées et contrées. En effet, étant située au plus près de la mémoire centrale, l'I/O MMU doit empêcher en particulier que des accès frauduleux à cette mémoire ne soient possibles. Les extensions ACS la complètent en veillant à ce qu'au-

cun contrôleur d'entrées-sorties ne puisse usurper l'identifiant d'un autre contrôleur dans le but d'obtenir les mêmes privilèges que celui-ci à la mémoire centrale. Cependant, ces contre-mesures ne sont pas directement efficaces contre les attaques par accès pair-à-pair présentées dans la prochaine section.

3.3 Contrôle des accès pair-à-pair

Dans la section précédente, nous n'avons considéré que les accès depuis les contrôleurs d'entrées-sorties destinés à la mémoire centrale. Il est opportun de remarquer que certains contrôleurs d'entrées-sorties, notamment certaines cartes graphiques et certaines cartes réseau, ont la faculté de communiquer entre eux, sans que les processeurs ne se chargent des transferts ni que la mémoire centrale ne soit utilisée comme relais. Dans les architectures PC plus anciennes, ces communications directes entre contrôleurs d'entrées-sorties s'effectuaient au travers de bus dédiés (et propriétaires) : il était alors nécessaire de coupler plusieurs contrôleurs au moyen de connecteurs spécifiques (par exemple, par un câble SLI ou CrossFire pour le cas de cartes graphiques). Les vitesses des bus actuels ont permis de s'affranchir de ces connecteurs spécifiques et les échanges se font aujourd'hui directement au travers des bus d'entrées-sorties. Nous parlons alors d'accès pair-à-pair.

La présente section s'organise comme suit. Nous commençons par rappeler les principes des accès pair-à-pair. Nous précisons en particulier les mécanismes internes aux *chipsets* qui permettent de mettre en œuvre ce type d'accès. Nous voyons sans trop de peine les bénéfices que pourrait alors tirer un attaquant d'un tel mécanisme. Les attaques par accès pair-à-pair sont d'autant plus dangereuses qu'elles ne modifient à aucun moment la mémoire centrale et, par là-même, constituent des menaces encore plus difficiles à détecter par les solutions logicielles habituelles. Après avoir détaillé le principe de ces attaques, nous présentons les résultats de l'étude que nous avons menée sur différents *chipsets* récents afin d'identifier les accès pair-à-pair possibles et les attaques qui peuvent en découler. Finalement, nous discutons des contre-mesures à celles-ci. Les résultats présentés dans cette section ont été publiés lors du Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC) [Lone Sang *et al.* 11] en 2011.

3.3.1 Attaques par accès pair-à-pair

Le *chipset* peut être considéré comme un réseau commuté organisé selon une topologie de bus hiérarchique. Chaque pont se charge d'aiguiller chaque accès qu'il reçoit sur l'un de ses bus d'entrées-sorties père ou fils (cf. figure 3.1). L'accès se propage ainsi, de proche en proche, au travers des bus jusqu'au contrôleur de destination où il sera traité. Afin que la configuration de l'ensemble du *chipset* soit cohérente, il incombe au BIOS de configurer individuellement, lors du démarrage du système, chaque pont au travers de l'espace de configuration PCI ou PCI Express qui leur est associé. Notamment, la paire de champs `I/O Base` et `I/O Limit` (cf. figure 3.3) indique la plage de ports d'entrées-sorties qui est décodée pour ses bus fils. De la même façon, la paire `Memory Base` et `Memory Limit` définit la plage d'adresses pour l'espace mémoire décodée pour ses bus fils. Ainsi, lorsqu'un accès est effectué depuis son bus père vers l'une de ces deux plages d'adresses, l'accès est aiguillé par le pont vers l'un de ses bus fils. Dans le cas inverse, l'accès est aiguillé de l'un des bus fils vers le bus père.

Comme indiqué précédemment, les accès d'entrées-sorties peuvent cibler différents espaces d'adressage (cf. sous-section 3.1.2). Ainsi, les accès pair-à-pair peuvent aussi bien (théorique-

ment) porter sur l'espace mémoire, sur l'espace d'entrée-sortie ou sur l'espace de configuration de contrôleur PCI et PCI Express. Les attaques qui abusent des accès pair-à-pair cherchent à exploiter directement les ressources fournies par le matériel, tout en contournant les éventuels mécanismes de protection mis en place par le système d'exploitation. En pratique, seuls les accès pair-à-pair au travers de l'espace mémoire sont actuellement exploités par des attaquants, en raison de la lenteur des transferts qu'impliquent les autres espaces d'adressage. Du point de vue des bus d'entrées-sorties, ceux-ci sont identiques en tout point aux accès directs à la mémoire centrale, et seules les adresses utilisées pour chaque type d'accès diffèrent. C'est pour cette raison qu'aujourd'hui la plupart des attaques par accès pair-à-pair se focalisent sur l'espace mémoire et sont à l'initiative soit des processeurs, soit des contrôleurs d'entrées-sorties, soit des périphériques.

M. Dornseif a probablement été l'un des premiers à démontrer la faisabilité de telles attaques. Pour illustrer ses travaux publiés dans [Dornseif 04], il a développé une preuve de concept d'attaque par accès pair-à-pair au travers de l'espace mémoire dans laquelle il modifie le *framebuffer* d'une carte graphique depuis un iPod connecté via le bus FireWire. Des scénarios d'attaque plus complets ont été développés par A. Triulzi. La modification du *firmware* d'une carte réseau lui a permis de transférer les trames respectant un certain motif dans la mémoire interne de la carte graphique, où a été implanté un serveur *shell* sécurisé [Triulzi 08a]. Il a ensuite étendu ses travaux en montrant que cet accès distant peut être utilisé pour charger, depuis la carte graphique, un nouveau *firmware*, téléchargé depuis l'Internet, dans une autre carte réseau [Triulzi 10b]. Ce *firmware* spécifique lui a permis d'altérer le comportement de la carte réseau de façon à ce que certaines trames soient directement transférées vers une autre carte réseau, contournant ainsi tout filtrage de paquets mis en œuvre au sein du système d'exploitation.

Les travaux de M. Dornseif et d'A. Triulzi se sont *a priori* focalisés sur l'ancienne architecture PC, dans laquelle le *chipset* est basé sur un bus système PCI. Étant physiquement un bus partagé, nous concevons que les accès pair-à-pair puissent se faire sur un bus PCI. Toutefois, les bus d'entrées-sorties dans les architectures PC actuelles ne reposent plus sur le protocole PCI mais implémentent le protocole PCI Express (cf. figure 3.1), qui est beaucoup plus performant. Chaque contrôleur d'entrées-sorties est alors relié par un lien dédié à un pont PCI Express qui se charge d'émuler une topologie de bus partagé. Dans cette configuration de bus, il n'est pas clair que les accès pair-à-pair soient possibles de la même façon que pour les bus PCI. Il peut être envisagé, par exemple, d'appliquer une politique de filtrage au niveau de ce pont. Dans le but d'avoir une meilleure connaissance de l'architecture PC actuelle et par là-même une meilleure maîtrise de la sécurité du système, nous avons mené une étude pour identifier quels sont les accès pair-à-pair possibles dans les *chipsets* récents. Cela est l'objet de la prochaine sous-section.

3.3.2 Mise en œuvre sur plusieurs *chipsets* récents

Afin d'identifier les *chipsets* actuels qui permettent de mettre en place des transferts pair-à-pair, nous avons procédé expérimentalement : nous avons lancé ce type de transfert de données entre différents contrôleurs d'entrées-sorties et nous avons consigné les résultats dans un tableau. La table 4.2 énumère les *chipsets* que nous avons considérés dans notre étude. Pour chacun de ces *chipsets*, 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 4.3 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 le premier test, nous avons laissé intacte la configuration du *chipset*. Pour le second, marqué par le signe † dans le tableau, nous avons modifié volontairement la configuration du *southbridge* [27]. Il est sans doute possible aussi pour un attaquant d'altérer la configuration du pont PCI Express-PCI depuis un contrôleur d'entrées-sorties si celui-ci est capable de générer des requêtes de configuration sur les bus d'entrées-sorties.

L'analyse de la table 4.3 nous permet d'identifier les transactions pair-à-pair, et donc les éventuelles attaques par accès pair-à-pair, 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. 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 pair-à-pair 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 réseau par exemple, directement dans la mémoire de la carte graphique afin que cette dernière puisse les traiter, réduisant ainsi la durée des transferts d'entrées-sorties. Soulignons, cependant, que les composants logiciels actuels (par exemple, les pilotes, les bibliothèques graphiques, etc.) 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'entrées-sorties, il n'a aucun retour direct sur la réussite ou l'échec de son attaque.

Nos expérimentations sur différents modèles de *southbridge* montrent que les résultats sont cohérents d'un *chipset* à l'autre. Il est possible d'établir un canal de communication depuis un contrôleur d'entrées-sorties quelconque vers un contrôleur du *northbridge*. Les requêtes autorisées entre ces contrôleurs sont ensuite définies par le *northbridge*. Les contrôleurs PCI peuvent également communiquer directement entre eux puisqu'ils sont connectés au même bus d'entrées-sorties. 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 pair-à-pair. 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 par accès pair-à-pair.

La possibilité d'effectuer des accès pair-à-pair 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.

[27] À titre informatif, nous donnons la commande que nous avons utilisée pour cela : `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 pair-à-pair 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.

	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 3.2 – 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 3.3 – Résultats expérimentaux

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.

Naturellement, nous avons implémenté plusieurs preuves de concept afin de montrer ce qui est réalisable en pratique avec cette classe d'attaque. Dans une première preuve de concept, nous avons exploité les accès pair-à-pair depuis un contrôleur malveillant (par exemple, une carte FireWire, une carte réseau) pour accéder à la mémoire vidéo d'une carte graphique. Dans le principe, cette attaque est similaire à celle proposée par M. Dornseif [Dornseif 04]. Toutefois, le fait que nous nous sommes restreints uniquement à des opérations de lecture et que nous avons déroulé notre attaque sur des *chipsets* intégrant davantage de moyens de protection distingue notre preuve de concept de celle qu'il a présentée. Une vidéo présentant la mise en œuvre de l'attaque est disponible [Lone Sang *et al.* 11b] : nous attaquons une carte graphique depuis un contrôleur FireWire, piloté depuis un périphérique malveillant (ici, un ordinateur portable). Par ce moyen, nous recopions le contenu de l'écran de la machine cible et nous réussissons à récupérer les identifiants ainsi que les mots de passe des utilisateurs affichés à l'écran. Les détails de notre preuve de concept sont fournis en annexe B.

Une autre preuve de concept que nous avons mise au point exploite la possibilité d'effectuer des accès pair-à-pair vers l'espace d'entrée-sortie. Bien que celui-ci soit aujourd'hui peu usité, nous observons que des contrôleurs (le contrôleur de clavier, les contrôleurs VGA, SATA, USB, etc.) continuent à y projeter certains de leurs registres. Le contrôleur de clavier projette, par exemple, des registres aux adresses `0x60` et `0x64` qui informent le système d'exploitation des événements du clavier ou de la souris. Nous avons exploité la possibilité de lire ces ports d'entrées-sorties depuis un contrôleur d'entrées-sorties pour implémenter un enregistreur de frappe au clavier. Actuellement, nous sommes capables d'enregistrer les frappes de claviers de type PS/2 et éventuellement des claviers de type USB lorsque l'option *USB Legacy Support* [28] est activée dans le BIOS. Le lecteur intéressé pourra consulter l'annexe C pour plus de détails sur la mise en œuvre de l'attaque et une vidéo présentant la mise en œuvre de notre preuve de concept est disponible [Lone Sang *et al.* 12a].

3.3.3 Contre-mesures envisageables

Afin de maîtriser les transferts de données directs entre contrôleurs d'entrées-sorties, nous pouvons nous appuyer sur différents mécanismes matériels existants.

Il est possible d'utiliser des contre-mesures matérielles implémentant le principe d'I/O MMU pour limiter les capacités d'interaction de pair-à-pair entre contrôleurs d'entrées-sorties, en particulier, ceux effectuant des transferts vers des adresses de l'espace mémoire. 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'entrées-sorties et la mémoire centrale. L'agencement de ces composants dans l'architecture matérielle fait en sorte qu'ils voient transiter toute requête à destination de la mémoire centrale. De ce fait, ils la protègent efficacement de tout accès arbitraire depuis un contrôleur d'entrées-sorties malveillant. En ce qui concerne les accès pair-à-pair, tout échange d'un contrôleur d'entrées-sorties situé dans le *northbridge* vers un contrôleur quelconque, qu'il soit dans le *northbridge* ou dans le *sou-*

[28] Nous rappelons que cette option permet d'utiliser certains périphériques USB (un clavier, une souris, un périphérique de stockage) au démarrage de la machine, bien avant que les contrôleurs USB ne soient initialisés par le système d'exploitation. Dans cette configuration, le BIOS (ou plus précisément la routine de traitement de la SMI) présente les claviers ou les souris USB au système d'exploitation comme étant des périphériques PS/2.

thbridge, transite obligatoirement par l'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 l'I/O MMU. Pour cette raison, les requêtes ne peuvent pas être contrôlées. Il est assez simple de vérifier cela sur les modèles de *southbridge* actuels en mettant en place un transfert pair-à-pair d'un contrôleur PCI vers d'autres contrôleurs du *southbridge*. Afin de contrôler entièrement les communications au sein du chipset, il serait envisageable d'avoir plusieurs I/O MMU (par exemple, une dans le *northbridge* et une dans le *southbridge*) placées au plus près des contrôleurs d'entrées-sorties et non au plus près de la mémoire centrale. Cependant, cette solution rend encore plus complexe l'architecture matérielle résultante car il est nécessaire que les I/O MMUs se fassent mutuellement confiance : si une première I/O MMU a déjà 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 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 celles-ci soient vérifiées par l'I/O MMU. Bien que cette solution puisse dégrader les performances, 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. 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'entrées-sorties. 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'entrées-sorties connectés au *southbridge* en activant l'*ACS P2P Egress Control*.

3.4 Conclusion

La motivation principale qui nous a conduits à l'élaboration d'un modèle d'attaques est d'étudier les différentes manières pour un attaquant de mettre en défaut la sécurité d'un système informatique. Dans ce chapitre, nous avons instancié notre modèle d'attaques pour une architecture de système informatique particulière, l'architecture PC, sur laquelle s'appuie la majorité des systèmes informatiques d'aujourd'hui. Afin de mieux cerner les attaques par entrées-sorties qui nous intéressent particulièrement, nous avons commencé par rappeler quelques considérations techniques sur cette architecture matérielle. À partir de là, nous avons étudié un sous-ensemble de ces attaques : les attaques impliquant les contrôleurs d'entrées-sorties et qui exploitent leur capacité à gérer eux-mêmes leurs transferts de données. Nous avons alors fait apparaître la faiblesse structurelle à l'origine de ces attaques dans les architectures PC, à savoir le manque de contrôle d'accès sur les entrées-sorties. Dans le but de mieux nous rendre compte de celle-ci, nous nous sommes mis dans la peau d'un attaquant. Nous avons décrit la mise en œuvre de quelques attaques et nous avons présenté plusieurs contre-mesures à chacune d'elles.

Jusqu'à présent, nous avons suivi une approche traditionnelle d'analyse d'un système informatique consistant à rechercher et identifier une vulnérabilité, développer une preuve de concept d'exploitation de la vulnérabilité pour finalement proposer des contre-mesures à celle-

ci. Cette approche s'est avérée fructueuse. En effet, elle nous a permis d'identifier plusieurs vulnérabilités qui sont susceptibles d'être exploitées sur les systèmes PC récents pour effectuer des attaques par entrées-sorties. Il convient, cependant, de nuancer ce bilan, compte tenu des difficultés liées à la mise en œuvre d'une approche d'analyse de vulnérabilités traditionnelle, quel que soit le système cible. Il s'agit, tout d'abord, d'une approche extrêmement chronophage. La conséquence immédiate à cela est que toute analyse ne peut généralement couvrir qu'une infirme partie du système cible. Par ailleurs, les résultats obtenus par cette approche varient généralement en fonction de l'expérience et de l'intuition de l'analyste. Afin de couvrir plus rapidement davantage de bogues potentiels (et éventuellement, davantage de vulnérabilités potentielles) dans le matériel, il paraît essentiel d'adopter une démarche plus systématique d'analyse de vulnérabilités. Cela fait l'objet du chapitre suivant. À cet effet, nous avons développé un contrôleur spécifique capable de générer tout type de trame sur les bus d'entrées-sorties. Entre autres, ce contrôleur va nous permettre d'obtenir les mêmes pouvoirs qu'un attaquant installant une porte dérobée dans le contrôleur qu'il conçoit ou dans lequel il a découvert une vulnérabilité le lui permettant.

Chapitre 4

Application des techniques de *fuzzing* sur les bus d'entrées-sorties

Dans le chapitre précédent, nous avons suivi la démarche classique en analyse de vulnérabilités consistant à découvrir une vulnérabilité, implémenter une preuve de concept, puis proposer des contre-mesures. Ce chapitre décrit une autre approche, complémentaire à l'approche empirique adoptée jusqu'ici, qui devrait permettre de découvrir les vulnérabilités de façon systématique. Elle s'inspire des techniques de recherche de vulnérabilités dans les composants logiciels et consiste à injecter des fautes directement sur les bus d'entrées-sorties. Nous couvrons ainsi un spectre plus large de vulnérabilités : les vulnérabilités dans les composants matériels et aussi, dans une certaine mesure, les vulnérabilités logicielles.

La mise en œuvre de cette approche nécessite, en premier lieu, un outil d'injection de fautes qui agisse au niveau des bus d'entrées-sorties. Un tel outil peut être obtenu de différentes manières. La première consiste à altérer le système logiciel exécuté par un contrôleur d'entrées-sorties sur étagère, par exemple en modifiant son *firmware* ou en exploitant une vulnérabilité au sein de celui-ci. Pour cette approche, il est nécessaire de s'assurer au préalable que le contrôleur d'entrées-sorties permette au *firmware* exécuté de générer tout type d'accès d'entrées-sorties, en particulier des accès d'entrées-sorties invalides vis-à-vis des spécifications des bus d'entrées-sorties. La plupart des contrôleurs d'entrées-sorties qui offrent une telle possibilité ne permettent généralement d'effectuer que les accès d'entrées-sorties les plus usités et n'autorisent, dans le meilleur des cas, que la modification de certains paramètres de ces accès. Cette approche n'est alors pas adaptée à une étude exhaustive des attaques par entrées-sorties. À l'instar de l'*Agilent U4305A PCIe Exerciser* [Agilent Technologies 12] et du *LeCroy Summit z3-16 Exerciser* [LeCroy Corporation 12], il existe des équipements commerciaux dédiés aux tests et à la vérification d'implémentations de standard de bus d'entrées-sorties. Il est probable que ces équipements permettent également de faire de l'injection de fautes sur les bus d'entrées-sorties. Malheureusement, ils s'adressent généralement à un marché de niche et restent très onéreux [29]. Ainsi, seuls les fabricants de *chipsets*, les fabricants de cartes-mère et un nombre restreint de fabricants de contrôleurs d'entrées-sorties peuvent se permettre d'investir dans un tel équipement. Finalement, l'approche la moins onéreuse consiste à concevoir soi-même un tel outil. Aujourd'hui, le développement d'un tel outil est facilité, entre autres, par les cartes de développement qui embarquent des technologies de logique programmable tels que les FPGA. Cette dernière approche s'est naturellement imposée pour nos travaux.

[29] Une carte *Agilent U4305A PCIe Exerciser* coûte à elle seule 13 065 euros. Cette carte nécessite l'acquisition d'une suite logicielle dont la licence s'élève à 27 318 euros.

Une fois l'outil d'injection de fautes mis au point, une seconde étape consiste à simuler, à l'aide de celui-ci, un ensemble d'attaques par entrées-sorties. La mise en œuvre de ces attaques peut être effectuée manuellement : un opérateur humain définit alors les entrées de test et les exécute une par une sur le système sous test. Cette façon de procéder est fastidieuse et sujette aux erreurs. Aussi, une mise en œuvre au moins partiellement automatisée est préférable.

Ces deux aspects font l'objet de ce chapitre qui s'organise comme suit. Nous commençons par présenter l'outil d'injection de fautes sur les bus d'entrées-sorties que nous avons développé au cours de nos travaux. Nous avons nommé notre prototype IronHide. Celui-ci est capable de générer tout type d'accès – valides mais surtout invalides – sur les bus d'entrées-sorties. Il se présente comme un contrôleur d'entrées-sorties classique. Nous explorons ensuite l'idée d'appliquer des techniques de *fuzzing* pour automatiser la mise en œuvre des différentes attaques par entrées-sorties. Cette technique de test va nous permettre, d'une part, de vérifier l'implémentation des bus d'entrées-sorties des systèmes Intel-PC et, d'autre part, d'identifier des vulnérabilités et, éventuellement, des fonctionnalités cachées. Nous rappelons ensuite les grands principes du *fuzzing* avant de décrire l'environnement de test que nous avons mis en œuvre et les résultats obtenus lors de nos expérimentations.

4.1 IronHide : outil d'injection de fautes sur les bus d'entrées-sorties

Le bus PCI Express est aujourd'hui la norme de connexion des contrôleurs d'entrées-sorties dans les architectures matérielles PC. Dans cette section, nous commençons par introduire quelques éléments sur ce standard de bus. Ces éléments permettent de mieux appréhender l'outil d'injection de fautes que nous présentons par la suite. Cet outil a fait l'objet d'un article présenté lors de la conférence SSTIC [Lone Sang *et al.* 12b].

4.1.1 Introduction aux bus PCI Express

Cette sous-section commence par présenter l'architecture d'un contrôleur d'entrées-sorties typique. Nous nous focalisons ensuite sur la couche protocolaire par laquelle se font les injections de fautes. Finalement, nous présentons le format des paquets PCI Express échangés.

4.1.1.1 Protocole PCI Express

L'architecture des contrôleurs d'entrées-sorties PCI Express s'inspire fortement du modèle *Open System Interconnection* (OSI). En effet, afin de réduire la complexité de leur logique, ils sont généralement architecturés en couches protocolaires indépendantes (cf. figure 4.1). Chaque couche utilise les services fournis par les couches de niveau inférieur et définit de nouveaux services pour les couches de niveau supérieur. Nous distinguons trois couches protocolaires principales : la couche transaction, la couche liaison de données et la couche physique.

La couche transaction est la couche la plus haute du modèle en couches. Elle se charge principalement du transport de bout en bout des données entre les contrôleurs d'entrées-sorties. À cet effet, elle encapsule ces données au sein d'un *Transaction Layer Packet* (TLP) et utilise les services fournis par les couches de niveau inférieur pour acheminer le paquet jusqu'au contrôleur destinataire. Un TLP se compose au minimum d'un en-tête dont la longueur dépend du type de transaction. Optionnellement, il peut transporter des données : les requêtes d'accès en écriture contiennent typiquement des données alors que les requêtes d'accès en lecture n'en nécessitent pas. Finalement, il peut également contenir un code détecteur et correcteur d'erreurs représenté sur la figure par le champ ECRC (pour *End-to-end CRC*). La couche liaison de données

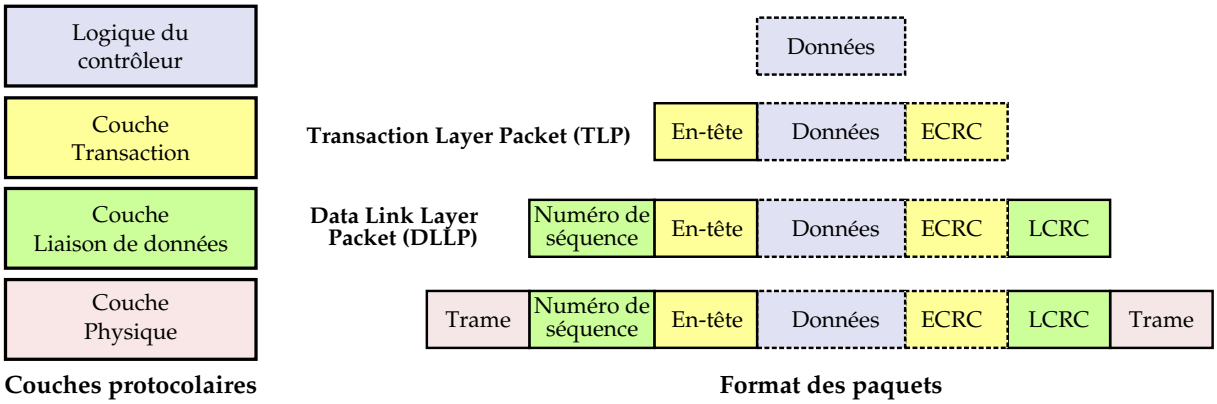

FIGURE 4.1 – Couches protocolaires PCI Express

se charge du transport des TLP entre contrôleurs d'entrées-sorties PCI Express adjacents. Un TLP est ainsi transporté de proche en proche, sur les pistes (en anglais, *lanes*) du bus PCI Express, vers le contrôleur d'entrées-sorties de destination. Pour cela, cette couche protocolaire encapsule à son tour chaque TLP dans un *Data Link Layer Packet* (DLLP), en ajoutant à celui-ci un numéro de séquence (pour la gestion des éventuels déséquencements) et un autre code correcteur d'erreurs appelé LCRC (pour *Link CRC*). Finalement, la couche physique se charge d'émettre ou de recevoir les DLLP sur les pistes PCI Express. Elle se compose de circuits électroniques : des convertisseurs parallèle-série et série-parallèle, des boucles à verrouillage de phase (en anglais, *Phase Locked-Loop*) et des circuits d'adaptation d'impédance, etc.

Il convient de remarquer que le standard de bus PCI Express définit deux types de CRC : un ECRC et un LCRC. Les deux types de CRC sont utilisés par les contrôleurs d'entrées-sorties pour vérifier l'intégrité des paquets reçus. À l'émission, un contrôleur d'entrées-sorties procède à un calcul sur les données contenues dans le paquet sortant et lui adjoint le résultat de ce calcul. À la réception, un contrôleur d'entrées-sorties effectue le même calcul sur le paquet reçu et compare le résultat à la valeur extraite du paquet. Le récepteur détecte une erreur de transmission lorsque les deux valeurs calculées sont différentes. Ces deux types de CRC se distinguent principalement par leurs tailles – 32 bits pour l'ECRC contre 16 bits pour le LCRC – et par la couche PCI Express qui est en charge de leur calcul et de leur vérification.

4.1.1.2 Classes de transactions PCI Express

Pour transférer des données vers les différents espaces d'adressage, les contrôleurs PCI Express reposent sur des transactions. Nous distinguons quatre principales classes de transactions. Les classes de transactions *memory*, *I/O* et *configuration* permettent d'accéder respectivement à l'espace d'adressage principal de la mémoire, à l'espace *Port I/O* et à l'espace de configuration des contrôleurs d'entrées-sorties PCI ou PCI Express. Une dernière classe de transactions, appelée *message*, permet de véhiculer des informations diverses entre les contrôleurs PCI Express. Elle est utilisée notamment pour signaler des interruptions, des erreurs ou pour véhiculer des messages de gestion d'énergie. La table 4.1 présente les principaux types de transactions définies dans le standard PCI Express pour chacune des classes de transactions.

Nous distinguons deux catégories de transactions, *posted* (sans réponse) et *non-posted* (avec réponse), selon qu'elles impliquent ou pas une réponse (appelée *completion*) de la part du contrôleur d'entrées-sorties qui traite l'accès. Le paquet *completion* a pour rôle d'acquitter la

Classes	Type de transaction	Avec ou sans réponse
Memory	*Memory Read*	*avec réponse*
	Memory Write	*sans réponse*
	Memory Read Lock	*avec réponse*
I/O	*IO Read*	*avec réponse*
	IO Write	*avec réponse*
Configuration	*Configuration Read Type 0*	*avec réponse*
	Configuration Read Type 1	*avec réponse*
	Configuration Write Type 0	*avec réponse*
	Configuration Write Type 1	*avec réponse*
Message	*Message*	*sans réponse*

TABLE 4.1 – Transactions PCI Express

bonne réception de la requête d'accès et contient éventuellement les données demandées. Les transactions d'écriture correspondent, typiquement, à des transactions *posted* car elles ne requièrent pas de réponse. Les transactions de lecture, quant à elles, correspondent à des transactions *non-posted* car elles requièrent le renvoi d'un paquet *completion* contenant les données demandées. Sur les bus d'entrées-sorties, ces transactions se matérialisent par des *Transaction Layer Packets* (TLP).

4.1.1.3 Format d'un *Transaction Layer Packet*

Un TLP se compose au minimum d'un en-tête et contient optionellement des données et un ECRC. La taille de cet en-tête est variable : il s'étend sur 12 octets ou sur 16 octets en fonction du type de transaction. Dans la suite, nous détaillons uniquement la partie qui leur est commune. Le lecteur est invité à se référer aux spécifications du standard PCI Express [Budruk *et al.* 03] pour la sémantique des champs spécifiques à chaque type de transaction. La figure 4.2 représente un exemple de TLP. Les champs grisés (marqués d'un `R`) représentent des champs réservés. Le standard PCI Express impose qu'ils soient positionnés à zéro lors de la construction d'un TLP et qu'ils soient ignorés par le contrôleur d'entrées-sorties en réception.

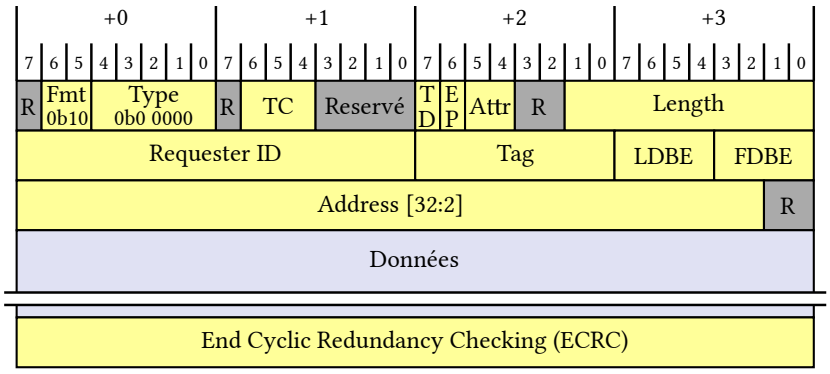

FIGURE 4.2 – Format d'un TLP de type *Memory Write Request* (format 32 bits)

Les champs `Fmt` et `Type` combinés identifient le type de transaction contenue dans le pa-

quet. Il s'agit, dans cet exemple, d'une transaction d'écriture (format 32 bits) [30] à destination de l'espace d'adressage principal de la mémoire. Le champ `TC` indique la classe de trafic à laquelle appartient le paquet. Il permet de différencier les transactions selon huit classes de trafic distinctes qui mettent en œuvre des politiques de service différentes. Le bit `TD` indique si le TLP contient un ECRC. Le bit `EP` indique si le TLP a été « empoisonné », c'est-à-dire que le paquet émis n'est pas valide. Le champ `Attr` contient des informations pour optimiser la gestion de trafic. Le second bit de ce champ indique, par exemple, si une mise en cohérence du cache (en anglais, *snooping*) est nécessaire pour cette transaction. Le champ `Length` indique la longueur (exprimée en blocs de 4 octets) des données transportées dans le TLP.

Il est intéressant de retrouver dans cet exemple plusieurs éléments utilisés par les technologies de type I/O MMU. Le champ `Requester ID` contient, par exemple, l'identité du contrôleur d'entrées-sorties qui initie la transaction. Le champ `Address` indique l'adresse de destination en mémoire centrale où sont écrites les données. Finalement, le champ `AT` indique si celui-ci contient une adresse DMA virtuelle ou une adresse (physique) issue de la traduction d'adresse effectuée par le contrôleur d'entrées-sorties dans le cadre des *devices-I/O TLB*.

4.1.2 Contrôleur d'entrées-sorties IronHide

Dans l'optique d'identifier, puis de classifier les actions malveillantes qu'un attaquant peut effectuer avec un contrôleur, nous avons mis au point notre propre contrôleur d'entrées-sorties en utilisant les technologies de logique programmable. Nous avons appelé notre prototype IronHide. Il se présente comme un contrôleur d'entrées-sorties classique et se connecte directement au bus PCI Express. Dans la suite, nous présentons les caractéristiques qui font sa singularité par rapport à d'autres contrôleurs d'entrées-sorties basés sur des FPGA.

4.1.2.1 Caractéristiques principales

Il existe aujourd'hui un certain nombre de contrôleurs d'entrées-sorties dédiés à l'évaluation de la sécurité des systèmes informatiques qui sont implémentés à partir de technologies de logique programmable tels que les FPGA [Carrier et Grand 04, Petroni *et al.* 04, Devine et Vissian 09, Aumaitre et Devine 10]. Malheureusement, tous ces contrôleurs se présentent comme des cartes d'extension PCI que l'on connecte aux *chipsets* actuels par le biais de connecteurs PCI ou PC Card disponibles sur la carte-mère. Les protocoles de bus PCI et PCI Express étant physiquement incompatibles, ces contrôleurs nécessitent l'utilisation d'un pont PCI Express-PCI intégré au *chipset* pour se connecter aux bus d'entrées-sorties PCI Express. Étant donné que nous ne maîtrisons pas la manière selon laquelle les requêtes PCI sont traduites sur les bus d'entrées-sorties PCI Express au niveau de ce pont, le fait de réutiliser un de ces contrôleurs ne nous a pas paru une solution adaptée pour étudier de façon exhaustive les attaques depuis les contrôleurs d'entrées-sorties. Aussi avons-nous décidé de développer notre propre contrôleur pour qu'il puisse se connecter directement aux bus PCI Express via un connecteur PCI Express ou Express Card.

Avec comme objectif de couvrir au maximum les vecteurs d'attaques possibles depuis les contrôleurs d'entrées-sorties, nous avons conçu notre contrôleur afin qu'il puisse générer des requêtes quelconques sur les bus PCI Express. L'émission de requêtes valides va alors nous permettre de vérifier l'implémentation du standard PCI Express par les autres contrôleurs

[30] Précisons que pour effectuer des accès aux adresses localisées au delà de 4 gigaoctets, les contrôleurs doivent générer des transactions dans le format 64 bits. Le champ `Address` dans le TLP s'étend alors sur 8 octets.

d'entrées-sorties alors que l'émission de requêtes invalides va nous permettre d'évaluer la robustesse de ces mêmes contrôleurs vis-à-vis de fautes injectées au travers des bus d'entrées-sorties.

Notre contrôleur est également polyvalent puisque son comportement est entièrement programmable. Il est envisageable de l'utiliser aussi bien pour faire de l'injection de fautes sur les bus PCI Express que pour émuler une carte d'extension PCI Express quelconque. Il suffit pour cela de modifier le logiciel, écrit en langage C, exécuté par le processeur qu'il embarque.

Enfin, nous avons conçu son architecture de façon à ce qu'elle puisse être étendue par plusieurs contrôleurs de périphériques. Il est, par exemple, possible d'y intégrer un contrôleur série RS-232 pour surveiller, sur une machine servant de moniteur, l'évolution du contrôleur d'entrées-sorties, un contrôleur Ethernet pour interagir avec lui au travers du réseau, un contrôleur USB pour enregistrer sur un périphérique de stockage USB les résultats de nos expérimentations, etc.

4.1.2.2 Architecture interne

La figure 4.3 présente l'architecture interne de notre contrôleur, basée sur la famille de FPGA Xilinx. Puisque ce contrôleur a pour vocation de tester les éventuelles fautes d'interaction entre contrôleurs, il nous est indispensable de pouvoir envoyer des paquets PCI Express quelconques depuis la couche protocolaire transaction. Pour cela, nous avons utilisé l'*IP Core Xilinx LogiCORE IP Endpoint Block Plus for PCI Express* [Xilinx 11b] qui permet, entre autres, de configurer le bloc PCI Express inclus nativement dans le FPGA et fournit une interface pour émettre ou recevoir des TLP PCI Express. Nous avons ensuite interfacé cette unité logique avec un système embarqué composé d'un processeur, de différentes mémoires, d'un contrôleur d'interruptions et, optionnellement, de plusieurs contrôleurs de périphériques (par exemple, des contrôleurs Ethernet, des contrôleurs série RS-232). Ce système embarqué constitue la logique du contrôleur.

Il existe deux manières pour interconnecter l'*IP Core* PCI Express avec notre système embarqué. La première méthode consiste à utiliser une autre unité logique commercialisée par Xilinx, appelée *Xilinx LogiCORE IP PLBv46 RC/EP Bridge for PCI Express* [Xilinx 10a], unité logique payante implémentant entièrement le protocole PCI Express. Elle est capable de traiter automatiquement les TLP reçus depuis l'interface PCI Express et fournit une interface pour émettre quelques requêtes d'entrées-sorties prédéfinies (par exemple, des requêtes d'accès à l'espace d'adressage principal de la mémoire pour le mécanisme d'accès direct à la mémoire). Cette interface est cependant très peu flexible et ne permet pas de générer des requêtes d'entrées-sorties quelconques (en particulier, celles avec un format invalide). Nous avons donc opté pour l'implémentation d'une interconnexion spécifique entre les deux composants, ce qui nous a conduits à développer notre propre *IP Core*. Notre unité logique se présente comme un contrôleur de bus PCI Express que l'on peut connecter à un bus PLBv46 interne à la carte FPGA. Elle offre une interface pour émettre ou recevoir des paquets quelconques sur les bus PCI Express. Ces derniers sont stockés temporairement dans une mémoire partagée.

Le processeur intégré dans le système embarqué constitue le cœur de notre contrôleur et nous permet d'obtenir les propriétés de modularité et de polyvalence voulues pour notre contrôleur. Le logiciel qu'il exécute traite les différents événements des contrôleurs de périphériques que nous avons configurés dans le FPGA. Il se charge en particulier de traiter les paquets reçus par l'unité logique PCI Express et implémente partiellement le protocole PCI Express. Concrètement, lorsqu'un paquet est reçu depuis les bus PCI Express, le pont PCIe vers PLBv46 le stocke temporairement dans une mémoire partagée avec le processeur et dé-

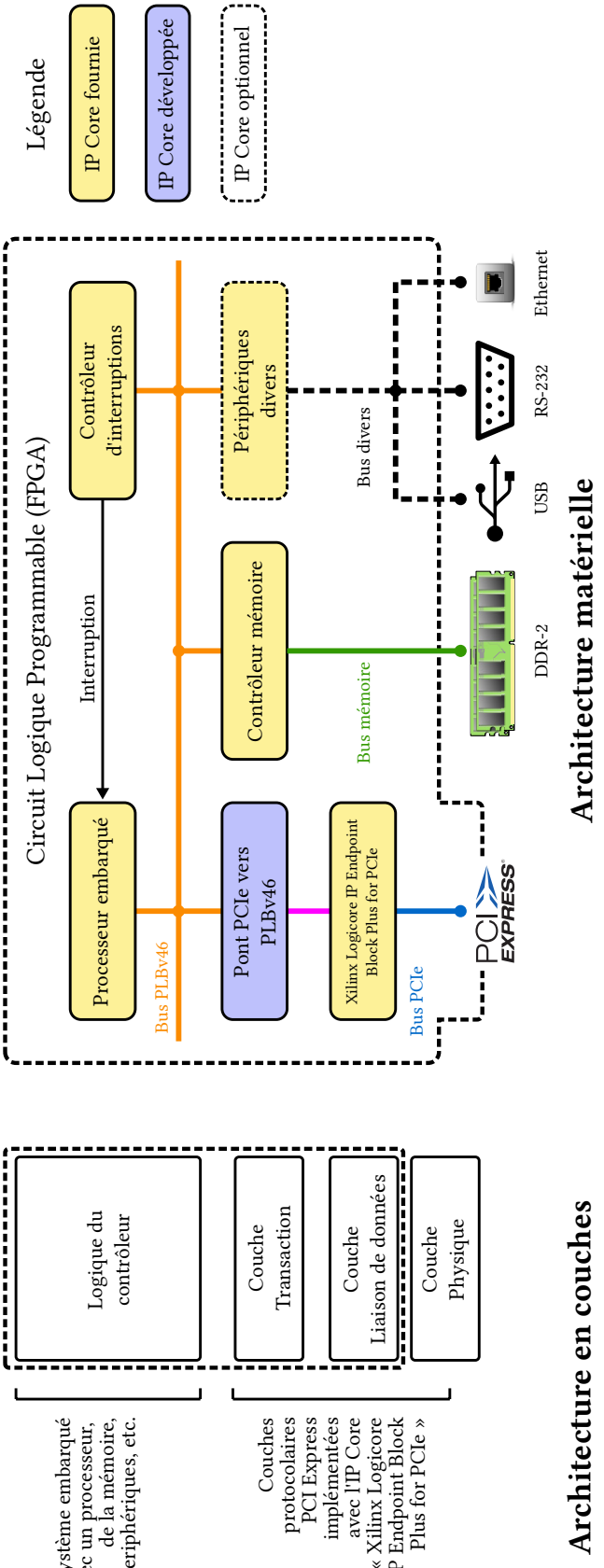

FIGURE 4.3 – Architecture de notre contrôleur d'entrées-sorties IronHide

clenche une interruption pour l'informer de cet événement. Une routine d'interruption vient alors lire ce paquet puis effectue les traitements associés. Elle vérifie que les différents champs de ce paquet (c'est-à-dire les en-têtes et les éventuelles données) sont conformes au protocole et construit un paquet *completion* lorsqu'une réponse est requise. Ce paquet *completion* est copié dans une autre mémoire partagée et est émis automatiquement par le pont PCIe vers PLBv46. Notez que le processeur embarqué peut également utiliser cette seconde mémoire partagée pour initier de lui-même des requêtes d'entrées-sorties valides ou invalides sur les bus PCI Express.

Finalement, l'architecture de notre contrôleur d'entrées-sorties peut être étendue par des contrôleurs de périphériques divers qui peuvent être utilisés pour surveiller le comportement du système embarqué ou pour lui permettre de communiquer avec un système extérieur (par exemple, une machine distante). À terme, il serait intéressant d'utiliser ces contrôleurs divers pour émuler des cartes d'extension PCI Express standards, telles que des cartes Ethernet tout en y insérant une porte dérobée permettant, par exemple, de prendre le contrôle à distance du contrôleur, par une trame réseau spécifique.

4.1.2.3 Implémentation

Cette architecture a été implémentée avec succès sur une carte de développement Pico E-17 [Pico Computing 11] commercialisée par Pico Computing. Il s'agit d'une carte de développement au format Express Card qui dispose d'un FPGA Xilinx Virtex-5 FX70T, d'une mémoire vive (DDR2-SDRAM) de 256 mégaoctets, d'une mémoire flash de 64 mégaoctets et de différents contrôleurs de périphériques (par exemple, plusieurs contrôleurs Ethernet et un contrôleur série RS-232). Le comportement de la carte de développement est entièrement défini par le logiciel écrit en langage C et exécuté par le processeur PowerPC qu'elle embarque. Notre implémentation est *a priori* compatible avec toutes les cartes de développement à base de FPGA Xilinx Virtex-5. En adaptant la configuration spécifique à la carte de développement que nous avons utilisée (c'est-à-dire les contraintes temporelles, de placement, etc.), il est théoriquement possible de porter notre implémentation sur une autre carte de développement. Une adaptation de notre implémentation sur la carte de développement MLX-1000-XC5V [ModularLogix 10] fabriquée par ModularLogix (respectant le format PCI Express) est actuellement en cours.

De la même façon que les contrôleurs d'entrées-sorties malveillants construits à partir de FPGA, le contrôleur IronHide peut être utilisé pour mettre en œuvre différentes attaques par entrées-sorties [31]. Nous l'avons utilisé, par exemple, pour développer plusieurs preuves de concept qui ont été présentées lors de la conférence SSTIC [Lone Sang *et al.* 12b]. En complément des travaux décrits dans ce chapitre, le lecteur intéressé pourra consulter l'annexe C qui détaille plusieurs attaques utilisant le contrôleur IronHide ainsi que les résultats obtenus. Cependant, le caractère polyvalent de notre contrôleur d'entrées-sorties nous permet d'envisager d'autres cas d'utilisation pour celui-ci : injection de fautes, émulation de contrôleur d'entrées-sorties, etc. Nous allons nous en servir principalement en tant qu'outil d'injection de fautes sur les bus d'entrées-sorties. Afin d'automatiser ces injections de fautes, nous mettons en œuvre du *fuzzing*. Il s'agit d'une technique de test largement utilisée dans la recherche de vulnérabilités

[31] En toute rigueur, IronHide ne devrait pas être considéré comme un outil d'attaque. Du point de vue d'un attaquant, il devrait plutôt être considéré comme un moyen de découvrir et de préparer des attaques par entrées-sorties. Les attaques réussies pourraient ensuite être implémentées, par exemple, dans des équipements contrefaits dans lesquels l'attaquant installerait un cheval de Troie et qu'il vendrait à des prix attractifs.

dans les implémentations de systèmes logiciels que nous adaptons à des systèmes matériels. La prochaine section introduit quelques éléments sur cette technique de test.

4.2 Éléments de *fuzzing*

Ces vingt dernières années, le *fuzzing* s'est imposé comme une technique de test intéressante pour révéler des vulnérabilités dans les implémentations de systèmes logiciels. Les origines du *fuzzing* [32] sont quelque peu anecdotiques. Lors d'une nuit d'orage, B. Miller (considéré comme le père du *fuzzing*) tenta de se connecter à un système distant au travers d'une connexion modem et d'exécuter sur celui-ci plusieurs programmes. Les perturbations sur les lignes téléphoniques dues à l'orage ont provoqué la réception de caractères aléatoires sur le terminal distant au lieu des caractères saisis. Il constata alors que ces caractères aléatoires, dans la majorité des cas, provoquaient des erreurs dans les programmes utilisés. Cette découverte a donné naissance à cette technique de test qui a fait l'objet, en automne 1988, de plusieurs projets d'étudiants à l'université Winconsin à Madison. L'objectif de ces travaux était de tester la robustesse des programmes de systèmes d'exploitation de type Unix vis-à-vis d'entrées invalides générées aléatoirement. Les résultats de ces travaux ont été publiés pour la première fois dans [Miller *et al.* 90]. La présente section commence par rappeler les principes de cette technique de test. Nous présentons ensuite les différentes phases qui la constituent et les méthodes utilisées pour sélectionner les entrées de test. La section 4.3 traitera des éléments spécifiques à la mise en œuvre de cette technique de test aux bus d'entrées-sorties.

4.2.1 Principes du *fuzzing*

Bien que les concepts sous-jacents au *fuzzing* soient relativement simples, il n'existe pas dans la littérature de définition qui soit universellement admise. Nous prenons comme référence celle qui a été proposée par P. Oehlert [Oehlert 05] : « *fuzzing is a highly automated testing technique that covers numerous boundary cases using invalid data (from files, network protocols, API calls, and other targets) as application input to better ensure the absence of exploitable vulnerabilities* ».

Ainsi, le *fuzzing* désigne avant tout une technique de test fonctionnel. Il se distingue cependant des autres techniques par les objectifs qu'il vise : le test des fonctions est marginal, l'objectif prédominant étant d'aboutir rapidement à la découverte de vulnérabilités. Il consiste ensuite à injecter des fautes au niveau des entrées du système et à observer ses sorties dans le but de détecter des éventuelles erreurs. Les entrées de test sont sélectionnées de façon à couvrir au maximum les cas limites. Il s'agit, dans la plupart des cas, d'une technique de test en boîte noire car aucune connaissance de la structure interne du système n'est *a priori* nécessaire. La sélection des entrées de test peut cependant être améliorée par une connaissance du comportement interne du système [Ganesh *et al.* 09, Godefroid *et al.* 12]. Dans ce cas, nous parlons plutôt d'approche en boîte grise. Finalement, le *fuzzing* est une technique de test dans laquelle le dépouillement des résultats est (en partie) automatisé. La mise en œuvre de ce processus repose sur des outils, des scripts, des applications ou des plates-formes de test que nous appelons *fuzzers*. Nous distinguons deux types de *fuzzers* : ceux qui se basent sur des techniques de mutation et ceux qui reposent sur la synthèse. Les *fuzzers* par mutation créent des entrées de test à partir d'échantillons d'entrées réelles du système sous test auxquelles ils appliquent des transformations. Pour cette raison, ils ne nécessitent pas la connaissance au préalable du for-

[32] Le terme *fuzzing* est issu de la contraction des termes anglo-saxon *fuzzy* (littéralement, flou) et *testing*.

mat de ces entrées. *A contrario*, les *fuzzers* par synthèse construisent des entrées de test à partir d'une modélisation de ces entrées.

4.2.2 Phases du *fuzzing*

Quel que soit le système auquel nous nous intéressons, il est possible de distinguer plusieurs phases dans la mise en œuvre du *fuzzing*. Cette sous-section les décrit succinctement.

1. *Identification du système sous test.* Avant de mettre en œuvre une technique de test quelconque, il est important de définir le système sur lequel les tests vont porter. Cette étape préliminaire permet généralement d'identifier les différentes approches qui peuvent être adoptées pour effectuer du *fuzzing* sur le système sous test.
2. *Identification des entrées du système.* Une fois le système sous test défini, une seconde étape consiste à énumérer l'ensemble de ses interfaces de service et à en sélectionner quelques unes pour interagir avec le système. Il est alors nécessaire d'identifier le format de leurs entrées ainsi que les éléments dans ces entrées qui, une fois modifiés, provoquent potentiellement des fautes dans le système sous test.
3. *Sélection des entrées de test.* Le test exhaustif est généralement impossible et il est nécessaire de sélectionner de manière pertinente un sous-ensemble du domaine d'entrée qui permette de couvrir l'ensemble des parties du système que l'on souhaite tester. En fonction de la connaissance du système sous test, il est possible de sélectionner les entrées de test de différentes manières. Elles sont discutées dans la sous-section 4.2.3.
4. *Exécution des entrées de test.* L'étape qui suit la sélection des entrées de test est l'exécution de ces entrées. Elle consiste à soumettre au système sous test ces entrées en espérant qu'un certain nombre d'entre elles déclenchent une erreur lors de leur exécution.
5. *Analyse des éventuelles erreurs.* Enfin, lorsque nous observons des erreurs, il est essentiel de les analyser et d'identifier leurs causes afin de proposer des correctifs ou au contraire pour tenter de les exploiter.

Dans la sous-section suivante, nous nous intéressons plus particulièrement aux différentes approches qui peuvent être mises en œuvre dans un *fuzzer* pour sélectionner des entrées de test.

4.2.3 Méthodes de *fuzzing*

La classification en *fuzzers* basés sur la mutation et en *fuzzers* basés sur la synthèse d'entrées de test peut être raffinée à son tour en plusieurs méthodes [Sutton *et al.* 07] qui sont présentées dans cette sous-section.

4.2.3.1 Génération d'entrées de test prédéfinies

Les fonctions d'un système sont généralement décrites dans un document de référence qui peut être un cahier des charges ou, plus couramment, un document de spécification. Afin de vérifier la conformité du système aux exigences, ce document contient souvent un jeu d'entrées de test qui permet de vérifier toutes les conditions de fonctionnement du système. Il s'agit là d'une première façon de sélectionner des entrées de test. La mise en œuvre de cette méthode exige un travail considérable en amont, notamment pour intégrer les entrées de test au *fuzzer*. Elle présente l'avantage de pouvoir être utilisée de façon uniforme pour vérifier plusieurs

implémentations d'un même système par rapport à sa spécification. Par ailleurs, puisque les entrées de test sont générées de façon déterministe, il est possible de rejouer plusieurs fois le même jeu d'entrées de test et de conserver une trace précise de chaque test. Cependant, cette méthode de sélection d'entrées de test peut s'avérer limitée si le document de spécification n'est pas suffisamment précis. En effet, il est possible qu'il omette plusieurs conditions de fonctionnement du système et, éventuellement, certaines violations. Il est souvent nécessaire de compléter cette méthode par des entrées de test aléatoires pour couvrir un domaine d'entrée plus important.

4.2.3.2 Génération d'entrées de test aléatoires

Une autre manière de sélectionner des entrées de tests est de les générer aléatoirement. L'idée consiste alors à fournir au système des entrées aléatoires de façon à influer (aléatoirement) sur l'enchaînement et le comportement de ses fonctions. Il s'agit probablement de la méthode la plus rapide à mettre en œuvre et celle qui nécessite le moins d'effort car elle implique uniquement l'utilisation d'un générateur pseudo-aléatoire. Elle est, cependant, de loin la moins efficace. En effet, un système bien conçu effectue systématiquement un pré-traitement de ses entrées afin de déterminer si celles-ci sont valides ou, au contraire, invalides. Pour cela, il applique des règles lexicales et grammaticales à ses entrées et rejette les entrées qui ne sont pas conformes à ces règles. En raison de leur caractère aléatoire, un grand nombre d'entrées ne seront pas pertinentes et n'iront pas au delà de ce pré-traitement. Il convient cependant de noter que cette méthode conduit parfois à des résultats intéressants. De nombreuses vulnérabilités logicielles ont été ainsi mises au jour [Miller *et al.* 90, Forrester et Miller 00, Miller *et al.* 07]. Il est à noter qu'il existe une variante de cette méthode [Holzleitner 09] dite à *feedback*, qui modifie la génération des entrées de test en fonction des résultats des tests précédents. Cette approche permet ainsi d'identifier plus rapidement des cas intéressants.

4.2.3.3 Mutation d'entrées de test

La mutation d'entrées de test nécessite au préalable la capture d'une ou plusieurs entrées valides. Celles-ci sont utilisées comme entrées de référence à partir desquelles sont produites de nouvelles entrées de test qui sont obtenues par plusieurs transformations successives. Les inversions de bits (en anglais, *bit-flips*), la substitution de valeurs et l'insertion de données aléatoires sont quelques exemples de transformations qui sont couramment effectuées. La mutation peut être opérée de façon manuelle. Un opérateur humain se charge alors d'effectuer les transformations sur l'entrée de référence. Cette approche est particulièrement adaptée pour des systèmes de petite taille et pour des tests fonctionnels rapides. La mutation peut également être automatique [Crouzet *et al.* 06]. Un algorithme est alors chargé d'appliquer les transformations à la place de l'opérateur humain. Étant donné que cette méthode de sélection d'entrées de test repose sur des entrées de test valides et que les transformations sur celles-ci sont opérées automatiquement, cette méthode est généralement plus efficace que celles qui ont été précédemment évoquées. Elle ne s'adapte cependant pas à tout type d'entrées, en particulier lorsqu'il existe une corrélation entre plusieurs de ses éléments (par exemple, un condensat, un CRC, etc.).

4.2.3.4 Synthèse d'entrées de test à partir d'un modèle

La synthèse d'entrées de test à partir d'un modèle est la méthode qui présente le plus d'avantages. Elle consiste à générer un jeu d'entrées de test à partir d'un modèle des entrées du système. Il s'agit également de la méthode qui exige le plus de travail de préparation. En effet, un travail de recherche est nécessaire au préalable pour comprendre la structure des entrées du système. Celle-ci est ensuite décrite selon une grammaire afin de rendre automatisable la génération des entrées de test. Au cours de cette phase, l'analyste identifie les parties des entrées qui doivent rester statiques, celles qui sont susceptibles de varier et les éventuelles relations qui les lient. Le *fuzzer* analyse alors le modèle établi, génère des entrées de test et les fournit au système sous test. Il convient de préciser que le succès de cette méthode dépend de la capacité de l'analyste à identifier les éléments dans les entrées qui sont susceptibles de conduire à des erreurs.

Chacune de ces approches de sélection de cas de tests a ses avantages et ses inconvénients. Dans le but de couvrir un domaine d'entrée plus large, il est important de combiner ces différentes méthodes dans un *fuzzer*. La prochaine section présente le *fuzzer* que nous avons développé pour automatiser la mise en œuvre (et l'étude) des attaques par entrées-sorties.

4.3 Mise en œuvre du *fuzzing* sur les bus d'entrées-sorties

Un grand nombre de *fuzzers* peuvent être identifiés dans la littérature. La plupart d'entre eux sont spécifiques à un type de systèmes sous test particulier (par exemple, des programmes, des protocoles, des bibliothèques de fonctions, etc.). Il existe cependant des *fuzzers* plus ou moins génériques, tel que Peach [Eddington 12], qui peuvent être utilisés pour différents types de systèmes. Malheureusement, aucun d'eux ne nous satisfait pleinement pour notre étude. En effet, certains outils intéressants ne sont plus à jour et la prise en main d'autres outils est parfois difficile. Aussi, avons-nous décidé de créer notre propre *fuzzer*. Nous introduisons brièvement son architecture et nous précisons quelques élements sur son implémentation avant de discuter des résultats des expérimentations que nous avons menées sur plusieurs *chipsets* Intel.

4.3.1 Architecture du *fuzzer*

La figure 4.4 présente l'architecture du *fuzzer* que nous avons mis en place pour notre étude. Il se compose de plusieurs éléments : un sélecteur d'entrées de test, un contrôleur de tests, un outil d'injection de fautes sur les bus d'entrées-sorties, un oracle et plusieurs agents. Cette sous-section détaille le rôle de chacun de ses éléments.

Le sélecteur d'entrées de test est une des parties les plus importantes du *fuzzer*. Il se charge de créer l'ensemble des entrées de test que le contrôleur de tests va soumettre au système sous test. Le contrôleur de tests est également un élément important car il permet d'automatiser l'exécution des entrées de test. À cet effet, il transmet séquentiellement chaque entrée de test à exécuter à l'outil d'injection de fautes (dans notre cas, il s'agit du contrôleur d'entrées-sorties IronHide) et envoie également à l'oracle le résultat attendu pour ce test afin que celui-ci puisse prononcer son verdict sur le test exécuté. Ce verdict se base sur la comparaison de la réponse attendue, fournie par le contrôleur de tests, à la réponse renvoyée par le système sous test. Lorsqu'un verdict est impossible à l'aide de ces deux éléments, l'oracle peut s'aider d'informations complémentaires fournies par les différents agents déployés sur le système sous test. À nouveau, si un verdict n'est toujours pas possible, l'oracle le signale à un opérateur humain pour

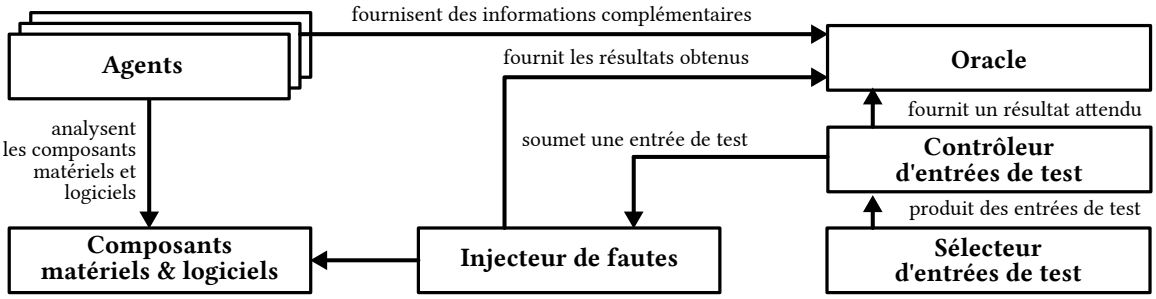

FIGURE 4.4 – Architecture du *fuzzer*

qu'il puisse effectuer un dépouillement manuel ultérieurement. Une fois le verdict prononcé, le contrôleur de tests transmet à nouveau à l'outil d'injection de fautes une nouvelle entrée de test. Ce cycle se termine lorsqu'il n'y a plus d'entrées de test à exécuter. Afin de pouvoir vérifier ultérieurement les résultats obtenus, nous avons journalisé dans notre *fuzzer* l'ensemble des tests effectués, les verdicts prononcés ainsi que les informations complémentaires remontées par les agents.

Avant de discuter des expérimentations que nous avons effectuées avec notre *fuzzer*, précisons quelques éléments sur son implémentation. La majorité des éléments du *fuzzer* ont été implémentés dans le langage Python. En particulier, nous nous sommes appuyés sur l'outil de manipulation de paquets réseau Scapy [Biondi 13] pour générer les *Transaction Layer Packets* valides et invalides. Puisque ce dernier n'implémente pas nativement le protocole PCI Express, nous l'avons étendu pour intégrer ce nouveau protocole. En ce qui concerne la sélection des entrées de test, nous avons favorisé l'approche par synthèse puisqu'elle donne généralement de meilleurs résultats [33]. Nous avons également combiné plusieurs méthodes afin de couvrir un domaine d'entrée plus important :

1. *Entrées de test prédéfinies.* Les spécifications du standard PCI Express sont relativement complètes. Elles contiennent des exemples d'entrées de test et les comportements attendus pour les contrôleurs d'entrées-sorties vis-à-vis de celles-ci. Ces entrées de test couvrent non seulement les accès d'entrées-sorties valides, mais également certains accès d'entrées-sorties invalides. Il peut être intéressant de les implémenter, puis de les exécuter afin de vérifier la conformité de l'implémentation du PCI Express par les *chipsets*.
2. *Entrées de test générées automatiquement.* La majorité des TLP respectent un format de TLP défini par la spécification. La sémantique et les plages de valeurs possibles pour chaque champ étant précisées dans les specifications, il est possible de générer des entrées de test automatiquement. Dans ce cas, il est important de générer à la fois des TLP valides pour vérifier l'implémentation du PCI Express par le *chipset*, mais également des TLP invalides, pour évaluer la robustesse de cette implémentation. Certains champs nous paraissent particulièrement intéressants lors de la génération d'entrées de test :
   — les champs réservés : ils correspondent généralement à des emplacements destinés à un usage futur. Un contrôleur d'entrées-sorties source est supposé s'assurer que ces champs soient positionnés à zéro. Quant au contrôleur d'entrées-sorties destinataire, ce dernier se doit d'ignorer les valeurs contenues dans ces champs. Il convient

[33] Dans [Miller et Peterson 07], les auteurs ont comparé les deux approches de génération d'entrées de test. En se focalisant sur les formats de fichiers *Portable Network Graphics* (PNG), ils ont estimé que l'approche par synthèse était 80% plus efficace que l'approche par mutation.

alors de s'assurer que les implémentations de contrôleurs d'entrées-sorties actuelles ignorent effectivement ces champs en y injectant des valeurs non nulles à l'émission d'un TLP.
- le champ `EP` : ce champ dans l'en-tête du TLP permet d'indiquer que le TLP a été « empoisonné » et qu'il n'est pas valide. À la réception d'un tel paquet, un contrôleur est censé ignorer le paquet. Il convient alors de s'assurer que le TLP est réellement ignoré lorsque ce champ est positionné.
- le champ `TD` : ce bit TD indique si le TLP contient un ECRC. La taille du paquet total est alors calculée à l'aide de la longueur des données annoncée dans le paquet auquel se rajoutent la taille de l'en-tête et la taille de l'ECRC. Positionner ce champ alors que le paquet ne contient pas d'ECRC peut éventuellement provoquer des effets de bord.
- le champ `Length` : ce champ indique la quantité de données transportées dans le TLP. Des débordements de tampons peuvent survenir, par exemple, lorsque la longueur annoncée dans le paquet est plus faible que les données réellement envoyées.
- le champ `Address` : les spécifications des bus PCI Express requièrent que les accès d'entrées-sorties se fassent par pages (4096 octets). Par exemple, la lecture de 4096 octets de données à partir de l'adresse `0x0ff0` nécessite au minimum deux accès : un premier de 16 octets vers l'adresse `0x0ff0` (correspondant à la première page), puis un second vers l'adresse `0x1000` pour les 4080 octets restants.

3. *Entrées de test aléatoires.* Le format de certaines parties des TLP n'est pas connu. C'est le cas typiquement de certains TLP de type message. Certains champs sont laissés à la discrétion du constructeur pour véhiculer, par exemple, des messages de gestion d'énergie ou des erreurs spécifiques à leur plate-forme. Pour ces champs dont la sémantique n'est pas connue, la génération aléatoire des entrées de test peut être intéressante. L'idée est alors d'affecter à ces champs des valeurs aléatoires afin de révéler, par exemple, un bogue matériel ou une fonctionnalité cachée.

Au cours de nos expérimentations, nous nous sommes concentrés uniquement sur le test de l'implémentation du protocole PCI Express par les contrôleurs d'entrées-sorties. Afin de limiter le spectre de notre recherche de vulnérabilités, nous n'avons délibérément pas tenu compte des fonctionnalités spécifiques à chacun [34] ni des protocoles dédiés qui permettent d'y accéder.

4.3.2 Expérimentations et résultats

La figure 4.5 présente l'environnement que nous avons mis en place pour nos expérimentations. Les différents composants qui constituent le *fuzzer* ont été répartis sur deux machines qui jouent respectivement le rôle d'une *cible* et d'un *moniteur*. Schématiquement, la machine *cible* représente la machine sous test pour laquelle nous souhaitons étudier le comportement du *chipset* en réponse à des requêtes d'entrées-sorties diverses. Elle contient l'outil d'injection de fautes et plusieurs agents. La machine *moniteur* correspond à l'entité qui contrôle l'exécution des entrées de test. Elle se charge de définir les entrées de test, de les soumettre à la machine *cible* et de vérifier leurs résultats. Elle contient également les principaux composants du *fuzzer*.

Le contrôleur de tests récupère les entrées générées par le sélecteur d'entrées de test et les soumet à l'outil d'injection de fautes au travers du réseau qui est connecté au système sous test par un bus PCI Express. Nous l'avons programmé pour servir de serveur mandataire (*proxy*)

[34] Dans le cas d'une carte réseau, ces fonctionnalités spécifiques peuvent correspondre, par exemple, à l'émission ou la réception de trames sur le réseau, aux transferts de ces trames entre la carte réseau et la mémoire centrale, etc.

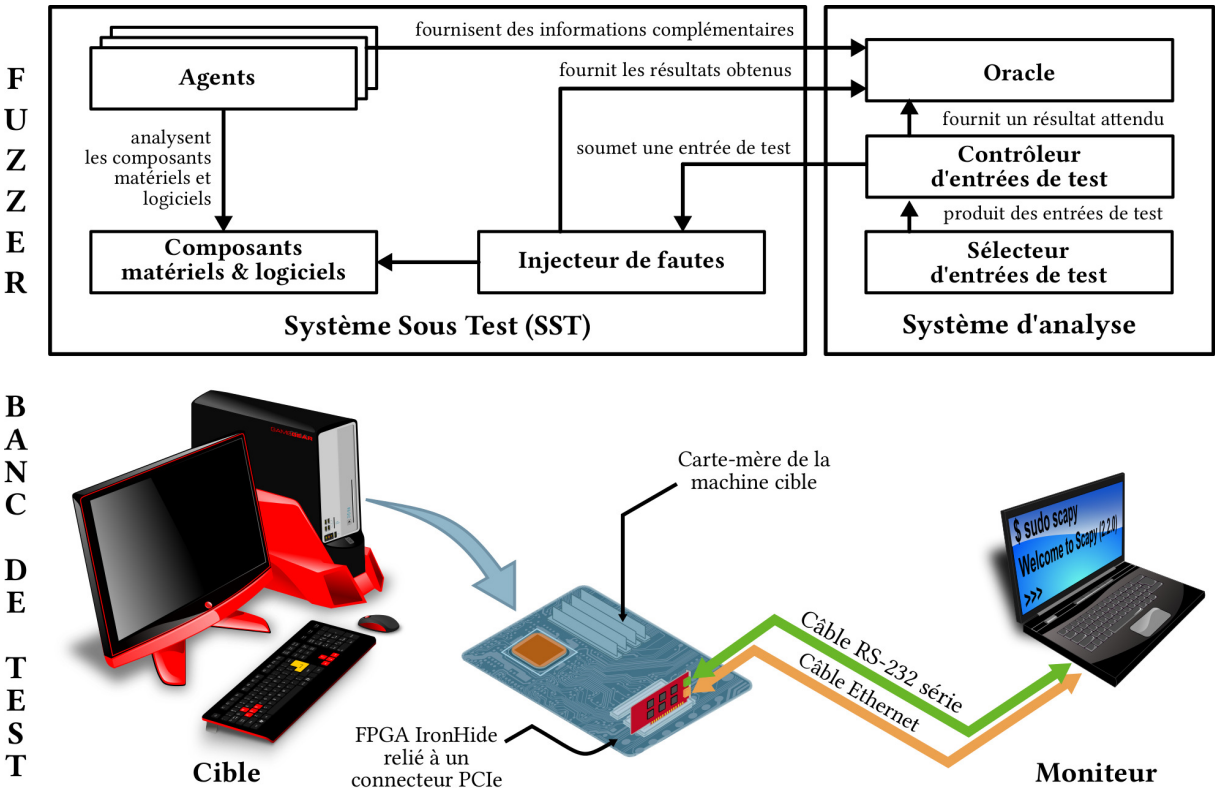

FIGURE 4.5 – Plate-forme expérimentale

PCI Express pour le contrôleur de tests : il émet tous les paquets reçus depuis l'interface Ethernet en provenance de la machine *moniteur* vers les bus PCI Express de la machine *cible* et renvoie vers la machine *moniteur* les réponses éventuelles. Le logiciel embarqué dans IronHide implémente une pile TCP/IP et contient un serveur TCP qui relaie les paquets entre le contrôleur de tests et les bus d'entrées-sorties du système sous test. Il est configuré avec une adresse IP statique et le serveur TCP embarqué attend de nouvelles connexions sur un port que nous avons défini. Il est à noter que nous aurions pu gagner en performance en choisissant l'utilisation d'un serveur UDP plutôt qu'un serveur TCP. Nous avons préféré utiliser le protocole TCP, de façon à fiabiliser les échanges et ainsi éviter les problèmes liés aux pertes de paquets réseau par congestion du contrôleur IronHide ou de la machine *moniteur* lors de nos expérimentations.

Nous avons procédé à des expérimentations sur différents *chipsets* Intel récents qui sont rappelés à la table 4.2. Précisons que nous avons repris le même parc de machines que lors de notre étude des attaques *peer-to-peer* (cf. section 3.3). Pour chaque machine expérimentée, nous rappelons le modèle de *chipset* concerné et le modèle de machine utilisé. Les expérimentations sur chacune de ces machines ont été faites selon deux configurations matérielles différentes. Dans la première configuration matérielle, nous avons volontairement désactivé l'ensemble des technologies matérielles, telles qu'Intel VT-d et les ACS, permettant de contrôler les entrées-sorties. Nous avons ensuite rejoué ces mêmes tests en présence de ces technologies matérielles. La comparaison des observations permet alors d'évaluer la robustesse de ces technologies matérielles.

Nous avons produit environ 65 000 entrées de test que nous avons jouées sur chaque machine. En fonction des machines testées, la campagne de *fuzzing* a duré entre 2 jours et 2 semaines. Le temps nécessaire pour exécuter chaque campagne de tests dépend directement du

	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 4.2 – Liste des *chipsets* expérimentés

nombre de fautes détectées sur une machine. En effet, puisque certains tests peut provoquer un blocage de la machine, il est nécessaire, dans ces cas précis, de redémarrer la machine testée.

La table 4.3 consigne quelques résultats que nous avons obtenus dans ces deux configurations matérielles. Par souci de lisibilité, nous avons regroupé les machines qui présentent des comportements similaires. Nous avons également rassemblé les tests par classe de transactions et par élément du TLP qui a été modifié. Certains de ces tests ont été marqués comme non-applicables soit parce que les accès ne sont pas autorisés par le *chipset* (cas des machines 0, 1, 2 et 3), soit parce que la configuration du *chipset* n'a pas pu être modifiée pour le permettre (cas de la machine 4).

L'analyse des résultats révèle que le comportement des machines 0, 1, 2 et 3 diffère du comportement de la machine 4. Pour le premier groupe de machines, nous remarquons qu'à la fois des paquets valides et des paquets invalides peuvent provoquer un déni de service du système. Il est intéressant de préciser que le déni de service affecte uniquement la partie des bus d'entrées-sorties concernée par la transaction et que le reste des bus d'entrées-sorties reste fonctionnel. Cependant, puisque le système d'exploitation n'est plus en mesure d'interroger les composants connectés à ces bus d'entrées-sorties, les pilotes des composants matériels qui ne sont pas conçus pour gérer ce type de situation entrent dans un état d'erreur et provoquent rapidement le blocage complet du système. Un redémarrage du système est alors nécessaire.

En enquêtant sur les causes de ces dénis de service, nous nous sommes rendu compte que ceux-ci résultent soit de l'utilisation de fonctionnalités auxiliaires prévues par les spécifications PCI Express mais qui n'ont pas été implémentées, soit d'une mauvaise configuration de la plate-forme matérielle. Pour ce qui est de la machine 4, elle parvient à détecter la majorité des paquets invalides et à signaler des erreurs au système d'exploitation qui les journalise. Cette différence de comportement entre les machines s'explique par les extensions du standard PCI Express qu'elles implémentent. Les machines 0, 1, 2 et 3 n'implémentent aucune extension particulière du PCI Express alors que la machine 4 dispose, en plus d'Intel VT-d et des ACS, une extension supplémentaire appelée *Advanced Error Reporting* (AER) qui permet de vérifier que les paquets PCI Express qui transitent sur les bus d'entrées-sorties sont conformes à la spécification. Cette extension effectue des vérifications sur différentes couches protocolaires. Au niveau de la couche transaction, celle-ci est notamment capable de distinguer et remonter différentes erreurs sur les paquets au système d'exploitation. Elle est capable de détecter par exemple un ECRC invalide, une communication avec un contrôleur d'entrées-sorties inexistant, des dépassements de tampons ou des TLP malformés. L'implémentation d'une telle extension montre une réelle volonté de la part des constructeurs de *chipsets* à prendre en considération les attaques par entrées-sorties.

Il convient de remarquer qu'il est normal d'obtenir, par cette approche d'analyse de vulnérabilités, uniquement des conséquences de type dénis de service car nous avons, dans nos expérimentations, injecte des données aléatoires. Pour aller plus loin, il serait intéressant d'ana-

lyser plus finement ces dénis de services afin d'identifier si les vulnérabilités dont ils sont issus peuvent être exploitées autrement. Étant donné que nous nous sommes focalisés sur le test des implémentations matérielles, un rapprochement auprès des constructeurs semble indispensable afin d'identifier si ceux si résultent de choix ou d'erreurs d'implémentation.

4.4 Conclusion

Au lieu de suivre la démarche classique en analyse de vulnérabilités, nous avons proposé, dans ce chapitre, une démarche complémentaire qui nous a permis de découvrir les vulnérabilités de façon systématique (sinon exhaustive). Elle consiste à injecter des fautes sur les bus d'entrées-sorties dans le but de simuler les conséquences d'un large spectre de fautes. À cet effet, nous avons développé un outil, baptisé IronHide, capable de générer tout type de trames sur les bus d'entrées-sorties. Celui-ci se présente comme un contrôleur d'entrées-sorties classique et a été implémenté à l'aide de technologies de logique programmable tels que les FPGA.

Notre contrôleur IronHide nous permet d'avoir les mêmes pouvoirs qu'un attaquant installant une porte dérobée dans un contrôleur qu'il conçoit ou dans lequel il a découvert une vulnérabilité le lui permettant. Nous avons ensuite utilisé cet outil d'injection de fautes pour mettre en œuvre différentes attaques par entrées-sorties. Afin d'automatiser ces attaques, nous nous sommes inspirés des techniques de test de systèmes logiciels. Nous avons appliqué à plusieurs systèmes matériels les techniques de *fuzzing* généralement utilisées dans la recherche de vulnérabilités dans des implémentations de systèmes logiciels. L'analyse des résultats (préliminaires) que nous avons obtenus lors de nos expérimentations révèle qu'à la fois les paquets valides et les paquets invalides peuvent provoquer un déni de service. Ceux-ci résultent alors soit de l'utilisation de fonctionnalités auxiliaires prévues par les spécifications PCI Express mais qui n'ont pas été implémentées, soit d'une mauvaise configuration de la plate-forme matérielle. Finalement, nous avons constaté que l'extension *Advanced Error Reporting* (AER) implémentée dans certaines plates-formes matérielles semble les immuniser aux attaques que nous avons mises en œuvre. Cette extension effectue des vérifications sur différentes couches protocolaires. Au niveau de la couche transaction, celle-ci est notamment capable d'identifier et remonter au système d'exploitation différentes erreurs sur les paquets (par exemple, ECRC invalide, communication avec un contrôleur d'entrées-sorties inexistant, dépassements de tampons et TLP malformés). Son implémentation dans certains *chipsets* montre une réelle volonté de la part des constructeurs à prendre en considération les attaques par entrées-sorties.

Entrées de test		Technologies matérielles inactives		Technologies matérielles actives	
Type de transaction	**Champs modifiés**	**Machine 1, 2, 3, 4**	**Machine 5**	**Machine 1, 2, 3, 4**	**Machine 5**
Générique	*Format* et *Type*	-	détecté	-	détecté
	Traffic Class (TC)	DoS	-	DoS	-
	TLP Digest (TD)	DoS	détecté	DoS	détecté
	TLP Poisonned (EP)	-	-	-	-
	Attribute (Attr)	-	détecté	-	détecté
	Address Type (AT)	-	détecté	-	détecté
	Length	DoS	détecté	-	détecté
	champs réservés	-	détecté	-	détecté
Memory et I/O	*Requester ID*	DoS	détecté	DoS	détecté
	Tag	-	-	-	-
	Last DW Byte Enable (LDBE)	DoS	détecté	DoS	détecté
	First DW Byte Enable (FDBE)	DoS	détecté	DoS	détecté
	Address	-	-	-	-
	champs réservés	-	détecté	-	détecté
Configuration	*Requester ID*	non-applicable	non-applicable	non-applicable	non-applicable
	Tag	non-applicable	non-applicable	non-applicable	non-applicable
	Last DW Byte Enable	non-applicable	non-applicable	non-applicable	non-applicable
	First DW Byte Enable	non-applicable	non-applicable	non-applicable	non-applicable
	Address	non-applicable	non-applicable	non-applicable	non-applicable
	champs réservés	non-applicable	non-applicable	non-applicable	non-applicable
	Completer ID	non-applicable	non-applicable	non-applicable	non-applicable
	Register Number	non-applicable	non-applicable	non-applicable	non-applicable
Message	*Requester ID*	DoS	détécté	DoS	détecté
	Tag	-	-	-	-
	Message Code	-	-	-	-
	champs spécifiques	-	-	-	-

TABLE 4.3 – Quelques résultats expérimentaux

Conclusion générale

À en juger par les évolutions actuelles, les systèmes informatiques vont vraisemblablement continuer à s'immiscer davantage dans notre quotidien. Le développement de ces systèmes de plus en plus ubiquitaires s'accompagne naturellement de nombreux défis technologiques tels que la mobilité, l'évolutivité et l'autonomie, la réactivité, etc. Parmi ces défis, la question cruciale de la sécurité reste un enjeu majeur en raison de la dématérialisation croissante de l'information et des risques de plus en plus importants liés à la complexité de ces systèmes.

Ce manuscrit de thèse traite de la sécurité en vie opérationnelle des systèmes informatiques sur étagère. Les travaux que nous avons réalisés se concentrent principalement sur la protection de ces systèmes contre les attaques informatiques impliquant non seulement les composants logiciels mais également les composants matériels. Ils suscitent actuellement un intérêt croissant autant dans le monde académique que le monde industriel en raison de la prolifération de ce type de menaces informatiques qui se font, aujourd'hui, de plus en plus oppressantes.

Afin de cerner cette problématique complexe, nous avons commencé par élaborer un modèle d'attaques sur lequel nous nous sommes basés pour identifier les attaques dans un système informatique. Ensuite, nous avons instancié ce modèle vis-à-vis des systèmes auxquels nos travaux se sont intéressés, à savoir les ordinateurs compatibles PC. Cette instance nous a alors permis, d'une part, de positionner les attaques que nous avons analysées par rapport à celles qui sont habituellement considérées et, d'autre part, d'identifier plusieurs scénarios d'attaque qui, jusqu'à présent, ont été négligés pour des raisons diverses.

Notre démarche d'analyse des attaques informatiques et des contre-mesures associées s'est orientée ensuite vers une étude expérimentale. Nous avons supposé que le problème des attaques (exclusivement) logicielles était résolu, nous poussant alors à nous concentrer spécifiquement sur celles qui impliquent à la fois des composants logiciels et matériels et, plus particulièrement, sur les attaques par entrées-sorties qui sont encore relativement peu explorées. Ces attaques atypiques détournent les entrées-sorties, plus précisément les mécanismes de communication entre les processeurs et les contrôleurs d'entrées-sorties, à des fins malveillantes. En nous appuyant sur notre modèle, nous les avons étudiées selon deux approches complémentaires : une première approche traditionnelle plutôt empirique, consistant à rechercher et identifier une vulnérabilité, développer une preuve de concept qui permette de l'exploiter puis proposer des contre-mesures ; et une seconde approche, complémentaire à la première, plus systématique et s'inspirant des techniques de recherche de vulnérabilités dans les composants logiciels. Ces deux approches ont permis de révéler, d'une part, des vulnérabilités dans les plates-formes matérielles actuelles et, d'autre part, des insuffisances dans les contre-mesures matérielles aux attaques par entrées-sorties. Pour chacune d'elles, nous avons formulé quelques recommandations pour se protéger d'attaques qui les exploiteraient.

1 Contributions

Nous trouvons dans la littérature différentes taxonomies d'attaques. Seulement, aucune d'elles ne nous a pleinement satisfaits pour notre étude. En effet, elles n'ont pas permis de représenter l'ensemble des attaques sur un système informatique que nous avons considérées. En nous inspirant de la logique de ces taxonomies d'attaques, nous avons proposé, dans un premier temps, une caractérisation de ces attaques informatiques (cf. chapitre 1). Pour cette analyse, nous sommes partis de la définition d'un système informatique, et nous avons conjecturé que ces attaques peuvent intervenir à plusieurs niveaux d'abstraction du système, notamment les composants logiciels, les composants matériels, les canaux de communication et l'environnement du système informatique. En décomposant les attaques en séquences d'actions élémentaires, et en étudiant celles qui sont malveillantes, autrement dit leurs attaques élémentaires, nous avons pu identifier les éléments sur lesquels portent ces attaques ainsi que les vecteurs d'attaque utilisés. Nous sommes parvenus à une classification de ces actions malveillantes selon trois axes complémentaires – le niveau d'abstraction du système sur lequel porte l'attaque, l'objet de l'attaque et le vecteur d'attaque utilisé sur cet objet – et cette classification sous-tend le reste de notre manuscrit.

Afin d'étudier méthodiquement les attaques dans un système informatique, nous avons ensuite proposé un modèle d'attaques (cf. chapitre 2). Le modèle que nous avons proposé permet de couvrir des scénarios d'attaque qui impliquent éventuellement plusieurs niveaux d'abstraction d'un système informatique. Dans ce modèle, nous avons considéré les composants matériels dans un système informatique comme des systèmes à part entière qui interagissent avec d'autres composants matériels (donc d'autres systèmes). Nous avons modélisé leur structure interne comme étant composé de plusieurs sous-systèmes – logiciel, logique (câblée), configuration et communication – interdépendants. En nous appuyant sur notre précédent travail de caractérisation des attaques, nous avons déterminé l'ensemble des actions élémentaires et des attaques élémentaires au sein d'un composant matériel. Ce modèle, combiné aux interactions entre les différents composants, nous a alors permis de déterminer les séquences d'attaques qui sont possibles dans un système informatique. Enfin, nous avons validé le modèle obtenu en le confrontant à plusieurs exemples réels d'attaques complexes.

Nous nous sommes alors focalisés sur le domaine des attaques par entrées-sorties qui, jusqu'à présent, n'a été que relativement peu exploré. Vis-à-vis de ces attaques, nous constatons que les contre-mesures, qui étaient jusqu'à présent presqu'exclusivement logicielles, ne suffisent plus et qu'une bonne connaissance de la plate-forme matérielle est nécessaire pour une meilleure maîtrise de la sécurité d'un système informatique. C'est pour cette raison que nos approches d'analyse de ces attaques informatiques se sont orientées résolument vers des études expérimentales. Celles-ci se sont principalement concentrées sur les systèmes informatiques compatibles PC. Notre première approche (cf. chapitre 3) s'apparente aux approches qui sont traditionnellement adoptées pour étudier les attaques informatiques : en s'aidant de documents de spécification, le système informatique ciblé est analysé en détail dans le but d'identifier des vulnérabilités qui sont éventuellement exploitables. En dépit des efforts conséquents qui ont été entrepris ces dernières années par différents acteurs de l'industrie informatique (constructeurs, consortiums de normalisation, etc.) pour tenter d'endiguer les problèmes liés aux utilisations des entrées-sorties à des fins malveillantes, nous avons pu constater plusieurs insuffisances dans les contre-mesures matérielles implémentées dans les plates-formes actuelles, notamment dans la technologie matérielle de virtualisation Intel VT-d et dans les *Access Control Services* [Lone Sang *et al.* 10a, Lone Sang *et al.* 10b, Lone Sang *et al.* 11]. De façon délibérée, nous avons ensuite exploité avec succès ces insuffisances afin de formuler quelques

recommandations pour se protéger d'attaques qui en abuseraient.

Finalement, notre seconde approche (cf. chapitre 4) complète la première en permettant de couvrir, de manière automatisée et systématique, davantage d'attaques par entrées-sorties sans requérir une pré-connaissance pointue du système informatique évalué. En nous inspirant des techniques de test et de recherche de vulnérabilités dans les systèmes logiciels, nous avons essayé d'adapter les techniques de *fuzzing* à des systèmes matériels. À cet effet, nous avons mis au point un contrôleur d'entrées-sorties spécifique en utilisant les technologies de logique programmable. Il se distingue des contrôleurs FPGA existants de par le fait qu'il est entièrement programmable et qu'il est capable de générer tout type de trame sur les bus d'entrées-sorties. Notre prototype, baptisé *IronHide*, nous a permis d'obtenir les mêmes pouvoirs qu'un attaquant installant une porte dérobée dans un contrôleur d'entrées-sorties qu'il conçoit ou dans lequel il a découvert une vulnérabilité le lui permettant [Lone Sang *et al.* 12b]. L'analyse des premiers résultats que nous avons obtenus par cette approche nous permet d'ores et déjà de conclure que de nombreuses plates-formes matérielles compatibles PC (notamment, les *chipsets* Intel d'avant 2009) ne sont pas tolérantes aux fautes injectées depuis les bus d'entrées-sorties. L'intégration de contre-mesures supplémentaires, notamment l'*Advanced Error Reporting*, dans les plates-formes matérielles plus récentes semblent montrer une réelle volonté, de la part des fabricants de *chipsets*, de prendre en considération des attaques par entrées-sorties.

2 Travaux futurs

Les travaux présentés dans ce manuscrit de thèse ouvrent des axes d'activité complémentaires intéressants. Outre les pistes d'amélioration d'ordre technique que nous avons déjà mentionnées en fin de chaque chapitre, plusieurs axes de recherche que nous avons envisagés s'inscrivent dans le prolongement de nos travaux.

2.1 Formalisation mathématique du modèle d'attaques proposé

Au chapitre 2, nous avons commencé à formaliser mathématiquement le concept d'attaque. Nous avons considéré une attaque comme une séquence d'actions élémentaires dont certaines composantes, les attaques élémentaires, sont malveillantes. Il serait intéressant de compléter ce travail de formalisation (et d'inclure, par exemple, la notion d'entrelacement d'actions élémentaires, de pré- et post-condition, etc.) puis de l'étendre à notre modèle d'attaques. Cela nous permettrait alors de construire puis d'identifier, de façon non-ambiguë mais surtout systématique, l'ensemble des attaques possibles pour un système informatique quelconque. Les concepteurs de systèmes pourraient également intégrer ce modèle formel au cycle de développement de leurs systèmes et le mettre en œuvre dès l'étape de spécification afin de détecter au plus tôt les éventuelles séquences d'attaque et de les corriger efficacement.

2.2 Étude des attaques par entrées-sorties pour d'autres systèmes

Afin de réduire les coûts de développement de systèmes informatiques, les industriels utilisent de plus en plus de composants sur étagère, même pour des applications plus ou moins critiques. Dans le contexte actuel de forte mondialisation, les composants logiciels et matériels qui constituent ces systèmes sont rarement produits par une seule et même entité : les composants matériels peuvent être assemblés dans un pays, et les logiciels développés par un autre

pays. Bien que théoriquement difficile à mettre en œuvre, il est légitime de penser que des attaquants, éventuellement aidés par des organisations étatiques, réussissent à insérer des portes dérobées dans des éléments du système, en particulier les composants matériels [Bockel 12].

Le cas des systèmes avioniques, de plus en plus ouverts vers l'environnement de l'avion, est particulièrement préoccupant [Laarouchi 09]. Étant donné le rôle de plus en plus prépondérant joué par ces systèmes, il serait intéressant d'étudier leur robustesse vis-à-vis d'attaques. En effet, une défaillance de ces systèmes pourraient avoir des effets catastrophiques qui conduiraient inévitablement à des pertes humaines considérables. Certains de ces systèmes reposent sur une architecture matérielle similaire à l'architecture PC que nous avons étudiée dans ce manuscrit. Par ailleurs, ceux-ci implémentent le protocole de bus PCI Express pour leurs bus d'entrées-sorties. Nous pourrions alors, sans trop de peine, utiliser notre *fuzzer* (cf. chapitre 4) sur ces systèmes afin d'évaluer l'impact d'attaques par entrées-sorties et proposer, par là-même, des mécanismes de tolérance aux fautes adaptés.

2.3 Détection et prévention des attaques par entrées-sorties

Jusqu'à présent, nous avons discuté uniquement des utilisations offensives de notre contrôleur d'entrées-sorties *IronHide* (cf. chapitre 4). Étant donné que le comportement de notre prototype peut entièrement être modifié simplement par un *firmware*, nous avons également envisagé de l'utiliser comme un composant de sécurité. En particulier, il pourrait constituer un outil de détection (et éventuellement de prévention) d'intrusions spécifiques aux entrées-sorties. Ceci est particulièrement important vis-à-vis des attaques par entrées-sorties, puisqu'elles sont très difficilement détectables par logiciel, dans la mesure où elles ne font généralement intervenir ni le processeur, ni la mémoire centrale.

En raison de la complexité et de la nature hétéroclite des attaques par entrées-sorties, l'approche de détection par analyse comportementale semble être la plus appropriée. Hormis les difficultés techniques pour la mise en œuvre d'un tel outil, un des défis scientifiques majeurs à relever consistera à établir une métrique qui permettrait, d'une part, d'élaborer un modèle de comportement stable des contrôleurs d'entrées-sorties et qui, d'autre part, distinguerait de manière efficace les comportements déviants (correspondant éventuellement à des attaques) des comportements légitimes. À cet égard, nous pourrions fortement nous inspirer des travaux issus du domaine de la détection d'intrusions dans les réseaux, dans lequel cette problématique est récurrente.

Finalement, une fois l'attaque détectée, se pose le problème de savoir comment riposter. Il pourrait alors être envisagé d'imposer une politique de sécurité préventive et de bloquer systématiquement tout comportement jugé anormal. La modification de l'ensemble des pilotes de contrôleurs d'entrées-sorties et de périphériques serait alors inévitable, afin de les rendre tolérants aux fautes qui peuvent y être introduites par une telle politique de sécurité. Une autre approche, moins drastique, consisterait simplement à informer le système d'exploitation qui prendrait alors les mesures nécessaires contre le contrôleur supposé malveillant, comme le désactiver complètement par exemple.

Annexe A

Preuve de concept d'attaque par partage d'identifiants PCI Express

Cette section présente une preuve de concept pour la vulnérabilité détaillée en section 3.2. Nous commençons tout d'abord par présenter le scénario d'attaque ainsi que la plate-forme expérimentale que nous avons considérés. 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.

A.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*.

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 A.1 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 3.2. 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 une 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 [Dornseif 04, Becher *et al.* 05] 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.

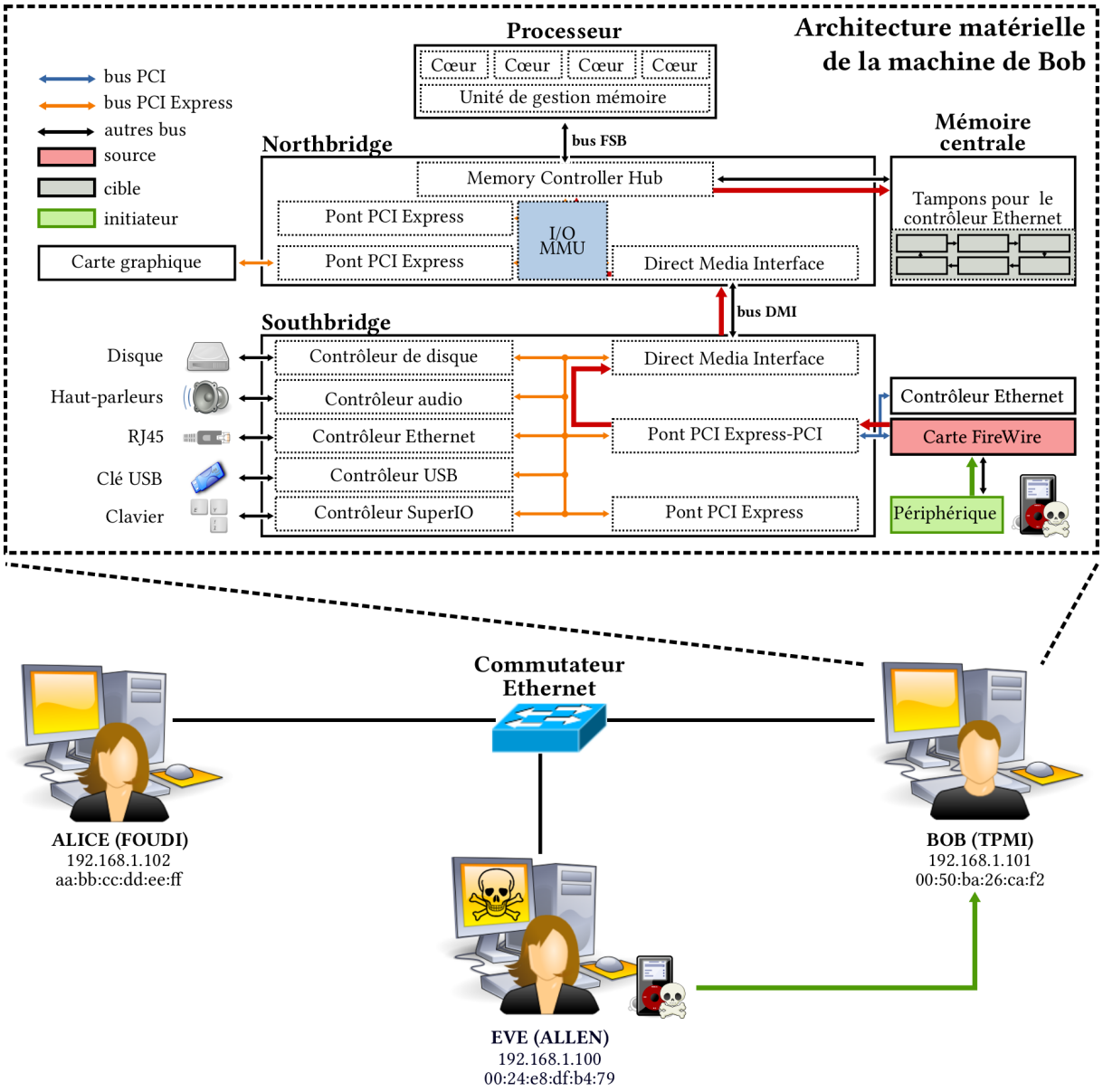

FIGURE A.1 – Plate-forme expérimentale considérée

A.2 Injection des données dans une carte Ethernet

Cette partie s'intéresse au fonctionnement de la carte Ethernet VIA Rhine VT6102 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.

A.2.0.1 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 mémoire tampon est libérée dès que le système d'exploitation l'a traitée. En mémoire centrale, la zone des mémoires tampons de réception est représentée par une liste circulaire de descripteurs de trame (figure A.2). 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` chaîne le descripteur de trame courant à son suivant. Notons que l'allocation de la liste de descripteurs de trame ainsi que des mémoires tampons associées est effectuée par le noyau du système d'exploitation.

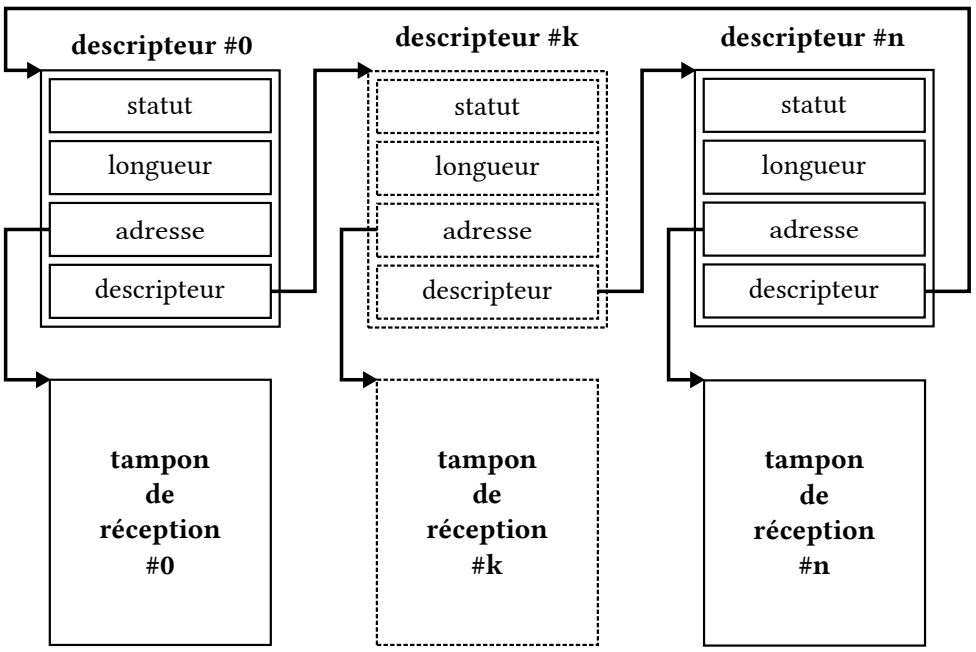

FIGURE A.2 – Le tampon de réception dans la carte Ethernet VIA Rhine VT6102

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 trame afin de décrire de manière cohérente la mémoire tampon où est copiée la trame reçue. Afin que le contrôleur Ethernet 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 la zone des tampons, 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.

A.2.0.2 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 centrale, directement dans le tampon de réception de sa carte réseau. Nous allons donc imiter 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 centrale utilisée par la carte Ethernet. (2) Il modifie ensuite les descripteurs de trame 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.

A.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 (*Address Resolution Protocol*) 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 [35]. Dans le cadre 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 :

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

1. *Ève* commence par forger les trames Ethernet qu'elle va injecter dans la machine de *Bob*. Elle peut utiliser par exemple l'outil logiciel SCAPY. 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 trame. 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 trame 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 en attente de traitement, dont les 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, pour laquelle une vidéo est disponible [Lone Sang *et al.* 10], 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 contrôleurs d'entrées-sorties partager une même zone mémoire.

Annexe B

Preuve de concept d'attaque par accès pair-à-pair 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 (par exemple, une carte FireWire ou une carte réseau) pour manipuler la mémoire vidéo d'une carte graphique. Nous rappelons brièvement pour commencer l'architecture d'une carte graphique.

B.1 Architecture d'une carte graphique

La figure B.1 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.

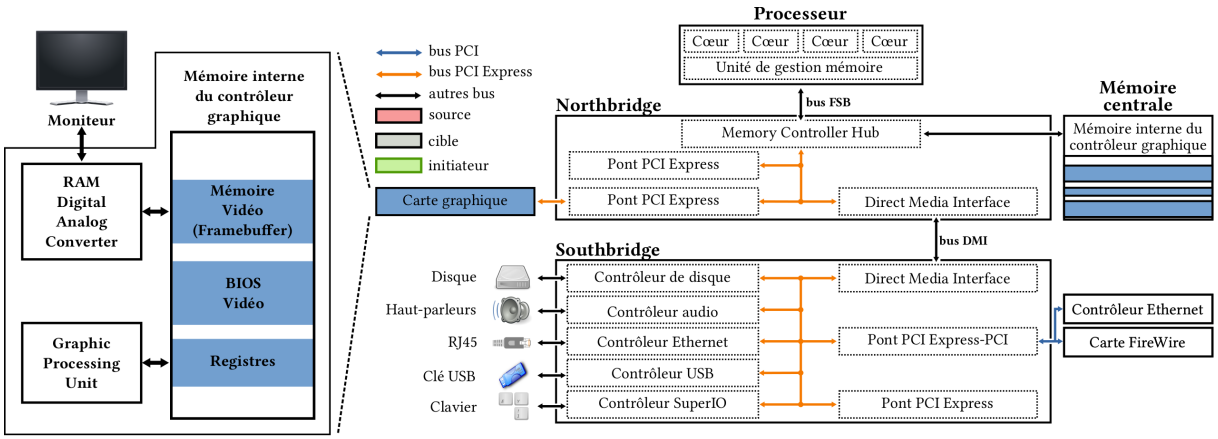

FIGURE B.1 – Architecture simplifiée d'une carte graphique

Une carte graphique est constituée de divers composants qui collaborent entre eux pour produire une image que l'on affiche sur un écran. 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 écran.
— 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 (par exemple, un 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, par exemple compatibles avec la norme VGA des écrans graphiques.

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 l'écran. 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 sections détaillent le scénario d'attaque que nous avons considéré et discutent des résultats obtenus.

B.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 [Duflot *et al.* 10] présente dans la machine de *Bob* et y placer un *rootkit* comme présenté dans [Triulzi 08a, Triulzi 10b, Delugré 10]. 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 (par exemple, un iPod modifié ou un ordinateur portable). Plusieurs articles récents ont montré que ceci est bien réalisable en pratique supposition [Dornseif 04, Becher *et al.* 05, Boileau 06]. 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. Elle exploite ensuite 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 cet exemple, nous utilisons la lecture de la mémoire vidéo de *Bob* pour afficher à distance le contenu de l'écran de *Bob* sur l'écran 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 B.2 représente ce scénario d'attaque.

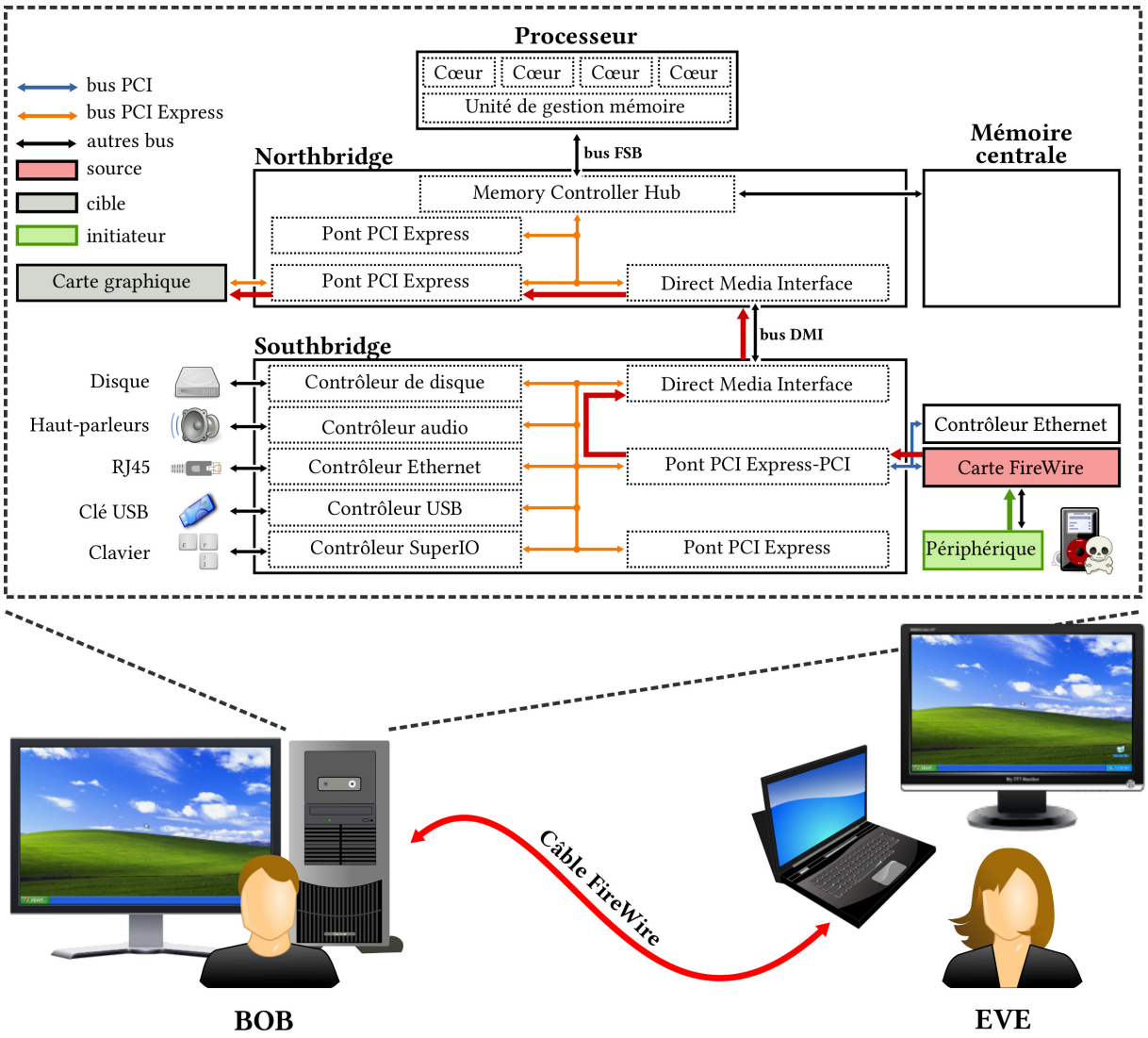

FIGURE B.2 – Scénario d'attaque

B.3 Résultats obtenus

Nous avons développé cette attaque en utilisant comme machine cible (*Bob*) une machine de type *desktop* disposant d'une carte-mère Intel DX58SO, laquelle embarque un *chipset* Intel x58 (plus précisément, IOH x58 / ICH10). Elle dispose aussi d'une carte graphique nVidia GeForce 9800 GT et d'un contrôleur FireWire de type PCI. C'est ce contrôleur FireWire qui, lors de l'attaque, sera piloté par la machine de l'attaquant (*Ève*), un ordinateur portable connecté sur le port FireWire. 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 (par exemple, un couple identifiant-mot de passe), mais insuffisant pour récupérer un flux en temps réel (par exemple, la 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. 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. Nous aurions pu obtenir de meilleures performances 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 améliorer la fluidité de la lecture 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 utiliser un matériel représentatif des systèmes actuels.

— 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.

Annexe C

Plusieurs exemples d'expérimentations avec le contrôleur IronHide

Cette annexe détaille les premières expérimentations que nous avons effectuées avec notre contrôleur IronHide une fois que celui-ci a été mis au point. Comme nos travaux visent à étudier les attaques depuis les entrées-sorties, et que l'interface fournie par la majorité des contrôleurs sur étagère est généralement restreinte aux requêtes d'accès à l'espace d'adressage principal de la mémoire (par exemple, pour le mécanisme d'accès direct à la mémoire), nous avons cherché à présenter dans cette annexe des expérimentations difficiles, voire impossibles à mettre en œuvre depuis des contrôleurs conventionnels. Ainsi, les prochaines sections détaillent ces expérimentations atypiques ainsi que les résultats associés. Afin de mieux les cerner, nous commençons par décrire la plate-forme d'expérimentation.

C.1 Description de la plate-forme d'expérimentation

La figure C.1 présente la plate-forme que nous avons utilisée pour nos expérimentations. Elle se compose principalement de deux machines qui jouent respectivement le rôle d'une *cible* et d'un *moniteur*. La machine *cible* représente la machine pour laquelle nous souhaitons étudier le comportement du *chipset* en réponse à des requêtes d'entrées-sorties diverses. Ces requêtes d'entrées-sorties sont initiées par notre contrôleur qui est relié à l'un des connecteurs d'extension PCI Express ou Express Card disponibles sur la carte-mère de la machine étudiée. Nous l'avons programmé pour servir de serveur mandataire (*proxy*) PCI Express pour la machine *moniteur* : il émet tous les paquets reçus depuis l'interface Ethernet en provenance de la machine *moniteur* vers les bus PCI Express de la machine *cible* et renvoie vers la machine *moniteur* les réponses éventuelles. Pour cela, le logiciel embarqué dans le contrôleur implémente une pile TCP/IP et contient un serveur TCP qui relaie les paquets entre la machine *moniteur* et les bus d'entrées-sorties. Le contrôleur est configuré avec une adresse IP statique (`192.168.1.10`) et le serveur TCP embarqué attend de nouvelles connexions sur le port `65535`. Il est à noter que nous aurions pu gagner en performance en choisissant l'utilisation d'un serveur UDP plutôt qu'un serveur TCP. Nous avons préféré utiliser le protocole TCP, de façon à fiabiliser les échanges et ainsi éviter les problèmes liés aux pertes de paquets réseau par congestion du contrôleur ou de la machine *moniteur* lors de nos expérimentations. Afin d'éviter toute ambiguïté, nous précisons que les adresses réseau que nous avons utilisées sont propres au contrôleur et ne sont pas liées aux éventuelles cartes réseau de la machine *cible*. Les trames reçues par le contrôleur sont directement traitées par sa pile TCP/IP interne et ne sont pas vues par le sys-

tème d'exploitation de la machine *cible*. En plus d'une interface Ethernet, nous avons ajouté au contrôleur une interface série RS-232 sur laquelle les activités du contrôleur sont régulièrement reportées.

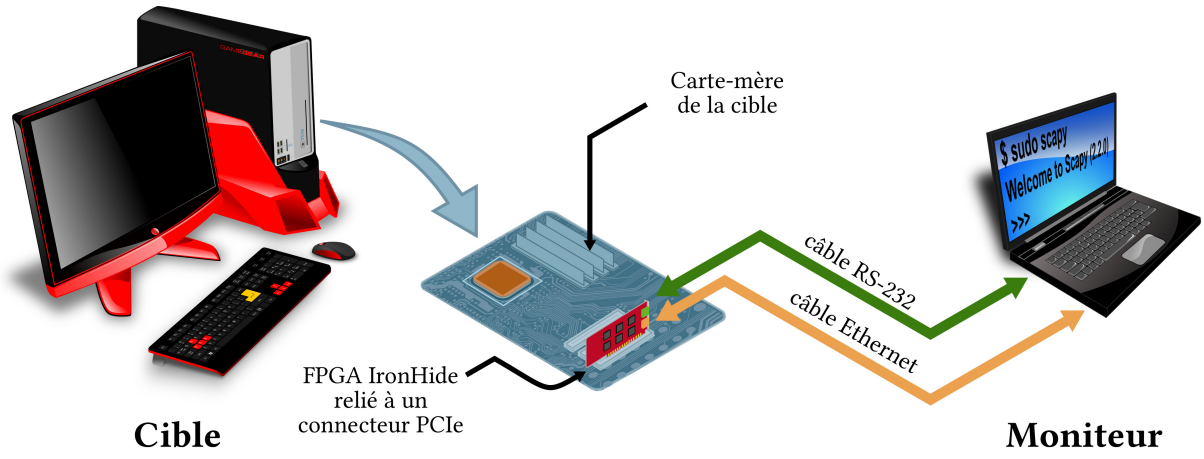

FIGURE C.1 – Plate-forme d'expérimentation

La machine *moniteur* est celle qui va définir les transactions PCI Express à émettre sur la machine *cible*. Pour construire et disséquer les *Transaction Layer Packets* (TLP) PCI Express, nous utilisons l'outil réseau Scapy. Comme ce dernier n'implémente pas nativement le protocole PCI Express, nous l'avons étendu pour intégrer ce nouveau protocole [36]. Scapy est également utilisé pour encapsuler les TLP PCI Express forgés dans un paquet TCP et se charge de les envoyer au contrôleur connecté à la machine *cible*.

C.2 Résultats expérimentaux

Dans cette section, nous détaillons quelques expérimentations que nous avons menées avec notre contrôleur. Nous pouvons les classer parmi les attaques hybrides qui détournent des fonctionnalités légitimes du matériel. Elles illustrent les possibilités d'attaques par entrées-sorties qui s'offrent à un attaquant avec un tel contrôleur.

C.2.1 Écoute sur les bus d'entrées-sorties

La première expérimentation que nous avons mise en place avec notre contrôleur consiste à effectuer de l'écoute passive sur les bus d'entrées-sorties. Nous souhaitons identifier, par cette expérience, les différentes interactions qui peuvent exister entre un contrôleur quelconque et les autres composants. Avant d'analyser les résultats obtenus, nous précisons que les bus PCI Express sont organisés de manière hiérarchique, et que les différents contrôleurs PCI Express sont interconnectés par des liaisons point-à-point. La notion de bus d'entrées-sorties est alors purement logique, et les contrôleurs en amont dans la hiérarchie (généralement des ponts PCI Express) se chargent de commuter les paquets entre différentes liaisons point-à-point et donnent ainsi l'illusion d'être sur un bus physique. Par conséquent, nous recevons, sur l'inter-

[36] Nous pensons que notre extension peut éventuellement être utilisée dans un autre contexte. Aussi, nous prévoyons de diffuser les codes-sources associés et une première version sera disponible très prochainement.

face PCI Express de notre contrôleur, uniquement les transactions qui lui sont destinées et les éventuelles transactions diffusées à tous les contrôleurs.

La figure C.2 présente notre contrôleur tel qu'il est vu par le système d'exploitation dans la machine *cible*. Pour nos expérimentations, nous avons gardé les identifiants par défaut de notre carte de développement (Pico E-17 assemblée par Pico Computing). Notez que notre contrôleur ne dispose pas de mémoire interne projetée dans l'espace d'adressage mémoire principale ni dans l'espace *Port I/O*. Bien que nous n'ayons développé ni installé de pilote pour cette carte de développement sur la machine *cible*, remarquez que celle-ci est malgré tout reconnue et activée par défaut par le système d'exploitation.

```
user@cible$ lspci -v
[...]
0e:00.0 Memory controller: Pico Computing Device 0e17 (rev 01)
    Subsystem: Device 4658:0000
    Flags: bus master, fast devsel, latency 0, IRQ 10
    Expansion ROM at f1e00000 [disabled] [size=1M]
    Capabilities: <access denied>
```

FIGURE C.2 – Détection de notre contrôleur par la machine *cible*

La figure C.3 présente un extrait des commandes exécutées sur la machine *moniteur* et les résultats qui lui ont été renvoyés. Puisque nous recevons un nombre important de paquets du contrôleur, nous avons uniquement gardé, sur cette figure, deux exemples de transactions que nous avons reçues.

```
user@moniteur$ sudo scapy
Welcome to Scapy (2.2.0)
>>> pkts = sniff(count=0, filter='tcp and src port 65535',
...              prn=lambda p:TLP(str(p[TCP].payload)).show())
[...]
###[ MemoryRequest ]###        ###[ MessageRequest ]###
  fmt       = 3DW HDR no data    fmt       = 4DW HDR with data
  type      = MRd32              type      = MsgD
  rsvd0     = 0x0L               rsvd0     = 0x0L
  tc        = 0x0L               tc        = 0x0L
  rsvd1     = 0x0L               rsvd1     = 0x0L
  attr      = 0x0L               attr      = 0x0L
  rsvd2     = 0x0L               rsvd2     = 0x0L
  th        = 0x0L               th        = 0x0L
  td        = 0x0L               td        = 0x0L
  ep        = 0x0L               ep        = 0x0L
  attr2     = 0x0L               attr2     = 0x0L
  at        = 0x0L               at        = 0x0L
  length    = 1L                 length    = 1L
  reqID     = 0:0.0              reqID     = 0:0.0
  tag       = 0x18               tag       = 0x0
  ldbe      = 0x0L               msgcode   = 0x7FL
  fdbe      = 0xfL               bfdID     = 0:0.0
  address32 = 0xf1e00000L        vendorID  = 0x8086L
  rsvd3     = 0x0L               specific  = 0x00000080L
                                 data      = ')\x00\x00\x00'
```

FIGURE C.3 – Quelques exemples de requêtes reçues par notre contrôleur

Nous remarquons que notre contrôleur reçoit deux types de transactions : des transactions

de type *memory* (à gauche) et des transactions de type *message* (à droite). Nous rappelons que les transactions de type *memory* sont utilisées pour des accès aux mémoires internes des contrôleurs qui sont projetées dans l'espace d'adressage principal de la mémoire. Cet accès est effectué uniquement au démarrage de la machine *cible* et correspond en réalité au chargement des ROM d'extensions par le BIOS, lesquelles contiennent des routines d'initialisation des contrôleurs fournies par les constructeurs, et exécutées par le BIOS. Le champ `reqID` précise que cet accès est initié par le processeur, au travers du contrôleur mémoire situé dans le *northbridge* dont l'identifiant de bus est `00:00.0` (c'est-à-dire, *bus* 00, *device* 00 et *function* 0).

Nous recevons également régulièrement des transactions de type *message*. Ce type de transactions véhiculent différents types d'informations entre les contrôleurs PCI Express. Elles sont utilisées notamment pour signaler des interruptions, des erreurs ou pour véhiculer des messages de gestion d'énergie. Il s'agit ici d'une requête propre aux *chipsets* Intel. En effet, le champ `msgcode` positionné à la valeur `0x7f` désigne un message spécifique aux constructeurs, et nous reconnaissons le constructeur Intel par le `vendorID` positionné à `0x8086`. Malheureusement, ce type de transaction *message* n'est pas documenté dans les spécifications du *chipset* que nous avons étudié. Nous ne savons pas, pour l'instant, à quoi correspond cette transaction ni les données qu'elle contient.

C.2.2 Non-respect de la configuration matérielle

Pour qu'un contrôleur sur étagère puisse effectuer des accès de type DMA, il est nécessaire que le pilote qui lui est associé positionne le bit *Bus Master Enable* (BME) dans la configuration du contrôleur. Le respect (ou le non-respect) de cette configuration n'engage malheureusement que le contrôleur. Nous avons voulu déterminer, par cette expérimentation, si un mécanisme dans le *chipset*, en l'absence d'une I/O MMU, est en mesure de détecter et de parer tout accès frauduleux initié depuis un contrôleur préalablement configuré par le système d'exploitation pour ne pas pouvoir initier, de lui même, des accès à la mémoire centrale. La figure C.4 présente les commandes exécutées sur la machine *cible* pour désactiver la fonctionnalité de *bus-mastering* dans le contrôleur.

```
user@cible$ lspci -v
[...]
0e:00.0 Memory controller: Pico Computing Device 0e17 (rev 01)
    Subsystem: Device 4658:0000
    Flags: bus master, fast devsel, latency 0, IRQ 10
    Expansion ROM at f1e00000 [disabled] [size=1M]
    Capabilities: <access denied>
user@cible$ sudo setpci -s 0e:00.0 COMMAND.W=0
user@cible$ lspci -v | grep -B 3 Flags
[...]
0e:00.0 Memory controller: Pico Computing Device 0e17 (rev 01)
    Subsystem: Device 4658:0000
    Flags: fast devsel, latency 0, IRQ 10
```

FIGURE C.4 – Désactivation du bit *Bus Master Enable* (BME) sur un contrôleur

Une fois la fonctionnalité désactivée dans le contrôleur, nous avons programmé des accès de type DMA depuis la machine *moniteur*. La figure C.5 présente le type de requêtes que nous avons initié depuis le contrôleur. À notre grand regret, les requêtes d'accès vers l'espace d'adressage principal de la mémoire que nous avons générées ont toutes abouti et il nous a été possible d'effectuer des accès DMA bien que le système d'exploitation ait délibérément

configuré le contrôleur pour empêcher ce type d'accès. L'utilisation d'une I/O MMU est alors indispensable pour bloquer (au moins partiellement) les éventuelles attaques initiées depuis les contrôleurs. Nous rappelons que les I/O MMU désignent des unités de gestion de la mémoire physique dédiées aux contrôleurs d'entrées-sorties. Elles ont été initialement conçues pour virtualiser l'espace d'adressage principal de la mémoire pour les contrôleurs. Avec l'avènement de la virtualisation, elles ont été étendues afin d'assurer également des propriétés d'isolation. Il est aujourd'hui possible de configurer les I/O MMU pour associer chaque contrôleur à un ou plusieurs domaines distincts. Ces dernières s'assurent en particulier qu'un attaquant ne détourne pas un contrôleur associé à un domaine pour accéder frauduleusement (par un accès de type DMA) aux régions de mémoire associées à un autre domaine. Dans notre exemple, le contrôleur n'est associé à aucun domaine étant donné que le bit BME n'est plus positionné. L'I/O MMU (si elle est correctement configurée) bloque alors tous les accès DMA de ce contrôleur.

C.2.3 Usurpation d'identité sur les bus d'entrées-sorties

Une des techniques pour passer au travers du contrôle d'une I/O MMU consiste à usurper l'identité d'un autre contrôleur. En effet, les technologies I/O MMU se basent uniquement sur l'adresse de bus fournie dans le paquet PCI Express (champ `Requester ID`) pour identifier le contrôleur à l'origine de l'accès. Il est alors possible de modifier un contrôleur de façon à ce qu'il émette des paquets utilisant l'identifiant d'un autre contrôleur et d'obtenir en conséquence les mêmes droits d'accès à l'espace d'adressage principal de la mémoire que le contrôleur ciblé. Par cette expérimentation, nous avons voulu identifier les limites pratiques qu'un attaquant peut rencontrer avec un tel moyen d'attaque. La figure C.5 présente le type de requêtes que nous émettons sur les bus d'entrées-sorties grâce à notre contrôleur. Il s'agit d'un accès en écriture à l'espace d'adressage principal de la mémoire dans lequel nous usurpons l'identité du contrôleur mémoire situé dans le *northbridge*.

Dans cet exemple, nous avons volontairement effectué un accès invalide afin que l'I/O MMU le détecte et le rapporte au système d'exploitation. Puisque le contrôleur mémoire ne dispose pas de tables de configuration qui lui sont associées, l'I/O MMU considère cet accès comme frauduleux et le bloque. Nous nous servons de cet accès invalide pour vérifier que nous avons effectivement usurpé l'identité du contrôleur mémoire. La figure C.6 présente un extrait des journaux systèmes rapportant cet accès frauduleux à l'espace d'adressage principal de la mémoire. Nous remarquons que l'I/O MMU accuse à tort le contrôleur mémoire, dont l'identifiant de bus est `00:00.0`, comme étant à l'origine de l'accès frauduleux. Cet identifiant correspond bien à celui que nous avons usurpé avec notre contrôleur d'entrées-sorties.

Afin d'étudier l'impact d'une usurpation d'identité sur un bus d'entrées-sorties, nous avons connecté notre contrôleur à différents emplacements dans le *chipset*, puis nous avons généré des accès en lecture et en écriture vers l'espace d'adressage mémoire principal en utilisant l'identité d'autres contrôleurs. Nous avons observé qu'un attaquant qui utilise cette technique pour tromper le contrôle d'accès d'une I/O MMU reste tout de même restreint à un nombre limité d'actions malveillantes. Il peut, par exemple, écrire dans les régions de mémoire pour lesquels le contrôleur dont on usurpe l'identité dispose de permissions d'accès en écriture. Il peut également initier des accès en lecture aux régions de mémoire pour lesquels le contrôleur ciblé dispose de permissions d'accès en lecture. En revanche, il ne recevra pas le paquet *completion* en réponse à sa requête d'accès, celui-ci étant acheminé par le *chipset* au contrôleur dont on a usurpé l'identité. Cela s'explique par la manière dont les diverses transactions PCI Express sont routées par le *chipset*. En effet, les transactions de type *memory* sont routées par le *chipset* en fonction de l'adresse à laquelle on souhaite accéder. En revanche, les éventuelles réponses asso-

```
user@moniteur$ sudo scapy
Welcome to Scapy (2.2.0)
>>> ip = IP(dst='192.168.1.10')
[...]
>>> tlp = MWr32(fmt=0x2, type=0x0, reqID='0:0.0',
...             address32=0x0, data='\xde\xad\xbe\xef')
>>> tlp.show2()
###[ MemoryRequest ]###
  fmt       = 3DW HDR with data
  type      = MWr32
  rsvd0     = 0x0L
  tc        = 0x0L
  rsvd1     = 0x0L
  attr      = 0x0L
  rsvd2     = 0x0L
  th        = 0x0L
  td        = 0x0L
  ep        = 0x0L
  attr2     = 0x0L
  at        = 0x0L
  length    = 1L
  reqID     = 0:0.0
  tag       = 0x2
  ldbe      = 0x0L
  fdbe      = 0x1L
  address32 = 0x0L
  rsvd3     = 0x0L
  data      = '\xde\xad\xbe\xef'
>>> send(ip/TCP(dport=65535, sport=synack.dport,
...             seq=synack.ack, ack=synack.seq+1)/tlp)
.
Sent 1 packets.
```

FIGURE C.5 – Transaction `MemoryWriteRequest` avec un identifiant quelconque

ciées (c'est-à-dire, les paquets *completion*) sont routées en fonction de l'identifiant du contrôleur qui a initié la requête. Comme nous avons usurpé l'identité d'un autre contrôleur, la réponse va naturellement être routée vers le contrôleur pour lequel nous essayons de nous faire passer. Précisons que cette attaque aurait pu être parée si les extensions matérielles *Acess Control Services* (ACS) avaient été activées et correctement configurées. Nous rappelons que les ACS désignent une technologie de contrôle d'accès implémentée dans certains contrôleurs PCI Express du *chipset* (son implémentation étant optionnelle). Il est possible, par exemple, d'activer le service *ACS Source Validation* dans les différents contrôleurs PCI Express du *chipset* afin de leur faire vérifier systématiquement que l'identité utilisée par un contrôleur correspond bien à celle qui lui a été attribuée, empêchant ainsi toute possibilité d'usurpation d'identité.

C.2.4 Enregistreur de frappe

Dans cette expérimentation, nous avons profité des fonctionnalités de notre contrôleur pour effectuer des accès vers l'espace *Port I/O*. Nous rappelons qu'il est difficile de mettre en œuvre une telle expérimentation depuis des contrôleurs d'entrées-sorties sur étagère, ces derniers étant généralement restreints aux requêtes d'accès à l'espace d'adressage mémoire principal.

Nous avons observé des résultats différents en fonction des *chipsets* sur lesquels nous avons mené nos expérimentations. Par exemple, sur les *chipsets* supportant les accès *peer-to-peer*, nous avons réussi à faire une image de l'espace *Port I/O* (cf. figure C.7). En revanche, sur les autres

```
user@cible$ dmesg | grep DMAR
[...]
DMAR:[DMA Write] Request device [00:00.0] fault addr 000000000
DMAR:[fault reason 05] PTE Write access is not set
```

FIGURE C.6 – Exemple d'accès frauduleux détecté par l'I/O MMU

chipsets, le *northbridge* ou le *southbridge* répondent explicitement que la transaction n'est pas supportée. En effet, ces derniers renvoient directement un paquet *completion* avec le statut de transaction positionné à *Unsupported Request*.

```
user@moniteur $ od -t x1 dump-pio-x58-via-controleur
00000000 8c 0a 0f af 24 92 04 ff 00 ff ff ff ff ff ff 00
*
00000020 01 ff ff ff 01 ff ff ff 01 ff ff ff 01 ff aa 00
00000030 01 ff ff ff 01 ff ff ff 01 ff ff ff 01 ff ff ff
00000040 98 10 ac ff ff ff ff ff ff ff ff ff ff ff ff ff
00000050 d0 06 03 ff ff ff ff ff ff ff ff ff ff ff ff ff
00000060 fe 2c ff ff 74 ff ff ff ff ff ff ff ff ff ff ff
00000070 ff 02 7d 01 0b 02 7d 01 ff ff ff ff ff ff ff ff
00000080 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090 ff 01 00 00 ff ff ff 00 ff 00 00 00 ff ff ff 00
000000a0 0c ff ff ff 0c ff ff ff 0c ff ff ff 0c ff ff ff
000000b0 0c ff a0 7f 0c ff ff ff 0c ff ff ff 0c ff ff ff
000000c0 00 80 d6 69 84 00 ff bf 02 00 fe ef 06 00 fa b5
000000d0 00 00 ff ff ff ff ff ff ff ff ff ff ff ff 00 00
000000e0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
000000f0 00 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
[...]
```

FIGURE C.7 – Espace *Port I/O* vu de notre contrôleur connecté à un *chipset* Intel x58

Bien que l'espace *Port I/O* soit aujourd'hui peu usité, nous observons que des contrôleurs (par exemple, le contrôleur de clavier, les contrôleurs VGA, SATA, USB) continuent à y projeter certains de leurs registres. Le contrôleur de clavier projette, par exemple, des registres aux adresses `0x60` et `0x64` qui informent le système d'exploitation des événements du clavier ou de la souris. Nous avons exploité la possibilité de lire ces ports d'entrées-sorties depuis notre contrôleur sur le *chipset* Intel x58 (cf. figure C.7) pour implémenter un enregistreur de frappe au clavier. Actuellement, nous sommes capables d'enregistrer les frappes de claviers de type PS/2 et éventuellement des claviers de type USB lorsque l'option *USB Legacy Support* est activée dans le BIOS. Nous rappelons que cette option permet d'utiliser certains périphériques USB (par exemple, un clavier, une souris, un périphérique de stockage) au démarrage de la machine, bien avant que les contrôleurs USB ne soient initialisés par le système d'exploitation. Dans cette configuration, le BIOS (ou plus précisément la routine de traitement de la SMI) présente les claviers ou les souris USB au système d'exploitation comme étant des périphériques PS/2. Cette expérimentation exploite la possibilité d'initier des accès *peer-to-peer* sur certains *chipsets*. Cette attaque aurait également pu être parée par une configuration adéquate des ACS. En particulier, il aurait été possible de bloquer tout accès *peer-to-peer* en activant le service *ACS Egress Control*.

Ces expérimentations esquissent l'étendue des possibilités qui s'offrent à nous (et aux atta-

quants) avec un tel contrôleur. Dans les expérimentations présentées dans cette annexe, nous nous sommes principalement intéressés aux requêtes bien formées et qui ne dévient pas ou qui dévient peu par rapport au protocole PCI Express. L'utilisation d'IronHide pour évaluer la robustesse des *chipsets* vis-à-vis de requêtes invalides (et explicitement interdites) par le protocole PCI Express est discutée au chapitre 4. Précisons, pour terminer, qu'IronHide a été conçu avec la capacité d'« émuler » un contrôleur PCI Express quelconque. Il pourrait alors être personnalisé pour répondre à des besoins (d'attaques ou de défenses) bien spécifiques. En adaptant son logiciel embarqué, il est possible de l'utiliser, par exemple, pour évaluer la robustesse des pilotes de contrôleurs d'entrées-sorties ou de périphériques.

Bibliographie

[Abbott *et al.* 76] Robert P. Abbott, Janet S. Chin, James E. Donnelley, William L. Konigsford, Shigeru Tokubo, et Douglas A. Webb. Security Analysis and Enhancements of Computer Operating Systems. Rapport technique numéro NBSIR 76-1041, Institute for Computer Sciences and Technology, National Bureau of Standards, Washington DC, (WA, USA), avril 1976. http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA436876.

[Abrial *et al.* 91] Jean-Raymond Abrial, Matthew K. O. Lee, David Neilson, P. N. Scharbach, et Ib Holm Sørensen. The B-Method. *In* Søren Prehn et Hans Toetenel, éditeurs : *VDM Europe : Formal Developments Methods (VDM 91)*, volume 552 de *Lecture Notes in Computer Science*, pages 398–405, Noordwijkerhout (The Netherlands). Springer Berlin Heidelberg, 21-25 octobre 1991. ISBN 978-3-540-54868-3. http://dx.doi.org/10.1007/BFb0020001.

[Adelsbach *et al.* 03] André Adelsbach, Dominique Alessandri, Christian Cachin, Sadie Creese, Yves Deswarte, Klause Kursawe, Jean-Claude Laprie, David Powell, Brian Randell, James Riordan, Peter Ryan, William Simmonds, Rober Stroud, Paulo Veríssimo, Michael Waidner, et Andreas Wespi. MAFTIA, Malicious and Accidental-Fault Tolerance for Internet Applications : Conceptual Model and Architecture. David Powell et Robert Stroud, éditeurs. Rapport technique, 31 janvier 2003. Research Project IST-1999-11583, delivrable D21. http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/maftia/deliverables/D21.pdf.

[Agilent Technologies 12] Agilent Technologies. *Agilent U4301A PCI Express 3.0 Analyzer Module - Datasheet*. USA, 26 mars 2012. http://cp.literature.agilent.com/litweb/pdf/5990-5018EN.pdf.

[Aleph One 96] Aleph One. Smashing The Stack For Fun And Profit. *Phrack Magazine*, 49(14), 8 novembre 1996. http://www.phrack.org/issues.html?issue=49&id=14.

[Anderson 80] James P. Anderson. Computer security threat monitoring and surveillance. Rapport technique , Contrat numéro 79F296400, James P. Anderson Co., Washington (WA, USA), 26 février 1980. http://csrc.nist.gov/publications/history/ande80.pdf.

[Anderson et Kuhn 98] Ross J. Anderson et Markus G. Kuhn. Low Cost Attacks on Tamper Resistant Devices. *In* Bruce Christianson, Bruno Crispo, Mark Lomas, et Michael Roe, éditeurs : *Proceedings of the* 5th *International Workshop on Security Protocols*, volume 1361 de *Lecture Notes in Computer Science*, pages 125–136, Paris (France). Springer Berlin Heidelberg, 7-9 avril 1998. ISBN 978-3-540-64040-0. http://dx.doi.org/10.1007/BFb0028165.

[Aslam 95] Taimur Aslam. A Taxonomy Of Security Faults In The Unix Operating System. Mémoire de D.E.A., Department of Computer Sciences, Purdue University, West

Lafayette (IN, USA), août 1995. http://cwe.mitre.org/documents/sources/ATaxonomyofSecurityFaultsintheUNIXOperatingSystem%5BAslam95%5D.pdf.

[Aumaitre 08] Damien Aumaitre. Voyage au coeur de la mémoire. *In Actes du 6e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 08)*, pages 378–437, Rennes (France). Ecole Supérieure et d'Application des Transmissions (ESAT), 4-6 juin 2008. http://actes.sstic.org/SSTIC08/Voyage_Coeur_Memoire/.

[Aumaitre et Devine 10] Damien Aumaitre et Christophe Devine. Virtdbg : Un débogueur noyau utilisant la technologie de virtualisation matérielle VT-x. *In Actes du 8e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 10)*, pages 74–133, Rennes (France). 9-11 juin 2010. http://www.sstic.org/media/SSTIC2010/SSTIC-actes/virtdbg/.

[Aumüller *et al.* 03] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, et Jean-Pierre Seifert. Fault Attacks on RSA with CRT : Concrete Results and Practical Countermeasures. *In* Burton S. Kaliski, çetin K. Koç, et Christof Paar, éditeurs : *Proceedings of the 4*th *International Workshop on Cryptographic Hardware and Embedded Systems (CHES 02)*, volume 2523 de *Lecture Notes in Computer Science*, pages 260–275, Redwood City (CA, USA). Springer Berlin Heidelberg, 2003. ISBN 978-3-540-00409-7. http://dx.doi.org/10.1007/3-540-36400-5_20.

[Avižienis *et al.* 04] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, et Carl E. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1(1):11–33, IEEE Computer Society Press, Los Alamitos (CA, USA), janvier 2004. ISSN 1545-5971. http://dx.doi.org/10.1109/TDSC.2004.2.

[Bareil 06] Nicolas Bareil. Playing with `ptrace()` for fun and for profit. *In Actes du 4e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 06)*, pages 89–106, Rennes (France). Ecole Supérieure et d'Application des Transmissions (ESAT), 31 mai - 2 juin 2006. http://actes.sstic.org/SSTIC06/Playing_with_ptrace/.

[Becher *et al.* 05] Michael Becher, Maximillian Dornseif, et Christian N. Klein. FireWire - all your memory are belong to us. *In CanSecWest/core05*, Vancouver (Canada). 4-5 mai 2005. http://md.hudora.de/presentations/#firewire-cansecwest.

[Bechta Dugan *et al.* 90] J. Bechta Dugan, Salvatore J. Bavuso, et M.A. Boyd. Fault trees and sequence dependencies. *In Proceedings of the Annual Reliability and Maintainability Symposium*, pages 286–293, Los Angeles (CA, USA). 23-25 janvier 1990. http://dx.doi.org/10.1109/ARMS.1990.67971.

[Biham et Shamir 97] Eli Biham et Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. *In* Jr. Kaliski BurtonS., éditeur : *Proceedings of the 17*th *Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 97)*, volume 1294 de *Lecture Notes in Computer Science*, pages 513–525, Konstanz (Germany). Springer Berlin Heidelberg, 1997. ISBN 978-3-540-63384-6. http://dx.doi.org/10.1007/BFb0052259.

[Biondi 13] Philippe Biondi. Scapy. Page consultée en juillet 2013. http://www.secdev.org/projects/scapy/.

[Bisbey et Hollingworth 78] Richard Bisbey et Dennis Hollingworth. Protection Analysis : Final Report. Rapport technique numéro ISI/SR-78-13, Information Sciences Institute,

University of Southern California, Marina Del Rey (CA, USA), mai 1978. http://csrc.nist.gov/publications/history/bisb78.pdf.

[Bishop 95] Matt Bishop. A Taxonomy of UNIX System and Network Vulnerabilities. Rapport technique CSE-95-8, Department of Computer Science, University of California, Davis (CA, USA), mai 1995. http://nob.cs.ucdavis.edu/bishop/notes/1995-cse-8/1995-cse-8.pdf.

[Bjørner et Jones 78] Dines Bjørner et Cliff B. Jones. *The Vienna Development Method : The Meta-Language*, volume 61 de *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1978. ISBN 978-3-540-08766-3. http://dx.doi.org/10.1007/3-540-08766-4.

[Bockel 12] Jean-Marie Bockel. La cyberdéfense : un enjeu mondial, une priorité nationale. Rapport d'information numéro 681, Commission des affaires étrangères, de la défense et des forces armées, 18 juillet 2012. http://www.senat.fr/rap/r11-681/r11-6811.pdf.

[Boileau 06] Adam Boileau. Hit by a Bus : Physical Access Attacks with FireWire. *In RUXCON 2006*, Melbourne (Australia). 30 septembre - 1 octobre 2006. http://www.ruxcon.org.au/files/2006/firewire_attacks.pdf.

[Boneh *et al.* 97b] Dan Boneh, Richard A. DeMillo, et Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. *In* Walter Fumy, éditeur : *Proceedings of the 16*th *Annual International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT 97)*, volume 1233 de *Lecture Notes in Computer Science*, pages 37–51, Konstanz (Germany). Springer Berlin Heidelberg, 1997. ISBN 978-3-540-62975-7. http://dx.doi.org/10.1007/3-540-69053-0_4.

[Boneh *et al.* 01a] Dan Boneh, Richard A. DeMillo, et Richard J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology*, 14(2):101–119, Springer-Verlag, 2001. ISSN 0933-2790. http://dx.doi.org/10.1007/s001450010016.

[Bouissou et Bon 03] Marc Bouissou et Jean-Louis Bon. A new formalism that combines advantages of fault-trees and markov models : Boolean logic driven Markov processes. *Reliability Engineering & System Safety*, 82(2):149–163, Elsevier Editions with the European Safety and Reliability Association, and the Safety Engineering and Risk Analysis Division, novembre 2003. ISSN 0951-8320. http://dx.doi.org/10.1016/S0951-8320(03)00143-1.

[Brinkley et Schell 95] Donald L. Brinkley et Robert R. Schell. What is There to Worry About ? An Introduction to the Computer Security Problem. *In* Marshall D. Abrams, Sushil Jajodia, et Harold J. Podell, éditeurs : *Information Security : An Integrated Collection of Essays*, pages 11–39. IEEE Computer Society Press, Los Alamitos (CA, USA), 1995. ISBN 978-0-818-63662-2.

[Budruk *et al.* 03] Ravi Budruk, Don Anderson, et Ed Solari. *PCI Express System Architecture*. PC System Architecture Series, MindShare Inc. Addison-Wesley Developer's Press, Boston (MA, USA), septembre 2003. ISBN 978-0-321-15630-3.

[Carrier et Grand 04] Brian Carrier et Joe Grand. A Hardware-based Memory Acquisition Procedure for Digital Investigations. *Digital Investigation Journal*, 1(1):50–60, février 2004. ISSN 1742-2876. http://www.digital-evidence.org/papers/tribble-preprint.pdf.

[Cesare 99] Silvio Cesare. Kernel Function Hijacking. novembre 1999. http://www.ouah.org/kernel-hijack.txt.

[Cheheyl *et al.* 81] Harris M. Cheheyl, Morrie Gasser, George A. Huff, et Jonathan K. Millen. Verifying Security. *ACM Computing Surveys (CSUR)*, 13(3):279–339, ACM Press, New York (NY, USA), septembre 1981. http://doi.acm.org/10.1145/356850.356853.

[Chifflier *et al.* 11] Pierre Chifflier, Loïc Duflot, Olivier Levillain, Fernand Lone Sang, Arnauld Michelizza, Benjamin Morin, et Yves-Alexis Perez. Sécurité et architecture PC : l'impossible confiance ? *Multi-System & Internet Security Cookbook (MISC)*, 58:18–47, Les Éditions Diamond, Sélestat (France), novembre/décembre 2011.

[Clarke et Emerson 08] Edmund M. Clarke et E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. *In* Orna Grumberg et Helmut Veith, éditeurs : *25 Years of Model Checking*, volume 5000 de *Lecture Notes in Computer Science*, pages 196–215. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-69849-4. http://dl.acm.org/citation.cfm?id=648063.747438.

[Cohen 86] Fred Cohen. *Computer Viruses*. Thèse de doctorat, Faculty of the grade school, University of Southern California, Los Angeles (CA, USA), janvier 1986. http://www.all.net/books/Dissertation.pdf.

[Collins 97d] Robert R. Collins. The Intel Pentium F00F Bug - Description and Workarounds. décembre 1997. http://www.rcollins.org/Errata/Dec97/F00FBug.html.

[Collins 12a] Robert R. Collins. Intel Bugs - The Errata Series. Page consultée en 2012. http://www.rcollins.org/Errata/ErrataSeries.html.

[Collins 12b] Robert R. Collins. Intel Secrets, Bugs and Undocumented Opcodes. Page consultée en 2012. http://www.rcollins.org/secrets/IntelSecrets.html.

[Collins 12c] Robert R. Collins. Undocumented OpCodes : SALC. Page consultée en 2012. http://www.rcollins.org/secrets/opcodes/SALC.html.

[Common Criteria 12] Common Criteria. Common Criteria for Information Technology Security Evaluation - Part 1 : Introduction and general model. Version 3.1, révision 4, numéro CCMB-2012-09-001, septembre 2012. http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R4.pdf.

[Corbató 91] Fernando J. Corbató. On Building Systems That Will Fail. *In ACM Turing Award Lecture*. 5 mars 1991. http://larch-www.lcs.mit.edu:8001/~corbato/turing91/.

[Cormen *et al.* 09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, et Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge (MA, USA), 3ᵉédition, 2009. ISBN 978-0-262-03384-8.

[Cousot 01] Patrick Cousot. Abstract Interpretation Based Formal Methods and Future Challenges. *In* Reinhard Wilhelm, éditeur : *Informatics - 10 Years Back. 10 Years Ahead.*, volume 2000 de *Lecture Notes in Computer Science*, pages 138–156. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-41635-7. http://dx.doi.org/10.1007/3-540-44577-3_10.

[Cousot et Cousot 77] Patrick Cousot et Radhia Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *In Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium*

on Principles of Programming Languages, pages 238–252, Los Angeles (CA, USA). ACM Press (NY, USA), 1977.

[crazylord 02] crazylord. Playing with Windows `/dev/(k)mem`. *Phrack Magazine*, 59(16), 28 juillet 2002. http://www.phrack.org/issues.html?id=16&issue=59.

[Creed 99] Creed. Knark - Kernel Based Linux Rootkit. 24 décembre 1999. http://biblio.l0t3k.net/magazine/en/b4b0/0009/b4b0-09.txt.

[Crenshaw 10] Adrian Crenshaw. Programmable HID USB Keyboard/Mouse Dongle for Pen-testing. *In DEF CON 18*, Las Vegas (NV, USA). 29 juillet - 1 août 2010. https://www.defcon.org/images/defcon-18/dc-18-presentations/Crenshaw/DEFCON-18-Crenshaw-PHID-USB-Device.pdf.

[Crouzet *et al.* 06] Yves Crouzet, Helene Waeselynck, Benjamin Lussier, et David Powell. The SESAME Experience : from Assembly Languages to Declarative Models. *In Proceedings of the* 2nd *Workshop on Mutation Analysis (MUTATION 06 - ISSRE Workshops 06)*, Raleigh (NC, USA). IEEE Computer Society, Los Alamitos (CA, USA), 7-10 novembre 2006. ISBN 978-0-769-52897-7. http://dx.doi.org/10.1109/MUTATION.2006.14.

[Davis 88] Alan M. Davis. A Comparison of Techniques for the Specification of External System Behavior. *Communications of the ACM*, 31(9):1098–1115, ACM Press, New York (NY, USA), septembre 1988. ISSN 0001-0782.

[Delugré 10] 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.

[Deransart *et al.* 96] Pierre Deransart, Laurent Cervoni, et AbdelAli Ed-Dbali. *Prolog : The Standard – Reference Manual*. Springer Berlin Heidelberg, 1996. ISBN 978-3-540-59304-1. http://dx.doi.org/10.1007/978-3-642-61411-8.

[Desautels 11] Adriel Desautels. Netragard's Hacker Interface Device (HID). Netragard, Inc., Boston (MA, USA), 24 juin 2011. http://pentest.netragard.com/2011/06/24/netragards-hacker-interface-device-hid/.

[Deswarte 03] Yves Deswarte. La sécurité des systèmes d'information. *In* Yves Deswarte et Ludovic Mé, éditeurs : *Sécurité des réseaux et systèmes répartis*, Traité IC2, série Réseaux et Télécoms, chapitre 1, pages 15–65. Hermès Science, octobre 2003. ISBN-10 2-7462-0770-2.

[Deswarte et Gambs 12] Yves Deswarte et Sébastien Gambs. Cyber-attaques et cyber-défenses : problématique et évolution. *Revue de l'Électricité et de l'Électronique (REE)*, (2):23–35, juin 2012. http://hal.inria.fr/hal-00736950.

[Devine et Vissian 09] Christophe Devine et Guillaume Vissian. Compromission physique par le bus PCI. *In Actes du 7e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 09)*, pages 169–193. 3-5 juin 2009. http://actes.sstic.org/SSTIC09/Compromission_physique_par_le_bus_PCI/.

[Diaz 82] Michel Diaz. Modelling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models. *In Proceedings of the IFIP WG6.1 Second International Workshop on Protocol Specification, Testing and Verification*, pages 465–510, Amsterdam (The Netherlands). North-Holland Publishing Co., 1982. ISBN 0-444-86481-4.

[Dijkstra 76] Edsger W. Dijkstra. *A Discipline of Programming*. Series in Automatic Computation. Prentice Hall PTR, Upper Saddle River (NJ, USA), 1976. ISBN 978-0-132-15871-8.

[Dornseif 04] Maximillian Dornseif. 0wned by an iPod - hacking by Firewire. *In PacSec/core04*, Tokyo (Japan). 11-12 novembre 2004. http://md.hudora.de/presentations/#firewire-pacsec.

[Dowd *et al.* 06] Mark Dowd, John McDonald, et Justin Schuh. *The Art of Software Security Assessment : Identifying And Preventing Software Vulnerabilities*. Addison-Wesley Professional, Boston (MA, USA), 20 novembre 2006. ISBN 978-0-321-44442-4.

[Duflot 07] Loïc Duflot. *Contribution à la sécurité des systèmes d'exploitation et des microprocesseurs*. Thèse de doctorat, Université de Paris XI, Paris (France), 18 octobre 2007. http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lti/these-duflot.pdf.

[Duflot et Levillain 09] Loic Duflot et Olivier Levillain. ACPI et routine de traitement de la SMI. *In Actes du 7ème Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 09)*, pages 132–168, Rennes (France). 3-5 juin 2009. http://actes.sstic.org/SSTIC09/ACPI_et_routine_de_traitement_de_la_SMI/.

[Duflot *et al.* 11] Loïc Duflot, Yves-Alexis Perez, et Benjamin Morin. Run-time firmware integrity verification : what if you can't trust your network card ? *In CanSecWest/core11*, Vancouver (Canada). 9-11 mars 2011. http://www.ssi.gouv.fr/IMG/pdf/Duflot-Perez_runtime-firmware-integrity-verification.pdf.

[Duflot *et al.* 10] Loïc Duflot, Yves-Alexis Perez, Guillaume Valadon, et Olivier Levillain. Can you still trust your Network Card ? *In CanSecWest/core10*, Vancouver (Canada). 24-26 mars 2010. http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf.

[Duran et Ntafos 84] Joe W. Duran et Simeon C. Ntafos. An Evaluation of Random Testing. *IEEE Transactions on Software Engineering*, 10(4):438–444, IEEE Press, Piscataway (NJ, USA), juillet 1984. ISSN 0098-5589. http://dx.doi.org/10.1109/TSE.1984.5010257.

[Dyer 92] Michael Dyer. *The Cleanroom Approach to Quality Software Development*. John Wiley & Sons, Inc., New York (NY, USA), 1992. ISBN 979-0-471-54823-5.

[Ebenau et Strauss 94] Robert G. Ebenau et Susan H. Strauss. *Software Inspection Process*. McGraw-Hill systems design & implementation series. McGraw-Hill, New York (NY, USA), 1994. ISBN 978-0-070-62166-4.

[Eddington 12] Michael Eddington. Peach Fuzzing Platform. Page consultée en 2012. http://peachfuzzer.com.

[Eichin et Rochlis 89] Mark W. Eichin et Jon A. Rochlis. With Microscope and Tweezers : An Analysis of the Internet Virus of November 1988. *In Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland (CA, USA). IEEE Computer Society, mai 1989. ftp://athena-dist.mit.edu/pub/virus/mit.PS.

[ErrProne 12] ErrProne. Jynx-Kit Release 2. 18 mars 2012. http://packetstormsecurity.org/files/110942/Jynx-Kit-Release-2.html.

[EurAsia 10] EurAsia. PS3 Glitch Hack. 8 avril 2010. http://www.eurasia.nu/wiki/index.php/PS3_Glitch_Hack.

[Fagan 76] Michael E. Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3):182–211, IBM Corporation, Riverton (NJ, USA), juin 1976. ISSN 0018-8670. http://dx.doi.org/10.1147/sj.382.0258.

[Falliere *et al.* 11] Nicolas Falliere, Liam O Murchu, et Eric Chien. W32.stuxnet dossier, version 1.4. Rapport technique, Symantec Security Response, Cupertino (CA, USA), février 2011. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.

[Filiol 03] Éric Filiol. *Les virus informatiques : théorie, pratique et applications*. Collection Iris. Springer Paris, Paris (France), 2003. ISBN 978-2-287-98199-9. http://dx.doi.org/10.1007/978-2-287-98240-8.

[Floyd 67] Robert W. Floyd. Assigning Meanings to Programs. *Mathematical Aspects of Computer Science*, 19(19-32):19–32, American Mathematical Society, Providence (RI, USA), 1967. ISBN 978-0-821-86728-0. http://www.eecs.berkeley.edu/~necula/Papers/FloydMeaning.pdf.

[Forrester et Miller 00] Justin E. Forrester et Barton P. Miller. An empirical study of the robustness of Windows NT applications using random testing. *In Proceedings of the 4*th *conference on USENIX Windows Systems Symposium (WSS 00) - Volume 4*, page 6, Seattle (WA, USA). USENIX Association, Berkeley (CA, USA), 2000. http://usenix.org/events/usenix-win2000/full_papers/forrester/forrester.pdf.

[Ganesh *et al.* 09] Vijay Ganesh, Tim Leek, et Martin Rinard. Taint-based directed whitebox fuzzing. *In Proceedings of the 31*st *International Conference on Software Engineering (ICSE 09)*, pages 474–484, Vancouver (Canada). IEEE Computer Society, Washington DC (WA, USA), 16-24 mai 2009. ISBN 978-1-4244-3453-4. http://dx.doi.org/10.1109/ICSE.2009.5070546.

[Gazet 11] Alexandre Gazet. Sticky fingers & KBC Custom Shop. *In Actes du 9e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 11)*, pages 180–193, Rennes (France). 8-10 juin 2011. http://www.sstic.org/media/SSTIC2011/SSTIC-actes/sticky_fingers_and_kbc_custom_shop/.

[Giuliani 11] Marco Giuliani. Mebromi : the first BIOS rootkit in the wild. Webroot, Broomfield (CO, USA), 13 septembre 2011. http://blog.webroot.com/2011/09/13/.

[Glory et Bergerand 90] Anne Cécile Glory et Jean-Louis Bergerand. SAGA : conception de logiciels et preuve de propriétés - L'approche synchrone. *Génie Logiciel et Systèmes Experts*, (18):37–44, 1990.

[Godefroid *et al.* 12] Patrice Godefroid, Michael Y. Levin, et David Molnar. SAGE : Whitebox Fuzzing for Security Testing. *Queue*, 10(1):20–27, ACM Press, New York (NY, USA), janvier 2012. http://dx.doi.org/10.1145/2090147.2094081.

[Gross et Yellen 03] Jonathan L. Gross et Jay Yellen. *Handbook of Graph Theory*. Discrete Mathematics and Its Applications. CRC Press, Boca Raton (FL, USA), 29 décembre 2003. ISBN 978-1584880905.

[Gruskovnjak 12] Jordan Gruskovnjak. Advanced Exploitation of Xen Hypervisor Sysret VM Escape Vulnerability. Vupen Security, Montpellier (France), 4 septembre 2012. http://www.vupen.com/blog/20120904.Advanced_Exploitation_of_Xen_Sysret_VM_Escape_CVE-2012-0217.php.

[Guttag et Horning 93] John V. Guttag et James J. Horning. *Larch : Languages and Tools for Formal Specification*. Springer New York, New York (NY, USA), 1993. ISBN 978-1-4612-7636-4. http://dx.doi.org/10.1007/978-1-4612-2704-5.

[halflife 97] halflife. Shared Library Redirection Techniques. *Phrack Magazine*, 51(8), 1 septembre 1997. http://www.phrack.org/issues.html?id=8&issue=51.

[Harel *et al.* 90] David Harel, Amir Pnueli, Hagi Lachover, Amnon Naamad, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, et Mark Trakhtenbrot. STATEMATE : A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, IEEE Press, Piscataway (NJ, USA), avril 1990. http://dx.doi.org/10.1109/32.54292.

[Heasman 07] John Heasman. Implementing and Detecting a PCI Rootkit. *In BlackHat DC*, Washington DC (WA, USA). 26 février - 1 mars 2007. http://www.blackhat.com/presentations/bh-dc-07/Heasman/Paper/bh-dc-07-Heasman-WP.pdf.

[Hoare 69] C A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–580, ACM Press, New York (NY, USA), octobre 1969. ISSN 0001-0782. http://dx.doi.org/10.1145/363235.363259.

[Holzleitner 09] Jürgen Holzleitner. Using feedback to improve black box fuzz testing of SAT solvers. Mémoire de D.E.A., Institut for Formal Model and Verification, Johannes Kepler University Linz, Linz (Austria), décembre 2009. http://80.66.43.7/~jh/UniProjects/images/fuzz/fuzz.pdf.

[Howard 97] John Douglas Howard. *An Analysis of Security Incidents on the Internet*. Thèse de doctorat, Carnegie Mellon University, Pittsburgh (PA, USA), avril 1997. www.cert.org/archive/pdf/JHThesis.pdf.

[IceLord 07] IceLord. BIOS Rootkit : Welcome home, my Lord ! 26 avril 2007. http://blog.csdn.net/icelord/article/details/1604884.

[Igure et Williams 08] Vinay M. Igure et Ronald D. Williams. Taxonomies of Attacks and Vulnerabilities in Computer Systems. *IEEE Communications Surveys & Tutorials*, 10(1):6–19, IEEE Press, Piscataway (NJ, USA), janvier-avril 2008. ISSN 1553-877X. http://dx.doi.org/10.1109/COMST.2008.4483667.

[ITSEC 91] ITSEC. *Information Technology Security Evaluation Criteria – Provisional Harmonised Criteria*. Commission of the European Communities. DG XIII., Document COM(90) 314, Office for Official Publications of the European Communities, Luxembourg, juin 1991. ISBN-10 9-2826-3004-8. http://www.ssi.gouv.fr/site_documents/ITSEC/ITSEC-uk.pdf.

[Jones 90] Cliff B. Jones. *Systematic Software Development using VDM*. Prentice-Hall, Inc., Upper Saddle River (NJ, USA), 1990. ISSN 978-0-13-880733-7. http://www.vdmbook.com/jones90.pdf.

[Kennedy 10] David Kennedy. Hacking your perimeter - Not everyone needs to use zero days .... 2010. http://www.secmaniac.com/files/Hacking%20the%20perimeter.pdf.

[Kocher *et al.* 99] Paul C. Kocher, Joshua Jaffe, et Benjamin Jun. Differential Power Analysis. *In* Michael Wiener, éditeur : *Proceedings of the 19*th *Annual International Cryptology Conference on Advances in Cryptology (CRYPTO 99)*, volume 1666 de *Lecture Notes in Computer Science*, pages 388–397. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66347-8. http://dx.doi.org/10.1007/3-540-48405-1_25.

[Krahmer 05] Sebastian Krahmer. x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique. Rapport technique, 28 septembre 2005. http://www.suse.de/~krahmer/no-nx.pdf.

[Krsul 98] Ivan Krsul. *Software Vulnerability Analysis*. Thèse de doctorat, Department of Computer Sciences, Purdue University, West Lafayette (IN, USA), 1998. https://www.cerias.purdue.edu/techreports-ssl/public/98-09.pdf.

[Laarouchi 09] Youssef Laarouchi. *Sécurités (immunité et innocuité) des architectures ouvertes à niveaux de criticité multiples : application en avionique*. Thèse de doctorat, Institut National des Sciences Appliquées (INSA) de Toulouse, Toulouse (France), 30 novembre 2009. http://tel.archives-ouvertes.fr/docs/00/46/89/23/PDF/Manuscrit_Youssef_Laarouchi.pdf.

[Lacombe *et al.* 09] Éric Lacombe, Vincent Nicomette, et Yves Deswarte. A Hardware-Assisted Virtualization-Based Approach on How to Protect the Kernel Space from Malicious Actions. *In Proceedings of the 18*th *EICAR Annual Conference*, Berlin (Germany). 11-12 mai 2009. http://www.eicar.org/files/eicar2009-elacombe-slides.pdf.

[Landwehr *et al.* 94] Carl E. Landwehr, Alan R. Bull, John P. McDermott, et William S. Choi. A Taxonomy of Computer Program Security Flaws, with Examples. *ACM Computer Survey*, 26(3):211–254, ACM Press, New York (NY, USA), septembre 1994. ISSN 0360-0300. http://dx.doi.org/10.1145/185403.185412.

[Laprie 04] Jean-Claude Laprie. Sûreté de fonctionnement des systèmes : concepts de base et terminologie. *Revue de l'Électricité et de l'Électronique (REE)*, (11):95–105, novembre 2004.

[Laprie *et al.* 96] Jean-Claude Laprie, Jean Arlat, Jean-Paul Blanquart, Alain Costes, Yves Crouzet, Yves Deswarte, Jean-Charles Fabre, Hubert Guillermain, Mohamed Kaâniche, Karama Kanoun, Corinne Mazet, David Powell, Christophe Rabéjac, et Pascale Thévenod. *Guide de la Sûreté de Fonctionnement*. Cépaduès, 2e édition, 1996. ISBN 978-2-854-28382-2.

[LeCroy Corporation 12] LeCroy Corporation. *Summit Z3-16 PCI Express Multi-Lane Exerciser - Product Datasheet*, mai 2012. http://cdn.lecroy.com/files/pdf/lecroy_summit_z3-16_datasheet.pdf.

[Li *et al.* 11] Yanlin Li, Jonathan M. McCune, et Adrian Perrig. VIPER : verifying the integrity of PERipherals' firmware. *In Proceedings of the 18*th *ACM conference on Computer and Communications Security (CCS 11)*, pages 3–16, Chicago (IL, USA). ACM Press, New York (NY, USA), 17-21 octobre 2011. ISBN 978-1-4503-0948-6. http://dx.doi.org/10.1145/2046707.2046711.

[Lindqvist et Jonsson 97] Ulf Lindqvist et Erland Jonsson. How to Systematically Classify Computer Security Intrusions. *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 154–163, Oakland (CA, USA). IEEE Computer Society, may 1997. http://dx.doi.org/10.1109/SECPRI.1997.601330.

[Lineberry 09] Anthony Lineberry. Alice in User-Land : Hijacking the Linux Kernel via `/dev/mem`. *BlackHat Europe*, Amsterdam (The Netherlands), 14-17 avril 2009. http://www.blackhat.com/presentations/bh-europe-09/Lineberry/BlackHat-Europe-2009-Lineberry-code-injection-via-dev-mem.pdf.

[Lone Sang *et al.* 10] Fernand Lone Sang, Vincent Nicomette, et Yves Deswarte. Démonstration d'une attaque par partage d'identifiants entre un contrôleur FireWire et un contrôleur réseau. janvier 2010. http://homepages.laas.fr/nicomett/Videos/.

[Lone Sang *et al.* 11a] Fernand Lone Sang, Vincent Nicomette, et Yves Deswarte. Attaques par entrée-sortie et contremesures. *In Actes de la Journée Sécurité des Systèmes & Sureté des Logiciels (3SL)*, pages 11–13, Saint-Malo (France). 10 mai 2011. http://www.univ-orleans.fr/lifo/evenements/3SL/actes/3sl.pdf.

[Lone Sang *et al.* 11b] Fernand Lone Sang, Vincent Nicomette, et Yves Deswarte. Démonstration d'une attaque pair-à-pair depuis un périphérique FireWire vers un contrôleur graphique. janvier 2011. http://homepages.laas.fr/nicomett/Videos/.

[Lone Sang *et al.* 11c] Fernand Lone Sang, Vincent Nicomette, et Yves Deswarte. I/O attacks in Intel PC-based architectures and countermeasures. *In Proceedings of the 1*st *SysSec Workshop*, pages 18–25, Amsterdam (The Netherlands). 6 juillet 2011. http://www.syssec-project.eu/media/page-media/3/syssec-d2.3-1st-project-workshop-proceedings.pdf.

[Lone Sang *et al.* 12a] Fernand Lone Sang, Vincent Nicomette, et Yves Deswarte. Démonstration d'une attaque par accès pair-à-pair depuis ironhide. janvier 2012. http://homepages.laas.fr/nicomett/Videos/.

[Lone Sang *et al.* 12b] Fernand Lone Sang, Vincent Nicomette, et Yves Deswarte. IronHide : plate-forme d'attaques par entrées-sorties. *In Actes du 10e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 12)*, pages 237–265, Rennes, France. 6-8juin 2012.

[Lone Sang *et al.* 11] Fernand Lone Sang, Vincent Nicomette, Yves Deswarte, et Loïc Duflot. Attaques DMA *peer-to-peer* et contre-mesures. *In Actes du 9e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 11)*, pages 150–179, Rennes (France). 8-10 juin 2011. http://www.sstic.org/media/SSTIC2011/SSTIC-actes/attaques_dma_peer-to-peer_et_contremesures/.

[Lone Sang *et al.* 10a] Fernand Lone Sang, Éric Lacombe, Vincent Nicomette, et Yves Deswarte. Analyse de l'efficacité du service fourni par une IOMMU. *In Actes du 8e Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC 10)*, pages 189–214, Rennes (France). 9-11 juin 2010. http://www.sstic.org/media/SSTIC2010/SSTIC-actes/Analyse_de_l_efficacite_du_service_fourni_par_une_/.

[Lone Sang *et al.* 10b] Fernand Lone Sang, Éric Lacombe, Vincent Nicomette, et Yves Deswarte. Exploiting an I/OMMU vulnerability. *In Proceedings of the 5*th *International Conference on Malicious and Unwanted Software (MALWARE 10)*, pages 7 –14, Nancy (France). IEEE, 19-20 octobre 2010. ISBN 978-1-4244-9355-5. http://dx.doi.org/10.1109/MALWARE.2010.5665798.

[Lough 01] Daniel L. Lough. *A Taxonomy of Computer Attacks with Applications to Wireless Networks*. Thèse de doctorat, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg (VA, USA), avril 2001. http://scholar.lib.vt.edu/theses/available/etd-04252001-234145/unrestricted/lough.dissertation.pdf.

[Martin 07] 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.

[May et Woods 79] Timothy C. May et Murray H. Woods. Alpha-Particle-Induced Soft Errors in Dynamic Memories. *IEEE Transactions on Electron Devices*, 26(1):2–9, Institute of Electrical and Electronics Engineers, New York (NY, USA), jan 1979. ISSN 0018-9383. http://dx.doi.org/10.1109/T-ED.1979.19370.

[Maynor 05] David Maynor. 0wn3d by everything else - USB/PCMCIA Issues. *In CanSecWest/-core05*, Vancouver (Canada). 4-5 mai 2005. http://cansecwest.com/core05/DMA.ppt.

[Miller *et al.* 90] Barton P. Miller, Louis Fredriksen, et Bryan So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, ACM Press, New York (NY, USA), décembre 1990. ISSN 0001-0782. http://dx.doi.org/10.1145/96267.96279.

[Miller *et al.* 07] Barton P. Miller, Gregory Cooksey, et Fredrick Moore. An empirical study of the robustness of MacOS applications using random testing. *SIGOPS Operating Systems Review*, 41(1):78–86, ACM Press, New York (NY, USA), janvier 2007. ISSN 0163-5980. http://dx.doi.org/10.1145/1228291.1228308.

[Miller et Peterson 07] Charlie Miller et Zachary N. J. Peterson. Analysis of Mutation and Generation-Based Fuzzing. Rapport technique, Independent Security Evaluators, mars 2007. http://securityevaluators.com/files/papers/analysisfuzzing.pdf.

[ModularLogix 10] ModularLogix. MLX-1000-XC5V Module - Product Data Sheet. 4 octobre 2010. http://files.modularlogix.com/Documents/MLX-1000-XC5V/MLX-1000-XC5V%20Datasheet.pdf.

[Myers *et al.* 04] Glenford J. Myers, Corey Sandler, Tom Badgett, et Todd M. Thomas. *The Art of Software Testing*. Business Data Processing : a Wiley Series. John Wiley & Sons, Inc., New York (NY, USA), 2004. ISBN 978-0-471-46912-4. http://dx.doi.org/10.1002/stvr.322/.

[Namestnikov 12] Yury Namestnikov. Kaspersky Security Bulletin - Statistics 2011. Kaspersky Lab ZAO, 01 mars 2012. http://www.securelist.com/en/analysis/204792216/Kaspersky_Security_Bulletin_Statistics_2011.

[Nergal 01] Nergal. The Advanced Return-into-lib(c) Exploits : PaX case study. *Phrack Magazine*, 58(4), 28 décembre 2001. http://www.phrack.org/issues.html?issue=58&id=4.

[Neumann et Parker 89] Peter G. Neumann et Donn B. Parker. A Summary of Computer Misuse Techniques. *In Proceedings of the 12*th *National Computer Security Conference*, pages 396–407, Baltimore (MD, USA). National Institute of Standards and Technology/National Computer Security Center (NIST/NCSC), 10-13 octobre 1989.

[Oehlert 05] Peter Oehlert. Violating Assumptions with Fuzzing. *IEEE Security and Privacy*, 3(2):58–62, IEEE Educational Activities Department, Piscataway (NJ, USA), mars 2005. http://dx.doi.org/10.1109/MSP.2005.55.

[Parker 75] Donn B. Parker. Computer Abuse Perpetrators and Vulnerabilities of Computer Systems. *In Proceedings of the National Computer Conference and Exposition*, American Federation of Information Processing Societies (AFIPS). ACM Press, New York (NY, USA), 7-10 juin 1975. http://dx.doi.org/10.1145/1499799.1499810.

[Petri 62] Carl Adam Petri. *Kommunikation mit Automaten*. Thèse de doctorat, Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, Universität Hamburg, Hamburg (Germany), 20 juin 1962. http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

[Petroni *et al.* 04] Nick L. Jr. Petroni, Timothy Fraser, Jesus Molina, et William A. Arbaugh. Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor. *In Proceedings of the 13*th *USENIX Security Symposium (SSYM 04)*, San Diego (CA, USA). USENIX Association, Berkeley (CA, USA), 9-13 août 2004. http://www.usenix.org/events/sec04/tech/full_papers/petroni/petroni.pdf.

[Pico Computing 11] Pico Computing. Pico Computing - the FPGA Computing Experts. Page consultée en 2011. http://www.picocomputing.com/e_series.html.

[Pisani *et al.* 10] Jason Pisani, Paul Caruga, et Richard Rushing. USB-HID - Hacker Interface Design. *In BlackHat USA*, Las Vegas (NV, USA). 28-29 juillet 2010. https://media.blackhat.com/bh-us-10/presentations/Rushing/BlackHatUSA-2010-Rushing-USB-HID-slides.pdf.

[Pragmatic et THC 99] Pragmatic et THC. (nearly) Complete Linux Loadable Kernel Modules. The definitive guide for hackers, virus coders and system administrators. mai 1999. http://www.thc.org/papers/LKM_HACKING.html.

[QiHoo 360 11] QiHoo 360. 360 has published a technical analysis of the BMW virus. Qihoo 360 Technology Co. Ltd., Beijing (China), 2 septembre 2011. http://bbs.360.cn/4005462/251096134.html.

[Queille et Sifakis 82] Jean-Pierre Queille et Joseph Sifakis. Specification and Verification of Concurrent Systems in CESAR. *In* Mariangiola Dezani-Ciancaglini et Ugo Montanari, éditeurs : *Proceedings of the 5*th *Colloquium on International Symposium on Programming*, volume 137 de *Lecture Notes in Computer Science*, pages 337–351, London (GB). Springer Berlin Heidelberg, 1982. ISBN 978-3-540-11494-9. http://dl.acm.org/citation.cfm?id=647325.721668.

[Rogers et Ruppersberger 12] Mike Rogers et Dutch Ruppersberger. Investigative Report on the U.S. National Security Issues Posed by Chinese Telecommunications Companies Huawei and ZTE. Rapport d'enquête, Permanent Select Committee on Intelligence, U.S. House of Representatives, 8 octobre 2012. http://intelligence.house.gov/sites/intelligence.house.gov/files/Huawei-ZTE%20Investigative%20Report%20(FINAL).pdf.

[Roper 92] Marc Roper. Software Testing : A Selected Annotated Bibliography. *Software Testing, Verification and Reliability*, 2(3):113–132, John Wiley & Sons, Inc., New York (NY, USA), 1992. ISSN 1099-1689. http://dx.doi.org/10.1002/stvr.4370020303.

[Rushby *et al.* 91] J. Rushby, F. von Henke, et S. Owre. An Introduction to Formal Specification and Verification Using EHDM. Rapport technique numéro SRI-CSL-91-02, Menlo Park (CA, USA), 1991. http://www.csl.sri.com/papers/csl-91-2/csl-91-2.ps.

[Rutkowska 07] Joanna Rutkowska. Beyond The CPU : Defeating Hardware Based RAM Acquisition. *In BlackHat DC*, Washington DC (WA, USA). 26-27 février 2007. http://www.blackhat.com/presentations/bh-dc-07/Rutkowska/Presentation/bh-dc-07-Rutkowska-up.pdf.

[Rutkowska et Wojtczuk 08] Joanna Rutkowska et Rafał Wojtczuk. Preventing and Detecting Xen Hypervisor Subversions. *In Black Hat USA*, Las Vegas (NV, USA). 6-7 août 2008. http://invisiblethingslab.com/resources/bh08/part2-full.pdf.

[Schneier 99] Bruce Schneier. Attack Trees - Modeling security threats. *Dr. Dobb's Journal of Software Tools*, 24(12), 1 décembre 1999. http://www.drdobbs.com/%20architect/184411129.

[Shacham 07] Hovav Shacham. The Geometry of Innocent Flesh on the Bone : Return-into-libc without Function Calls (on the x86). *In Proceedings of the 14*th *ACM conference on Computer and Communications Security (CCS 07)*, pages 552–561, Alexandria (VA, USA). ACM Press, New York (NY USA), 29 octobre - 2 novembre 2007. ISBN 978-1-59593-703-2. http://dx.doi.org/10.1145/1315245.1315313.

[Shamir et Tromer 04] Adi Shamir et Eran Tromer. Acoustic cryptanalysis - On nosy people and noisy machines. Rump Session at the 23th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT O4), Interlaken (Switzerland), 2-6 mai 2004. http://tau.ac.il/~tromer/acoustic/.

[Sheyner 04] Oleg Mikhail Sheyner. *Scenario Graphs and Attack Graphs*. Thèse de doctorat, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh (PA, USA), 14 avril 2004. http://reports-archive.adm.cs.cmu.edu/anon/anon/usr0/ftp/usr/ftp/2004/CMU-CS-04-122.pdf.

[Shirey 94] Robert W. Shirey. Security Architecture for Internet Protocols : A Guide for Protocol Designs and Standards. novembre 1994. Internet Draft : draft-irtf-psrg-secarch-sect1-00.

[Skorobogatov et Anderson 03] Sergei P. Skorobogatov et Ross J. Anderson. Optical Fault Induction Attacks. *In* Burton S. Kaliski, çetin K. Koç, et Christof Paar, éditeurs : *Proceeding of the 4*th *International Workshop on Cryptographic Hardware and Embedded Systems (CHES 02)*, volume 2523 de *Lecture Notes in Computer Science*, pages 2–12, Redwood City (CA, USA). Springer Berlin Heidelberg, 2003. ISBN 978-3-540-00409-7. http://dx.doi.org/10.1007/3-540-36400-5_2.

[Solar Designer 97] Solar Designer. Getting around non-executable stack (and fix). 10 août 1997. http://seclists.org/bugtraq/1997/Aug/63.

[Song *et al.* 01] Dawn X. Song, David Wagner, et Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. *In Proceedings of the 10th USENIX Security Symposium (SSYM 01)*, pages 25–25, Washington DC (WA, USA). USENIX Association, Berkeley (CA, USA), 13-17 août 2001. http://static.usenix.org/events/sec01/full_papers/song/song.pdf.

[Spafford 88] Eugene H. Spafford. The Internet Worm Program : An Analysis. Rapport technique numéro CSD-TR-823, Department of Computer Sciences, Purdue University, West Lafayette (IN, USA), 1988. http://spaf.cerias.purdue.edu/tech-reps/823.pdf.

[Spengler 12] Brad Spengler. grsecurity. Page consultée en 2012. http://grsecurity.net/.

[Spivey 88a] Mike J. Spivey. *Understanding Z : A Specification Language and its Formal Semantics*, volume 3 de *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, New York (NY, USA), 1988. ISBN 978-0-521-05414-0.

[Spivey 92b] Mike J. Spivey. *The Z Notation : A Reference Manual*. International Series in Computer Science. Prentice Hall International, Ltd., Hertfordshire (GB), 1992. ISBN 978-0-139-78529-0. http://spivey.oriel.ox.ac.uk/~mike/zrm/.

[styxˆ 12] styx÷ Infecting Loadable Kernel Modules - Kernel Versions 2.6.x/3.0.x. *Phrack Magazine*, 68(11), 14 avril 2012. http://www.phrack.org/issues.html?issue=68&id=11#article.

[Sutton *et al.* 07] Michael Sutton, Adam Greene, et Pedram Amini. *Fuzzing : Brute Force Vulnerability Discovery*. Addison-Wesley Professional, 2007.

[ter Huurne 12] Maarten ter Huurne. [PATCH] /dev/mem : Add kernel config option to omit this device. 29 mars 2012. http://lkml.org/lkml/2012/3/29/403.

[TESO 04] Team TESO. Adore-ng Rootkit. 6 janvier 2004. http://stealth.7350.org/rootkits/adore-ng-0.23.tgz.

[Thevenod-Fosse 91] Pascale Thevenod-Fosse. Software Validation by Means of Statistical Testing : Retrospect and Future Direction. *In* Algirdas Avižienis et Jean-Claude Laprie, éditeurs : *Proceeding of the 1*st *IFIP International Working Conference on Dependable Computing for Critical Applications (DCCA)*, volume 4 de *Dependable Computing and Fault-Tolerant Systems*, pages 23–50, Santa Barbara (CA, USA). Springer Vienna (Autriche), 1991. ISBN 978-3-7091-9125-5. http://dx.doi.org/10.1007/978-3-7091-9123-1_2.

[Trefis 12] Trefis. ANALYSIS for INTEL. Rapport technique, Trefis, 100 City Hall Plaza, Boston (MA, USA), 2 août 2012. http://pdfs.trefis.com/6298-FHoMvURdX8S27OEB/Intel_2012-08-02.pdf.

[Trefis Team 12] Trefis Team. Can Intel Continue To Dominate The PC Microprocessor Market ? Forbes Magazine, 31 mai 2012. http://www.forbes.com/sites/greatspeculations/2012/05/31/can-intel-continue-to-dominate-the-pc-microprocessor-market/.

[Triulzi 08a] Arrigo Triulzi. Project Maux Mk.II - « I 0wn the NIC, Now I want a Shell ! ». *In PacSec/core08*, Tokyo (Japan). 12-13 novembre 2008. http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-PACSEC08-Project-Maux-II.pdf.

[Triulzi 10b] Arrigo Triulzi. The Jedi Packet Trick takes over the Deathstar (or : « Taking NIC Backdoors to the Next Level »). *In CanSecWest/core10*, Vancouver (Canada). 24-26 mars 2010. http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-CANSEC10-Project-Maux-III.pdf.

[truff 03] truff. Infecting Loadable Kernel Modules. *Phrack Magazine*, 61(10), 13 août 2003. http://www.phrack.org/issues.html?issue=61&id=10#article.

[van de Ven 08] Arjan van de Ven. [PATCH] make /dev/kmem a config option. 10 février 2008. http://lkml.org/lkml/2008/2/10/316.

[van Eck 85] Wim van Eck. Electromagnetic Radiation from Video Display Units : An Eavesdropping Risk ? *Computers & Security*, 4(4):269–286, Elsevier Advanced Technology Publication, Oxford (GB), décembre 1985. ISSN 0167-4048. http://dx.doi.org/10.1016/0167-4048(85)90046-X.

[van Emden 92] Maarten H van Emden. Structured Inspections of Code. *Journal of Software Testing, Verification, and Reliability*, 2:133–153, John Wiley & Sons, Inc., New York (NY, USA), 1992. http://dx.doi.org/10.1002/stvr.4370020304.

[von Neumann 45] John von Neumann. First Draft of a Report on the EDVAC. Rapport technique , Contrat numéro W-670-ORD-4926, Moore School of Electrical Engineering, University of Pennsylvania, Pennsylvania (PA, USA), 30 juin 1945. http://dx.doi.org/10.1109/85.238389.

[Vuagnoux et Pasini 09] Martin Vuagnoux et Sylvain Pasini. Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. *In Proceedings of the 18*th *USENIX Security Symposium (SSYM 09)*, pages 1–16, Montreal (Canada). USENIX Association, Berkeley (CA, USA), 2009. https://www.usenix.org/legacy/event/sec09/tech/full_papers/vuagnoux.pdf.

[Waeselynck 93] Hélène Waeselynck. *Vérification de logiciels critiques par le test statistique*. Thèse de doctorat, Institut National Polytechnique (INP) de Toulouse, Toulouse (France), 19 janvier 1993. http://homepages.laas.fr/waeselyn/papers/these_Helene_WAESELYNCK.pdf.

[Weyuker 82] Elaine J. Weyuker. On Testing Non-Testable Programs. *The Computer Journal*, 25(4):465–470, 1982. http://dx.doi.org/10.1093/comjnl/25.4.465.

[Wojtczuk et Rutkowska 09a] Rafał Wojtczuk et Joanna Rutkowska. Attacking SMM Memory via Intel CPU Cache Poisoning. Rapport technique, Invisible Things Lab (ITL), 19 mars 2009. http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf.

[Wojtczuk et Rutkowska 11b] Rafał Wojtczuk et Joanna Rutkowska. Following the White Rabbit : Software Attacks against Intel VT-d. Rapport technique, Invisible Things Lab (ITL), mai 2011. http://www.invisiblethingslab.com/resources/2011/Software%20Attacks%20on%20Intel%20VT-d.pdf.

[Wojtczuk *et al.* 09] Rafał Wojtczuk, Joanna Rutkowska, et Alexander Tereshkin. Another Way to Circumvent Intel Trusted Execution Technology - Tricking SENTER into misconfiguring VT-d via SINIT bug exploitation. Rapport technique, Invisible Things Lab, décembre 2009. http://invisiblethingslab.com/resources/misc09/Another%20TXT%20Attack.pdf.

[Wright 87] Peter Wright. *Spycatcher—The Candid Autobiography of a Senior Intelligence Officer*. Viking Pr, juillet 1987. ISBN 0-85561-098-0.

[Xilinx 10a] Xilinx. LogiCORE IP PLBv46 RC/EP Bridge for PCI Express - Product Specification. 14 décembre 2010. http://www.xilinx.com/support/documentation/ip_documentation/plbv46_pcie.pdf.

[Xilinx 11b] Xilinx. LogiCORE IP Endpoint Block Plus for PCI Express - Product Specification. 22 juin 2011. http://www.xilinx.com/support/documentation/ip_documentation/pcie_blk_plus/v1_15/pcie_blk_plus_ds551.pdf.

[Yen *et al.* 03] Sung-Ming Yen, Sangjae Moon, et Jae-Cheol Ha. Hardware Fault Attack on RSA with CRT Revisited. *In* PilJoong Lee et ChaeHoon Lim, éditeurs : *Proceedings of the 5th International Conference on Information Security and Cryptology (ICISC 02)*, volume 2587 de *Lecture Notes in Computer Science*, pages 374–388, Seoul (Korea). Springer Berlin Heidelberg, 2003. ISBN 978-3-540-00716-6. http://dl.acm.org/citation.cfm?id=1765361.1765394.

Résumé

Les attaques ciblant les systèmes informatiques vont aujourd'hui au delà de simples logiciels malveillants et impliquent de plus en plus des composants matériels. Cette thèse s'intéresse à cette nouvelle classe d'attaques et traite, plus précisément, des attaques par entrées-sorties qui détournent des fonctionnalités légitimes du matériel, tels que les mécanismes entrées-sorties, à différentes fins malveillantes. L'objectif est d'étudier ces attaques, qui sont extrêmement difficiles à détecter par des techniques logicielles classiques (dans la mesure où leur mise en œuvre ne nécessite pas l'intervention des processeurs) afin de proposer des contre-mesures adaptées, basées sur des composants matériels fiables et incontournables. Ce manuscrit se concentre sur deux cas : celui des composants matériels qui peuvent être délibérément conçus pour être malveillants et agissants de la même façon qu'un programme intégrant un cheval de Troie ; et celui des composants matériels vulnérables qui ont été modifiés par un pirate informatique, localement ou au travers du réseau, afin d'y intégrer des fonctions malveillantes (typiquement, une porte dérobée dans son *firmware*). Pour identifier les attaques par entrées-sorties, nous avons commencé par élaborer un modèle d'attaques qui tient compte des différents niveaux d'abstraction d'un système informatique. Nous nous sommes ensuite appuyés sur ce modèle d'attaques pour les étudier selon deux approches complémentaires : une analyse de vulnérabilités traditionnelle, consistant à identifier une vulnérabilité, développer des preuves de concept et proposer des contre-mesures ; et une analyse de vulnérabilités par *fuzzing* sur les bus d'entrées-sorties, reposant sur un outil d'injection de fautes que nous avons conçu, baptisé IronHide, capable de simuler des attaques depuis un composant matériel malveillant. Les résultats obtenus pour chacunes de ces approches sont discutés et quelques contre-mesures aux vulnérabilités identifiées, basées sur des composants matériels existants, sont proposées.

Mots-clés: sécurité informatique, attaques par entrées-sorties, modèle d'attaques, analyse de vulnérabilités, *fuzzing*.

Abstract

Nowadays, attacks against computer systems may involve hardware components in order to bypass the numerous countermeasures against malicious software. This PhD thesis focuses on this novel class of attacks and specifically deals with Input/Output attacks. In such attacks, attackers divert legitimate hardware features, such as I/O mechanisms, to achieve different malicious actions. Since detecting such attacks by conventional software techniques is not easy (as far as they do not require the intervention of the CPU), we have analyzed these attacks in order to propose appropriate countermeasures based mainly on reliable and unavoidable hardware components. This manuscript focuses on two cases : hardware components that can be deliberately designed to be malicious and acting in the same way as a program incorporating a Trojan horse ; and vulnerable hardware components that have been modified by a hacker, either locally or through the network, to include malicious functions (typically a backdoor in the firmware). To identify the potential I/O attacks, we developed an attack model which takes into account the different abstraction levels in a computer system. Then, we studied these attacks with two complementary approaches : the classical approach to vulnerability analysis consisting in identifying a vulnerability, developing a proof-of-concept and proposing countermeasures ; and fuzzing-based vulnerability analysis, using IronHide, a fault injection tool we have designed, which is able to simulate a powerful malicious hardware. The results obtained with both approaches are discussed and several countermeasures to the vulnerabilities we identified, based on existing hardware components, are proposed.

Keywords: computer security, I/O attacks, attack model, vulnerability analysis, fuzzing.