Compromission physique par le bus PCI

Christophe Devine, Guillaume Vissian
devine(@)bob.cat
guillaume.vissian(@)lab.b-care.net

Thales Security Systems

Résumé Le bus PCI permet à tout périphérique d'effectuer des accès à la mémoire physique (accès DMA). Cet article a pour but d'illustrer le développement d'une preuve de concept visant à compromettre le système d'exploitation d'une station de travail ou d'un portable par l'intermédiaire d'une carte PCI ou Cardbus ; il présentera finalement les moyens de s'en protéger.

1 Introduction

L'utilisation d'un accès direct à la mémoire dans le but de compromettre un ordinateur, ou inversement à des fins de récupération d'évidences numériques, est connue depuis plusieurs années.

En 1998, Silvio Cesare présente une méthode de compromission du noyau Linux [1] ne nécessitant pas le support des modules noyaux chargeables à la volée. Cette technique repose sur la modification de la table des appels systèmes via le périphérique /dev/kmem, qui fournit un accès à l'ensemble de la mémoire virtuelle depuis un processus utilisateur. Par la suite, sd et devik publient dans Phrack 58 le rootkit SuckIT [2] réputé pour son emploi en 2003 sur plusieurs serveurs Debian compromis ; la conception de ce type de rootkit est traitée de manière plus avancée dans [3].

De façon similaire, il est possible sous Windows XP et Server 2003 pre-SP1 d'accéder à la mémoire physique au travers du nœud de périphérique \Device \PhysicalMemory [4], et à la mémoire virtuelle via l'appel système ZwSystemDebugControl(). Microsoft a retiré ces fonctionnalités avec le Service Pack 1 de Windows Server 2003 et Vista, en conséquence de l'apparition de logiciels malveillants profitant de ces accès ; une technique de contournement a été présentée en 2006 par Alex Ionescu [5].

Les méthodes qui viennent d'être évoquées sont purement logicielles, et requièrent un niveau de privilèges déjà élevé sur le système (root ou Administrateur). Il est toutefois possible d'accomplir les mêmes opérations via un accès physique à la machine cible.

En 2003, Brian Carrier et Joe Grand présentent une carte PCI dédiée à l'acquisition de la mémoire physique à des fins d'investigation post-mortem [6]. Un produit commercial analogue est la carte Raptor de la société KeyCarbon. Dotée d'un FPGA et d'une mémoire Flash, cette carte permet de capturer les touches frappées sur un ordinateur portable. On pourrait imaginer la convertir aussi bien en dispositif de récupération de la mémoire qu'en rootkit matériel difficile à détecter ; il est dommage que le format mini-PCI de cette carte la rende inutilisable sur tout portable récent.

Puis Maximillian Dornseif en 2005 [7] et Adam Boileau en 2006 [8] montrent comment exploiter une connexion FireWire pour lire et écrire dans la mémoire. Selon Adam Boileau, David Hulton aurait présenté à la conférence Schmoocon'06 un exemple d'attaque faisant intervenir une carte PC Card ; malheureusement la présentation n'est pas disponible sur Internet. L'utilisation d'une carte PC Card est aussi mentionnée en 2007 par Nicolas Ruff et Matthieu Suiche [9]. Citons finalement les travaux de Damien Aumaitre [10] qui présente de manière très détaillée les utilisations possibles d'un l'accès à la mémoire physique.

La vulnérabilité exploitée dans le cadre de cet article est l'absence de vérification des accès à la mémoire depuis le bus PCI dans les ordinateurs à base de processeur x86. Originellement pour des raisons de performances, et afin de ne pas occuper inutilement le processeur, les périphériques reliés au bus accèdent directement à l'ensemble de la mémoire physique ; le contrôleur mémoire est transparent et n'effectue pas de translation d'adresse ou de filtrage particulier.

Nous ferons dans la suite du présent article un bref rappel sur les mécanismes de pagination de l'architecture x86, la programmation matérielle et le rôle du bus mastering au sein de la norme PCI. Ensuite nous présenterons la réalisation concrète d'une preuve de concept montrant la compromission du système d'exploitation au travers d'une carte PCI ou PC Card. En dernier lieu seront présentées des contre-mesures permettant de se prémunir contre cette attaque.

2 Rappels

2.1 Mécanismes de pagination x86

On appelle pagination la translation faite par le processeur depuis une adresse virtuelle pour retrouver l'adresse en mémoire physique correspondante. La pagination procure le cloisonnement des processus, ainsi que l'exécution de multiples processus

ayant la même adresse virtuelle de chargement ; usuellement, 0x08048000 sous Linux.

A chaque processus est associé un répertoire de pages dont l'adresse physique est stockée dans le registre CR3 du processeur. Le changement de contexte entre processus est fait par le noyau à intervalle régulier et met à jour la valeur de ce registre ainsi que de plusieurs autres.

En pratique une seule table serait trop volumineuse car l'espace mémoire d'un processus est rempli de trous, c'est-à-dire d'adresses virtuelles n'ayant pas de correspondances dans la mémoire physique. Les ingénieurs d'Intel ont donc conçu un mécanisme de translation à deux niveaux :

- Les bits 31 à 22 inclus de l'adresse virtuelle sont l'index dans le répertoire principal qui contient 1024 mots de 32 bits. Chaque mot est l'adresse physique d'une table de pages, ou bien 0.

- Les dix bits suivants sont l'index dans la table de page, qui contient de même 1024 mots de 32 bits. Pour chaque entrée de la table, les vingt bits de poids fort constituent l'adresse physique de la page alignée sur 4096.

- A cette adresse physique est ajoutée les 12 bits de poids faible de l'adresse virtuelle, ce qui donne l'adresse physique finale (figure 1).

On remarque que les 12 bits de poids faible des mots contenus dans le répertoire et les tables de pages ne semblent pas utilisée, les adresses physiques étant implicitement alignées sur 4096 octets. En fait, ces bits ont un rôle de drapeau pour indiquer l'état de la page : [1]

```
#define PAGE_PRESENT       0x01 /* l'entree est valide */
#define PAGE_WRITABLE      0x02 /* ecriture autorisee  */
#define PAGE_USER          0x04 /* non privilegie      */
#define PAGE_WRITETHROUGH  0x08
#define PAGE_NONCACHEABLE  0x10
```

La pagination est un peu difficile à comprendre de premier abord. En pratique, considérons l'exemple suivant :

[1] Se référer aux manuels Intel [11] pour plus de détails.

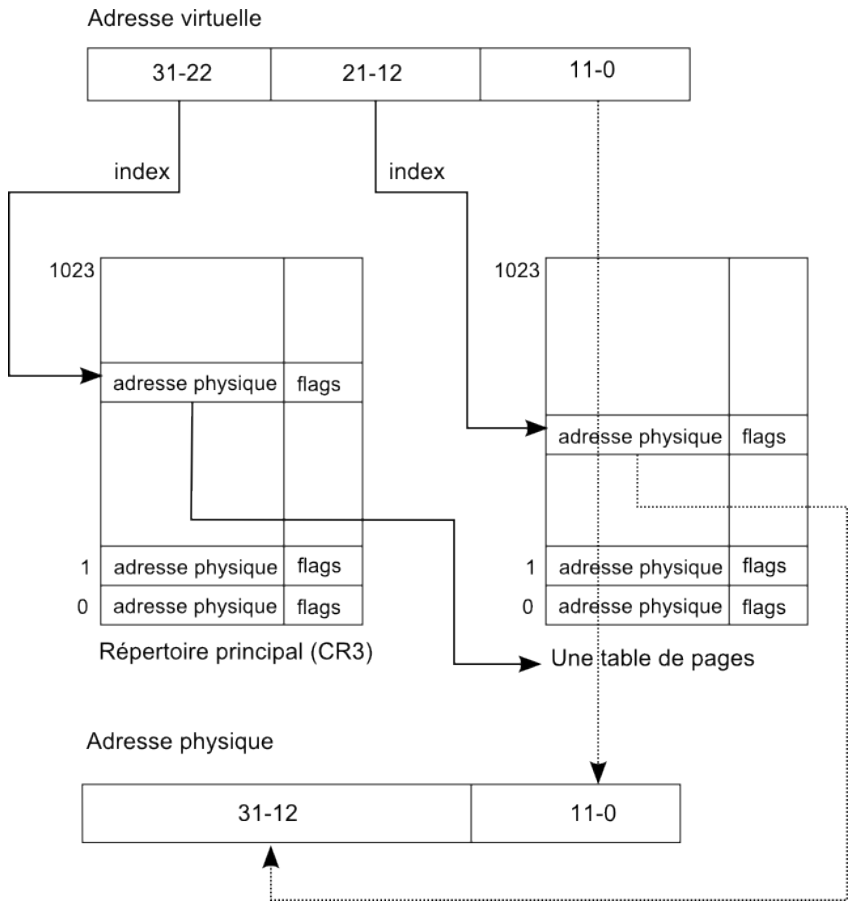

Fig. 1. Translation d'une adresse virtuelle en adresse physique

Le débuggeur WinDbg de Microsoft est connecté à une machine virtuelle Windows XP SP3 tournant sous VMware avec le mode PAE désactivé (/noexecute=alwaysoff /NOPAE). On cherche l'adresse physique à laquelle est mappé le fichier winlogon.exe.

```
kd> !process 0 0
[...]
PROCESS 81bd47c0  SessionId: 0  Cid: 0254    Peb: 7ffdd000  ParentCid: 01fc
    DirBase: 085f5000  ObjectTable: e14313a8  HandleCount: 472.
    Image: winlogon.exe
[...]

kd> lm
start    end        module name
01000000 01082000   winlogon   (deferred)
[...]

kd> db 01000000 L60
01000000  4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00  MZ..............
01000010  b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00  ........@.......
01000020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
01000030  00 00 00 00 00 00 00 00-00 00 00 00 f0 00 00 00  ................
01000040  0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68  ........!..L.!Th
01000050  69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f  is program canno
```

Le processus winlogon est mappé en mémoire virtuelle à l'adresse 0x1000000. D'autre part DirBase correspond à la valeur du registre CR3 pour le processus. Pour examiner la mémoire physique, il faut ajouter un point d'exclamation devant la commande de lecture : [2]

```
kd> !dd 85f5000 L400
# 85f5000 08694067 086df067 089ea067 08cfb067
# 85f5010 08693067 09a62067 0a2e5067 09783067
# 85f5020 00000000 00000000 00000000 00000000
# 85f5030 00000000 00000000 00000000 00000000
[...]
# 85f57e0 0866e067 087c8067 00000000 00000000
# 85f57f0 00000000 086a4067 00000000 08691067
# 85f5800 0003b163 004001e3 0003e163 010001e3
# 85f5810 014001e3 018001e3 01c001e3 020001e3
[...]

kd> !dd 08693000 L40
# 8693000 05d7f025 08670025 00000000 00000000
# 8693010 00000000 00000000 00000000 00000000
# 8693020 016014fc 016014fe 0652c025 0652d025
# 8693030 01601c04 0652f025 06530025 01601c0a

kd> !db 05d7f000 L60
# 5d7f000 4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00 MZ..............
# 5d7f010 b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......
# 5d7f020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

[2] WinDbg dispose par ailleurs d'extensions spécifiques (!address, !pte, !pfn) qui facilitent l'examen de la mémoire.

```
# 5d7f030 00 00 00 00 00 00 00 00-00 00 00 00 f0 00 00 00 ................
# 5d7f040 0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68 ........!..L.!Th
# 5d7f050 69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f is program canno
```

On retrouve effectivement l'adresse physique de la table de pages (0x8693000) ; la première entrée pointe sur l'endroit en mémoire physique contenant l'exécutable. Notons ci-dessus à l'adresse 0x85f5800 (en virtuel 0x800000000) le début des pages du noyau NT.

Trois nouveaux modes de pagination ont été ajoutés par Intel dans les versions successives de ses processeurs :

- Le mode Page Size Extensions (PSE) introduit avec le processeur Pentium ajoute la création de pages de 4 Mo. Les noyaux Linux et NT utilisent notamment ce mode pour éviter d'encombrer le cache de translations (TLB) lors de changements de contexte.

- Le mode Physical Address Extension (PAE) introduit avec le Pentium Pro permet d'adresser jusqu'à 64 Go de mémoire physique à l'aide d'un niveau supplémentaire de translation ; la taille des entrées est étendue à 64 bits.

- Le mode PSE-36 étend le mode PSE pour adresser de même jusqu'à 64 Go ; il est plus simple d'implémentation que le PAE mais reste peu utilisé en pratique.

En 2003, AMD élargit le mode PAE avec un niveau supplémentaire de tables de pages ; de plus le bit 63 d'une page définit si le processeur a le droit d'y exécuter du code (protection généralement connue sous le nom de NX ou XD). Sous Linux, le programme kmaps.c, écrit par paxguy1 [12] offre la possibilité d'inspecter visuellement la hiérarchie de tables de pages en mode PAE via le périphérique /dev/mem.

On constate donc que l'écriture directe en mémoire physique pose le problème de la translation inverse des adresses mémoires : pour un processus donné, il faut retrouver le répertoire principal puis descendre la hiérarchie de tables de page. Ce problème est contournable par l'emploi de code relogeable en mémoire, ie. ne dépendant pas d'adresses virtuelles fixes.

2.2 Programmation matérielle

Le composant de base en électronique digitale est le transistor. On peut le considérer comme une valve électrique : une tension de contrôle appliquée sur la grille

d'un transistor à effet de champ module le passage du courant entre le drain et la source.

Dans un circuit digital, les transistors ont un rôle de commutateur entre les deux niveaux de logique Booléenne. Si l'on considère le cas du PCI classique (Vcc = 3.3V), la convention est la suivante : le niveau logique 0 correspond au voltage -0.5V à Vcc*0,3 = 1V, et le niveau logique 1 correspond au voltage Vcc*0,5 = 1.65V à Vcc+0,5 = 3.8V.

L'interconnexion de transistors permet de construire des composants logiques de base (inverseur, porte non-ET, porte non-OR) à partir desquels on déduit deux grandes catégories de circuits :

- Circuits combinatoires : la sortie est une expression logique des valeurs d'entrées (additionneurs, multiplexeurs, décodeurs, etc.) ;
- Circuits à mémoire : bascule ou flip-flop contenant un bit d'information mis à jour lorsqu'une condition est remplie, par exemple le front montant d'un signal d'horloge. Un groupe de bascules est appelé registre.

Le schéma 2 montre un inverseur. Si l'entrée vaut Vcc, le transistor t1 (à canal P) est bloqué et t2 (à canal N) est saturé, donc la sortie est à la masse (0). Inversement si l'entrée est à la masse, la sortie aura comme valeur Vcc.

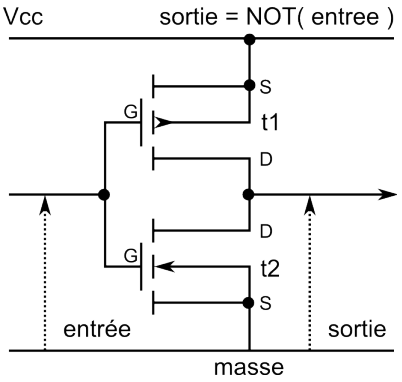

Fig. 2. Inverseur logique à base de transistors à effet de champ

Les circuits digitaux courants peuvent alors être vus comme un assemblage de bascules reliées par de la logique combinatoire, le contenu de ces bascules se mettant à jour à chaque cycle d'horloge.

Un signal électrique est dit actif-haut s'il est actif lorsqu'il est compris dans la gamme de voltage du niveau logique 1. Il est dit actif-bas dans le cas inverse (niveau 0). Un bus étant constitué de lignes électriques partagées, il n'est pas souhaitable de forcer la valeur d'un signal en permanence (ce qui bloquerait l'utilisation du bus). On parle donc de logique à trois états : soit le périphérique contrôle la ligne électrique à 0 ou 1, soit il n'impose pas de valeur ; le signal est dit de haute impédance ou flottant. En VHDL la valeur correspondante est 'Z'.

Si aucun circuit ne contrôle une ligne électrique, le signal pourrait donc fluctuer aléatoirement entre 0 et 1, engendrant des effets indésirables. On rajoute donc des résistances pour forcer une valeur à 0 (pull-down) ou à 1 (pull-up) en cas de flottement. Ces valeurs sont dites faibles car elles n'empêchent pas un circuit d'outrepasser la résistance afin de contrôler le signal.

Un dernier point important concerne le délai de propagation des signaux électriques. Plus la complexité d'un circuit combinatoire est élevée, plus le délai de mise à jour des signaux de sortie est important, ce qui limite la fréquence de l'horloge. Par exemple, supposons qu'un circuit exécutant une multiplication requiert 20 ns entre une nouvelle entrée et la stabilisation du signal de sortie. La fréquence d'horloge ne doit donc pas dépasser pour ce circuit 50 MHz.

2.3 La norme PCI et le bus-mastering

Le bus local PCI (Peripheral Component Interconnect) permet le partage entre plusieurs périphériques de lignes électriques vers le contrôleur d'entrées / sorties, communément appelé southbridge. Ce dernier est relié au contrôleur mémoire, dit northbridge, qui assure l'interface avec le processeur et la mémoire DRAM. A noter que les processeurs x86 récents (AMD64, Intel Nehalem) intègrent désormais le contrôleur mémoire pour des raisons de performance.

La norme définissant PCI a été publiée par Intel en 1992. Plusieurs versions successives ont par la suite été diffusées mais n'apportent pas de changements fondamentaux au fonctionnement du bus. Cette norme est payante ; en pratique, une recherche Google (comme par exemple « PCI specification ») fournit de nombreux résultats.

Présenter de manière complète la spécification PCI ne serait pas pertinent dans le cadre de cet article ; nous décrirons simplement le fonctionnement de lectures et écritures dans la mémoire physique initiées par un périphérique connecté au bus.

Les lignes électriques suivantes sont propres à chaque périphérique et le relient directement au contrôleur :

- PCI_REQ : actif bas, mis à 0 lorsque le périphérique initiateur demande le contrôle du bus ;
- PCI_GNT : actif bas, mis à 0 lorsque le contrôleur d'entrées/sorties autorise l'accès au bus ;

PCI_REQ et PCI_GNT sont uniques : chaque périphérique dispose de ses propres lignes. Autrement, le contrôleur ne pourrait pas distinguer les périphériques connectés et jouer son rôle d'arbitre vis-à-vis de demandes de bus mastering.

Les lignes suivantes sont partagées par l'ensemble des périphériques :

- PCI_CLK : horloge, signal périodique de fréquence 33 MHz ;
- PCI_RST : actif bas, mis à 0 lors d'une réinitialisation de l'ordinateur ;
- PCI_CBE : contient d'abord la commande (par exemple : Memory Read), puis durant le ou les phase(s) de donnée(s) défini quels octets sont valides (Byte Enable, actif bas) ;
- PCI_DATA : 32 lignes qui contiennent d'abord l'adresse de l'accès, puis les données à transférer ;
- PCI_PAR : bit de parité devant être positionné correctement lors d'un transfert de données ;
- PCI_FRAME : actif bas, mis à 0 par l'initiateur tant qu'il souhaite envoyer/recevoir des données ; remis à 1 au dernier mot de 32-bit échangé ;
- PCI_DEVSEL : actif bas, mis à 0 quant la cible considère que l'adresse appartient à son domaine ;
- PCI_IRDY : actif bas, mis à 0 par l'initiateur (ici la carte PCI) lorsqu'il est prêt à envoyer ou recevoir les données ;
- PCI_TRDY : actif bas, mis à 0 par la cible (ici le contrôleur mémoire) dès qu'il est prêt à recevoir ou envoyer les données ;

Les autres lignes du bus sont utilisées pour la génération d'interruptions (INTx), les accès exclusifs (LOCK) et plusieurs cas spécifiques de gestion des erreurs qui ne

nous intéressent pas ici.

Le schéma 3 montre un exemple typique d'écriture dans la mémoire depuis un périphérique PCI. On appelle ce type de représentation une forme d'ondes (waveform) car il montre de façon simplifiée l'évolution de signaux oscillant au cours du temps. Les signaux en majuscule sont ceux contrôlés par le périphérique PCI (sorties) ; ceux en minuscule sont les entrées.

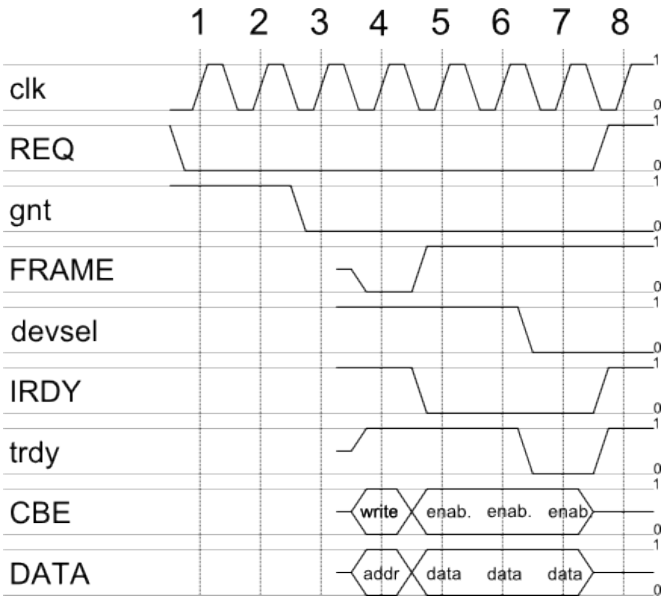

Fig. 3. Cycle typique d'écriture sur le bus PCI

- Cycle 1 : demande d'accès au bus, REQ passe à 0 ;
- Cycle 2 : attente de gnt ;
- Cycle 3 : accès au bus autorisé (gnt vaut 0) ;
- Cycle 4 : envoi de la commande (write = 0111) et de l'adresse sur 32 bits (les deux bits de poids faible doivent être à 0 pour l'alignement). Comme requis par le protocole, FRAME est mis à 0 ;
- Cycle 5 : IRDY passe à 0 pour indiquer que la donnée présente sur DATA est valide ; FRAME repasse à 1 pour indiquer qu'il s'agit du dernier mot à écrire.

Vérification de devsel et trdy : ils sont à 1, donc l'écriture n'a pas lieu au cycle 5 ;
- Cycle 6 : trdy est toujours à 1, l'écriture n'a pas lieu ;
- Cycle 7 : devsel et trdy sont positionnés à 0 par la cible. Le transfert de donnée est achevé ;
- Cycle 8 : REQ et IRDY passent à 1 pour signaler que l'accès au bus est relâché.

Il est possible qu'aucune cible ne réponde à la demande de lecture ou d'écriture : si devsel ne passe pas à 0 au bout de N cycles, la transaction a échoué et l'accès au bus doit être relâché pour un minimum de deux cycles.

L'accès en lecture est symétrique, à la différence près que le périphérique PCI lit le contenu de DATA, qui contient alors des données valides lorsque devsel et irdy sont simultanément à 0.

Les informations précédentes s'appliquent de même aux cartes au format PC Card (autrement appelées cartes PCMCIA) qui communiquent avec le protocole 32-bit Cardbus, très proche de PCI.

3 Travaux pratiques

3.1 Matériel et logiciels utilisés

Nous utiliserons ici un circuit FPGA reprogrammable contenant quatre catégories d'éléments de bases : LUTs (Look-Up Table, tables de vérité à 4 bits), flip-flops, ressources de routage et entrées / sorties. Deux signaux sont distribués globalement à l'ensemble des flip-flops : l'horloge (clock) et la réinitialisation (reset). Ont été achetées les cartes suivantes :

- La carte PCI Raggedstone1 est vendue par Enterpoint ; elle dispose d'un FPGA Spartan3 XC3S400, d'une mémoire flash de 512 Ko et d'un afficheur 4x8 segments.

- La carte COM-1300-C (option Cardbus) est vendue par ComBlock ; elle dispose d'un XC3S400 et d'une mémoire flash pour le stockage du fichier de configuration du FPGA.

Des problèmes avec ces cartes ont été relevés :

– Un bug dans la configuration par défaut du FPGA de Raggedstone1 empêche certains ordinateurs récents de démarrer : le BIOS ne s'initialise pas, ou provoque parfois un crash de Windows (écran bleu). Le constructeur n'ayant pas pu fournir de correctif, la solution retenue a consisté à reprogrammer la carte avec le bitstream du projet open-source rs1_7seg_pci ;

– Un bug dans le driver Cardbus fourni avec la carte COM-1300 empêche de reprogrammer cette dernière sur les ordinateurs portables dotés d'un processeur double-cœur ; l'écriture de la mémoire flash échoue avec le message d'erreur Memory Program Failed. La seule solution que nous avons trouvée revient à utiliser un ancien portable (ThinkPad X22) pour programmer la carte.

Xilinx ISE en version 8.2.03i a servi pour les phases de développement, et ModelSim 6.1e pour la simulation. Un fois le design implémenté, il est transféré en mémoire flash soit par un câble parallèle relié au port JTAG dans le cas de la carte PCI, ou encore avec le logiciel de ComBlock qui fait transiter les données de configuration du FPGA (fichier mcs) par l'interface Cardbus. Notons que pour réinitialiser la carte dans sa configuration d'origine il faut relier les pins A20 et B20 du port d'extension.

3.2 Ecriture dans la mémoire physique

Le langage de VHDL offre un niveau d'abstraction plus élevé que le câblage manuel d'éléments de base évoqués dans la partie rappels. Le compilateur synthétise à partir de la description de haut niveau la configuration des éléments du FPGA (LUT, registres) et assure leur interconnexion.

Présentons maintenant un exemple de programme VHDL simple écrivant quatre octets dans la mémoire. On commence par préciser les entrées / sorties de ce composant :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY pci_toplevel is
PORT(

    PCI_CLK         : IN    std_logic;
    PCI_RST         : IN    std_logic;
```

```
        PCI_CBE         : INOUT std_logic_vector(  3 downto 0 );
        PCI_DATA        : INOUT std_logic_vector( 31 downto 0 );
        PCI_PAR         : INOUT std_logic;

        PCI_REQ         : OUT   std_logic;
        PCI_GNT         : IN    std_logic;
        PCI_IRDY        : INOUT std_logic;
        PCI_TRDY        : INOUT std_logic;
        PCI_FRAME       : INOUT std_logic;

        -- autres signaux PCI inutilises( non affiches ici )
    );
    END pci_toplevel;
```

Ensuite sont définis les signaux internes au composant. Deux signaux de type enable sont présents, afin de laisser les signaux PCI de contrôle et de données dans l'état flottant lorsque la carte n'a pas la maîtrise du bus.

```
    ARCHITECTURE arch of pci_toplevel is

        TYPE fsm_state is ( ST_INIT, ST_REQ, ST_DATA, ST_IDLE );

        SIGNAL s_state          : fsm_state := ST_INIT;
        SIGNAL s_count          : natural range 0 to 1024 := 0;

        SIGNAL en_PCI_CTRL      : std_logic;
        SIGNAL en_PCI_DATA      : std_logic;
        SIGNAL so_PCI_CBE       : std_logic_vector(  3 downto 0 );
        SIGNAL so_PCI_DATA      : std_logic_vector( 31 downto 0 );
        SIGNAL so_PCI_PAR       : std_logic;
        SIGNAL so_PCI_FRAME     : std_logic;
        SIGNAL so_PCI_IRDY      : std_logic;
```

Le code proprement dit suit ; il s'agit d'une machine à état peu complexe :

- INIT : le FPGA attend un certain nombre de coups d'horloge pour être assuré que le bus est dans un état stable suite à l'insertion de la carte ;
- REQ : le FPGA demande le contrôle du bus. Dès que le contrôleur accepte, l'adresse est envoyée ;
- DATA : la donnée à écrire est positionnée sur le bus jusqu'à ce que la cible déclare être prête ou que le timeout expire ;
- IDLE : l'accès au bus est relâché.

```
    p_ext_state : PROCESS( PCI_RST, PCI_CLK, PCI_GNT, s_state, s_count )
    BEGIN
```

```
        IF( PCI_RST = '0' ) then

            s_state <= ST_INIT;
            s_count <= 0;

        ELSIF( rising_edge( PCI_CLK ) ) then

            CASE s_state is

                WHEN ST_INIT =>

                    PCI_REQ         <= '1';
                    en_PCI_CTRL     <= '0';
                    en_PCI_DATA     <= '0';

                    s_count <= s_count + 1;
                    IF( s_count = 4 ) then
                        s_state     <= ST_REQ;
                        s_count     <= 0;
                    END if;

                WHEN ST_REQ =>

                    PCI_REQ      <= '0';            -- demande d'acces au bus

                    IF( PCI_GNT = '0' ) then        -- demande acceptee

                        en_PCI_CTRL     <= '1';
                        en_PCI_DATA     <= '1'; -- ecriture en memoire a 0x60204
                        so_PCI_DATA     <= "00000000000001100000001000000100";
                        so_PCI_CBE      <= "0111";
                        so_PCI_FRAME    <= '0';
                        so_PCI_IRDY     <= '1';
                        s_state         <= ST_DATA;

                    ELSE

                        en_PCI_CTRL     <= '0';
                        en_PCI_DATA     <= '0';
                        s_state         <= ST_REQ;

                    END if;

                WHEN ST_DATA =>

                    PCI_REQ         <= '0';
                    en_PCI_CTRL     <= '1';
                    en_PCI_DATA     <= '1'; -- donnee : 0xFF5500AB
                    so_PCI_DATA     <= "11111111010101010000000010101011";
                    so_PCI_CBE      <= "0000";
                    so_PCI_FRAME    <= '1';
                    so_PCI_IRDY     <= '0';

                    s_count <= s_count + 1;
                    IF( PCI_TRDY = '0' or s_count = 64 ) then
                        so_PCI_IRDY     <= '1';
                        s_state         <= ST_IDLE;
```

```
                    END if;

                WHEN ST_IDLE =>

                    PCI_REQ         <= '1';
                    en_PCI_CTRL     <= '0';
                    en_PCI_DATA     <= '0';

                WHEN others =>

                    s_state         <= ST_IDLE;

            END case;

        END if;

    END process;
```

En parallèle au processus de mise à jour de la machine à état, sont définies les valeurs des signaux de sortie :

```
    PCI_CBE   <= so_PCI_CBE   WHEN( en_PCI_DATA = '1' ) else ( others => 'Z' );
    PCI_DATA  <= so_PCI_DATA  WHEN( en_PCI_DATA = '1' ) else ( others => 'Z' );
    PCI_PAR   <= so_PCI_PAR   WHEN( en_PCI_DATA = '1' ) else 'Z';
    PCI_IRDY  <= so_PCI_IRDY  WHEN( en_PCI_CTRL = '1' ) else 'Z';
    PCI_FRAME <= so_PCI_FRAME WHEN( en_PCI_CTRL = '1' ) else 'Z';
```

Ce programme met en valeur deux points importants :

- L'utilisation de machines à état (parfois appelées automates finis) qui est fondamentale en programmation de matériel ;

- La valeur des signaux se met à jour au coup d'horloge suivant. Par exemple, si la condition de test IF(PCI_TRDY = 0) est vraie, l'état des variables de sortie so_PCI_IRDY et so_PCI_FRAME ne vaudra 1 qu'au prochain cycle. Il faut donc garder à l'esprit ce décalage permanent, qui n'est pas tangible dans les langages de programmation séquentiels classiques.

Une fois le code synthétisé, il est possible de visualiser le schéma RTL de l'implémentation faite par ISE. On voit sur la figure 4 une petite partie du circuit : est représenté un additionneur sur 10 bits, un comparateur et un registre, en l'occurrence s_count. Ensuite, la simulation permet de s'assurer du bon comportement du programme par l'inspection des sorties (figure 5). Ici les signaux d'entrée ont été définis à la main ; un design plus complexe aurait nécessité la création en VHDL d'un banc d'essai de génération d'entrées.

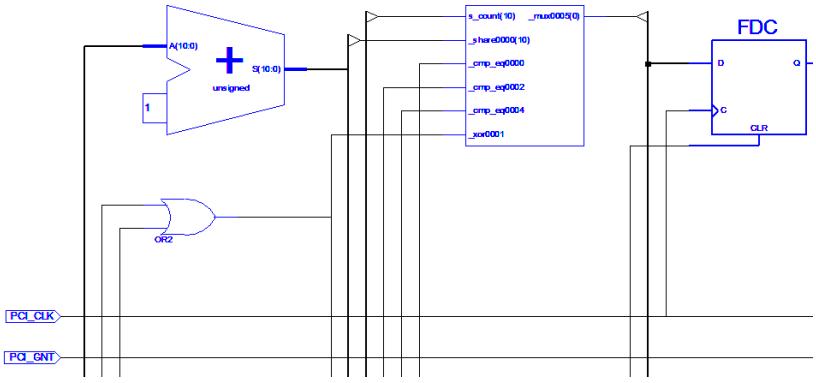

Fig. 4. Détail du schéma RTL de l'implémentation d'une écriture PCI

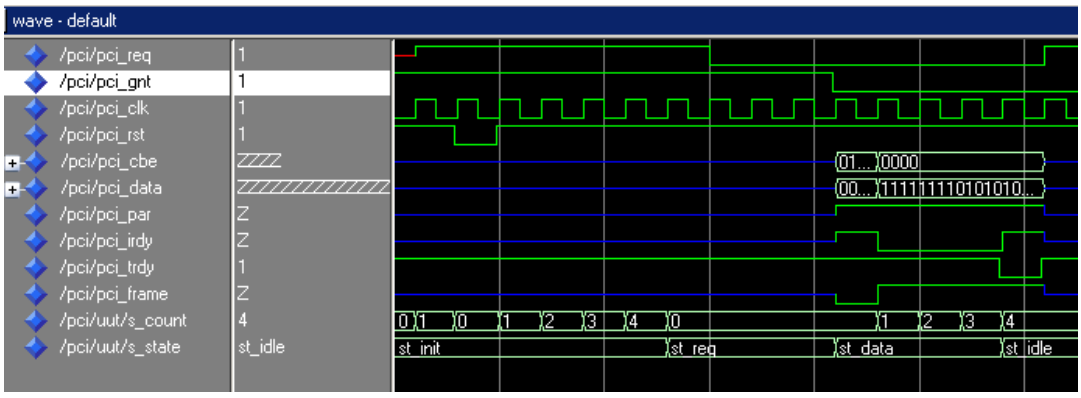

Fig. 5. Forme d'onde générée par le simulateur ModelSim

Au code VHDL de plus haut niveau est associé un fichier de contraintes fourni par le constructeur, et qui réunit chaque signal d'entrée / sortie à l'emplacement physique correspondant sur le FPGA. Des contraintes supplémentaires (timing, niveaux de voltage, etc.) peuvent être définies. Voici un extrait du fichier de contraintes relatif à la carte COM-1300-C :

```
NET "PCI_REQ"       LOC = "N1"  | IOSTANDARD = PCI33_3 | S;
NET "PCI_GNT"       LOC = "R12" | IOSTANDARD = PCI33_3 | S;

NET "PCI_RST"       LOC = "H1"  | IOSTANDARD = PCI33_3 | S;
NET "PCI_CLK"       LOC = "N8"  | IOSTANDARD = PCI33_3 | S
                                | TNM_NET = "PCI_CLK";

TIMESPEC "TS_PCI_CLK" = PERIOD "PCI_CLK" 30 ns HIGH 50 %;

NET "PCI_CBE<0>"    LOC = "T12" | IOSTANDARD = PCI33_3 | S;
NET "PCI_CBE<1>"    LOC = "R13" | IOSTANDARD = PCI33_3 | S;
NET "PCI_CBE<2>"    LOC = "R11" | IOSTANDARD = PCI33_3 | S;
NET "PCI_CBE<3>"    LOC = "N2"  | IOSTANDARD = PCI33_3 | S;
```

Des tests ont ensuite été faits sur un autre ordinateur, les crashs dus à une mauvaise manipulation du bus PCI étant fréquents. Quelques problèmes notables ont été rencontrés :

– Il est nécessaire d'ajouter la contrainte PULLUP au pin relatif au signal PCI_FRAME ; autrement, l'insertion de la carte sur un portable Lenovo R60 gèle le système (clavier et souris bloqués, pas de mise à jour de l'écran), le système redevenant fonctionnel après éjection de la carte ;

– Les pins non utilisés doivent être configurés en mode flottant dans le menu des propriétés de la tâche Generate Programming File. Sans cela, aucune écriture n'a lieu.

Finalement, on vérifie avec un éditeur hexadécimal comme WinHex que l'écriture est effective (figure 6). On note que la valeur apparaît immédiatement après insertion de la carte. L'attaque est plus furtive que dans le cas du FireWire, qui nécessite un traitement par le driver correspondant du système ; ici, la carte est invisible et aucun message n'est affiché par Windows après l'insertion.

3.3 Implémentation de l'attaque winlockpwn

La fonction MsvpPasswordValidate de la librarie MSV1_0.dll est appelée lors de la vérification du mot de passe d'un l'utilisateur local. La valeur retournée par la

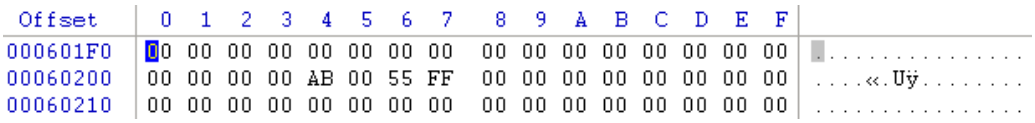

Fig. 6. Examen de la mémoire physique après insertion de la carte

fonction de hachage est comparée à celle stockée dans la base locale SAM.

L'attaque consiste à remplacer dans le code ci-dessous le saut conditionnel jnz par des instructions sans effet (nop) de manière à ce que la fonction retourne toujours vrai.

```
.text:77C798A4                  call    ds:RtlCompareMemory
.text:77C798AA                  cmp     eax, 10h
.text:77C798AD                  jnz     short loc_77C798C0
.text:77C798AF
.text:77C798AF loc_77C798AF:            ; CODE XREF: LsaApLogonTermin...
.text:77C798AF                          ; LsaApLogonTerminatedge+3E1
.text:77C798AF                  mov     al, 1
```

Le code VHDL précédent a été modifié afin d'itérer par sauts de 4096 octets sur les quatre Go de mémoire physique : il cherche la signature 107511B0 à l'adresse courante de la page plus 0x18AC (offset pour Windows XP SP3 anglais) et le remplace par 109090B0. Pour cela, deux registres supplémentaires contenant respectivement l'adresse courante et le mot lu en mémoire ont été ajoutés ainsi que trois nouveaux états (ST_REQ_READ, ST_DATA_READ et ST_CMP).

Le programme pourrait être assez facilement modifié pour prendre en compte d'autres niveaux de Service Pack ou versions de Windows, via l'ajout d'un registre codant l'index du système ainsi qu'une petite mémoire de type BlockRAM stockant offset, contenu cherché et modification à appliquer.

3.4 Injection de code dans le système d'exploitation

La programmation en langage VHDL peut s'avérer difficile, notamment lors de la création de machines à état compliquées. C'est pourquoi il a été décidé de concevoir et d'implémenter un processeur au sein du FPGA, qui exécutera le code d'altération de la RAM de l'ordinateur depuis un bloc de mémoire interne (dit BlockRAM).

Notre processeur fait maison était initialement basé sur DLX, qui présentait l'avantage d'un jeu d'instructions RISC réduit, assez proche de l'architecture MIPS originelle. En fin de compte, le processeur cowgirl [13] a servi de modèle car plus simple et disposant d'exemples de codes sources en VHDL.

Afin de faciliter les interactions avec le bus PCI, le processeur travaille avec des mots de 32 bits stockés dans un ensemble de 256 registres au total (R0 à R255). En pratiques seule une petite dizaine de registres est utilisée, sachant que R0 est câblé à la valeur 0. S'y ajoute un registre conditionnel CC contenant trois bits de drapeaux :

- Le bit zero est positionné à 1 si le résultat de la précédente opération est égal à zéro ; d'autre part il est mis à 1 si la lecture ou l'écriture sur le bus PCI échoue, par exemple dans le cas ou l'adresse est invalide ;

- Le bit négatif est positionné à 1 si le résultat de la précédente opération est négatif ;

- Le bit vuln est positionné à 1 si le résultat de l'opération induit un dépassement de capacité.

Enfin est introduit un registre nommé PC (Program Counter) qui contient l'adresse de l'instruction suivante dans la mémoire interne au FPGA.

Le tableau 1 liste l'ensemble des codes d'opération (opcodes) et leur équivalent en pseudo-assembleur (ici, ld qui est un déplacement immédiat sur 8 bits), et le schéma 7 montre le format machine des instructions de type 1 (standard) et de type 2 (avec constante d'adressage sur 32 bits).

Une partie du processeur après synthèse est présenté figure 8 ; on voit qu'il serait fastidieux de tout câbler à la main. Notons que le logiciel ISE permet de basculer la visualisation au niveau des composants individuels du CPU : unité arithmétique (ALU), registres, RAM, PCI ainsi que la logique de décodage et d'exécution des instructions.

A titre d'exemple, exposons maintenant le fonctionnement de la fonction d'addition de l'unité de traitements arithmétiques :

```
    PRODURE add( result : inout bit_32;
                 op1, op2 : in integer;
```

Tab. 1. Jeu d'instructions implémentées dans le processeur

Opérateur	**Opcode (hex.)**	**Opérande(s)**	**Destination**
add	0	rA, rB	rC
sub	1	rA, rB	rC
mul	2	rA, rB	rC
div	3	rA, rB	rC
cmp	4	rA, rB	rC
or	5	rA, rB	rC
xor	6	rA, rB	rC
mask	7	rA, rB	rC
and	8	rA, rB	rC
addq (add quick)	10	rA, ld	rC
subq (sub quick)	11	rA, ld	rC
mulq	12	rA, ld	rC
divq	13	rA, ld	rC
mdr (load from bram)	20	rA + addr32	rC
str (store to bram)	21	rC	rA + addr32
mdp (load from pci)	28	rA + addr32	rC
stp (store to pci)	29	rC	rA + addr32
ldq (load quick)	30	rA+ld	rC
stq (store quick)	31	rC	rA+ld
br (branch)	40	PC + addr32	PC
bri (branch indexé)	41	rA + addr32	PC
brq	50	PC+lb	PC
brz (branch if zero)	42	rA + addr32	PC
bnz (branch if not zero)	43	rA + addr32	PC
shl (décalage à gauche)	60	rA, rB	rC
shr (décalage à droite)	61	rA, rB	rC

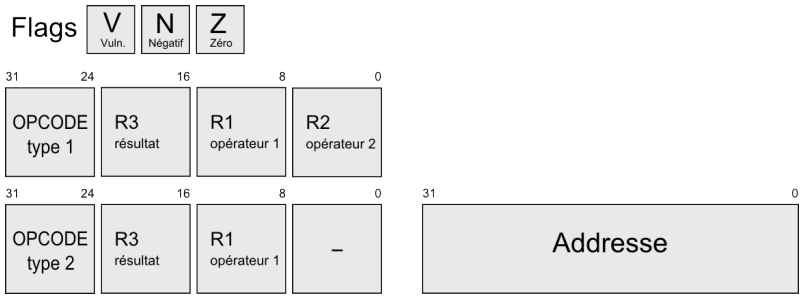

Fig. 7. Format des instruction du processeur

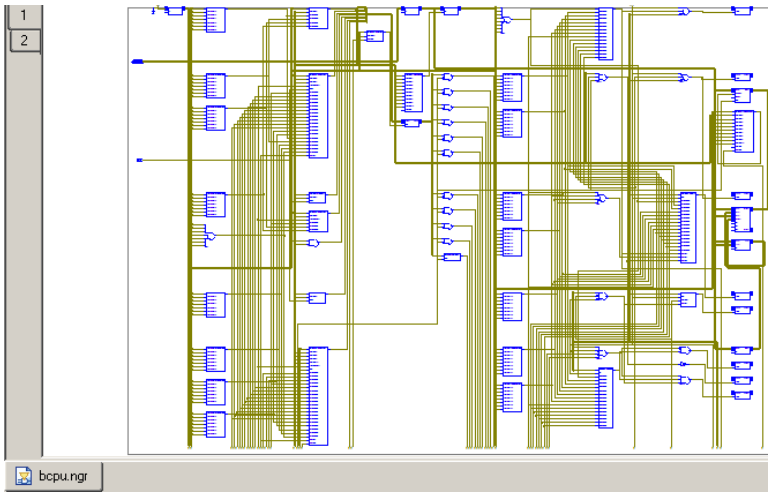

Fig. 8. Schéma RTL du processeur après synthèse par ISE

```
                V, N, Z : out bit) is
BEGIN
    IF op2 > 0 and op1 > integer'high-op2 then  -- depassement
        int_to_bits(((integer'low+op1)+op2)-integer'high-1, result);
        V := '1';
    ELSIF op2 < 0 and op1 < integer'low-op2 then -- depassement
        int_to_bits(((integer'high+op1)+op2)-integer'low+1, result);
        V := '1';
    ELSE              int_to_bits(op1 + op2, result);
        V := '0';
    END if;
    N := result(31);
    Z := bool_to_bit(result = X"0000_0000");
END add;
```

Comme décrit dans le tableau 1, la fonction add prend en entrée deux nombres entiers ici appelés op1 et op2 et renvoie en sortie result ainsi que trois bits (V, N, Z).

La mémoire BlockRAM interne du FPGA offre une capacité de 288 Kbits (36 Ko). Cet espace mémoire est partitionné de la manière suivante :

– 0x0000 – 0x0BFF : code exécutable pour le processeur interne ;

– 0x0C00 – 0x0FFF : données relatives aux systèmes d'exploitations : pour chaque système, offset et signature à modifier ;

– 0x1000 – 0x87FF : code x86 à écrire dans la mémoire de l'ordinateur cible.

Deux approches sont possibles pour amener le système d'exploitation à exécuter le code injecté : modifier un pointeur de fonction vers la page mémoire dans laquelle se trouve le code, ou encore modifier des octets du code du processus pour insérer une instruction de saut (inline hook) ; c'est cette seconde solution qui a été retenue.

Parmi l'ensemble des processus effectuant des tâches à intervalle régulier, winlogon est particulièrement intéressant : en outre de disposer du niveau de privilèges SYSTEM, la fonction SASWndProc est appelée lorsque l'utilisateur presse une combinaison de touches comme Ctrl-Alt-Del (hotkeys) ; elle est aussi appelée lors de la connexion et déconnexion d'un périphérique (souris USB, ...). Son fonctionnement a été analysé par eEye [14] ; quelques obscurs sites chinois y font de même référence.

Le FPGA va alors insérer au début de cette fonction un call vers l'emplacement du code injecté ; cette technique s'inspire en partie du programme BootRoot [15]. N'importe quelle zone inutilisée dans une page marquée comme exécutable de winlogon conviendrait ; nous avons choisi d'écrire le code injecté à l'adresse 0x1071970, ce qui fournit un espace d'environ 5000 octets comme on peut le constater ci-après :

```
0:018> !address
[...]
    01000000 : 01000000 - 00001000
                    Type     01000000 MEM_IMAGE
                    Protect  00000002 PAGE_READONLY
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageImage
                    FullPath C:\WINDOWS\system32\winlogon.exe
                01001000 - 00071000
                    Type     01000000 MEM_IMAGE
                    Protect  00000020 PAGE_EXECUTE_READ
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageImage
                    FullPath C:\WINDOWS\system32\winlogon.exe

0:018> db 01071940 L60
01071940  00 00 00 00 00 00 00 00-00 00 00 00 52 53 44 53  ............RSDS
01071950  7e 38 b7 83 71 5c ed 43-87 41 b9 7d 3b fb 21 75  ~8..q\.C.A.};.!u
01071960  02 00 00 00 77 69 6e 6c-6f 67 6f 6e 2e 50 44 42  ....winlogon.PDB
01071970  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
01071980  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
01071990  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................

0:000> eb 01071980 cc
0:000> g
(42c.430): Break instruction exception - code 80000003 (first chance)
eax=7ffdd000 ebx=00000000 ecx=00000000 edx=0006fad4 esi=010253e6 edi=0006fae0
eip=01071980 esp=0006fa7c ebp=0006faa4 iopl=0         nv up ei pl nz na po nc
```

```
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
winlogon!_NULL_IMPORT_DESCRIPTOR+0x3238:
01071980 cc              int     3
```

La version actuelle du code ne fait rien de remarquable : les protections de la page contenant le hook inline sont modifiées pour restaurer le code d'origine, puis est fait un appel à la fonction MessageBoxA. Dans un second temps, il est prévu d'écrire un code d'exploitation plus complexe qui allouerait une certaine quantité de mémoire, puis y placerait un marqueur afin que le FPGA trouve à quel endroit écrire le code x86 contenant la charge finale. Nous envisageons également de porter l'attaque sur les versions plus récentes de Windows ainsi que Linux.

4 Conclusion et perspectives

Nous avons montré dans cet article en quoi le protocole PCI tel qu'implémenté au sein de l'architecture x86 constitue une faille de sécurité susceptible de faciliter la compromission d'un ordinateur. Cependant des contre-mesures existent :

- Les processeurs AMD64 et Core2 avec les extensions de virtualisation (VT-x) disposent d'une IOMMU capable de filtrer les entrées / sorties des périphériques connectés. Le support de l'IOMMU a été ajouté dans le noyau Linux en version 2.6.24 ; il est activé avec les options CONFIG_DMAR et CONFIG_DMAR_ON. Néanmoins plusieurs distributions GNU/Linux comme Debian n'ont pas activé cette option par souci de compatibilité.

- Comme montré par Joanna Rutkowska [16], il serait possible de configurer le pont HyperTransport d'un processeur AMD64 pour provoquer un déni en service en cas de tentative d'acquisition de la mémoire depuis le bus PCI ;

- Une autre possibilité, sur une machine dotée d'un processeur x86_64 et disposant de plus de 4 Go de mémoire, est de configurer le noyau Linux pour réserver les quatre premiers Go de RAM aux accès DMA, et stocker les pages de code et données au dessus de cette frontière. Les accès depuis le bus PCI étant limités à 32 bits d'adressage physique, l'attaque ne fonctionnerait plus.

- Finalement, en l'absence d'une IOMMU il est recommandé de désactiver complètement les ports Cardbus (ainsi que FireWire). La désactivation se fait sous Windows dans le gestionnaire de périphériques ; depuis Linux, il faut

décharger le module yenta_socket. Alternativement, il serait envisageable de rendre ces ports physiquement inutilisables, par exemple avec de la résine.

Les attaques récentes sur d'autres composants matériels (citons en particulier les travaux de Loïc Duflot [17] [18]) démontrent s'il en est encore besoin qu'un ordinateur ne doit pas être considéré comme un bloc indivisible mais comme un ensemble d'éléments capables d'être chacun compromis. Par ailleurs il a été montré que des clefs de chiffrement pouvaient être recouvrées depuis les barrettes de mémoire d'un ordinateur, même plusieurs minutes après leur extraction [19].

A cet égard, les consoles de jeux modernes possèdent une importante longueur d'avance en matière de sécurité matérielle ; ainsi chaque Xbox 360 intègre une clef unique dans le processeur utilisée par le chargeur en ROM pour déchiffrer le noyau. Il faut souhaiter que les technologies actuellement en cours de développement (telle que CryptoPage [20]) feront l'objet d'une implémentation dans de futures architectures de confiance.

Références

1. Cesare, S. : Runtime kernel kmem patching. (1998)
2. sd, devik : Linux on-the-fly kernel patching without LKM. Phrack **11**(58) (2001)
3. Lacombe, E., Raynal, F., Nicomette, V. : De l'invisibilité des rootkits : application sous Linux. In : Symposium sur la Sécurité des Technologies de l'Information et des Communications 2007, École Supérieure et d'Application des Transmissions (2007) 145–178
4. crazylord : Playing with Windows /dev/(k)mem. Phrack **11**(59) (2002)
5. Ionescu, A. : Subverting Windows 2003 SP1 Kernel Integrity Protection. In : Proceedings of REcon. (2006)
6. Carrier, B.D., Grand, J. : A hardware-based memory acquisition procedure for digital investigations. Digital Investigation **1**(1) (2004) 50–60
7. Becher, M., Dornseif, M., Klein, C.N. : FireWire : all your memory are belong to us. Proceedings of CanSecWest (2005)
8. Boileau, A. : Hit by a bus : Physical access attacks with Firewire. Proceedings of Ruxcon (2006)
9. Ruff, N., Suiche, M. : Enter Sandman (why you should never go to sleep). Proceedings of PacSec (2007)
10. Aumaitre, D. : Voyage au coeur de la mémoire. In : Symposium sur la Sécurité des Technologies de l'Information et des Communications 2008, École Supérieure et d'Application des Transmissions (2008) 378–437
11. Intel : Intel 64 and IA-32 Architectures Software Developer's Manuals (2009) http://www.intel.com/products/processor/manuals/.
12. paxguy1 : kmaps : simple page table walker based on /dev/mem (2008) http://www.grsecurity.net/~{}paxguy1/kmaps.c.

13. Gamroth, N.R. : Cowgirl CPU (2006) http://thebeekeeper.net/hw/cowgirl.html.
14. Soeder, D. : Reverse Engineering Case Study #1 : Windows System Hotkeys. eEye Industry Newsletter **1**(4) (2005)
15. Soeder, D., Permeh, R. : eEye BootRoot : A Basis for Bootstrap-Based Windows Kernel Code. Proceedings of BlackHat USA (2005)
16. Rutkowska, J. : Beyond the CPU : Defeating hardware based RAM acquisition. Proceedings of BlackHat DC (2007)
17. Duflot, L., Etiemble, D., Grumelard, O. : Utiliser les fonctionnalités des cartes mères ou des processeurs pour contourner les mécanismes de sécurité des systèmes d'exploitation. In : Symposium sur la Sécurité des Technologies de l'Information et des Communications 2006, École Supérieure et d'Application des Transmissions (2006) 210–227
18. Duflot, L., Levillain, O. : ACPI et routine de traitement de la SMI : des limites à l'informatique de confiance ? In : Symposium sur la Sécurité des Technologies de l'Information et des Communications 2009, École Supérieure et d'Application des Transmissions (2009)
19. Halderman, J., D.Schoen, S., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., J.Feldman, A., Appelbaum, J., W.Felten, E. : Lest we remember : Cold boot attacks on encryption keys. In : Proceedings of USENIX Security Symposium. (2008)
20. Brulebois, C., Duc, G., , Keryell, R. : Crypto-Page : une architecture efficace combinant chiffrement, intégrité mémoire et protection contre les fuites d'informations. In : Symposium sur la Sécurité des Technologies de l'Information et des Communications 2007, École Supérieure et d'Application des Transmissions (2007) 35–51