Quelle confiance dans les composants matériels ?

Loïc Duflot

DCSSI 51 bd. De la Tour Maubourg 75700 Paris Cedex 07 France

Résumé Nous présentons ici quelques unes des problématiques relatives à la sécurité des composants matériels. Nous montrons en particulier comment l'absence de modèle de sécurité au niveau matériel, la complexification des architectures matérielles et la multiplication des nouvelles technologies au sein des processeurs et des chipsets peuvent avoir un impact sur la sécurité d'une plateforme de confiance. Nous présentons par exemple comment un attaquant peut contourner toute fonction de sécurité implémentée au niveau logiciel en exploitant un bogue ou un piège d'un processeur. L'étude se concentre sur les microprocesseurs x86 (32 ou 64 bits) et les architectures matérielles associées.

Mots Clés : processeur, chipset, piégeage matériel, bogue matériel, x86.

1 Introduction

L'informatique de confiance repose sur différentes briques logicielles et matérielles à partir desquelles la confiance dans une plateforme peut être obtenue, presque par transitivité. Jusqu'à il y a quelques années, la communauté scientifique s'est essentiellement penchée sur la question de la confiance dans les composants logiciels. Différentes questions ont été posées telles que celles de savoir comment développer un composant logiciel sans bogue ou quel doit être le périmètre de tel ou tel composant logiciel. Ce n'est que très récemment que l'intérêt de la communauté pour les questions de confiance dans le matériel est apparu. Au delà de la confiance dans le Trusted Platform Module [25], c'est bien de la confiance dans les processeurs et les chipsets qu'il s'agit. Comment avoir confiance dans un système logiciel quel qu'il soit si les composants matériels mis en oeuvre par la plateforme sont piégés ou bogués ? Nous allons ici nous intéresser à cette problématique. Nous allons montrer comment l'absence de modèle de sécurité au niveau matériel a déjà par le passé pu être exploitable par d'éventuels attaquants, et comment la complexification des architectures matérielles rend de plus en plus difficile la formalisation d'un modèle de fonctionnement du matériel. Ensuite, nous étudierons les conséquences sur la sécurité des systèmes d'exploitation ou des moniteurs de machine virtuelles d'un bogue ou d'un piégeage dans un composant.

Nous avons fait le choix de concentrer notre étude sur les processeurs de la famille x86 (32 bits ou 64 bits), et sur les chipsets associés. Les processeurs Pentium®, Xeon®, Core Duo^{TM}, Athlon^{TM}, Turion^{TM} font partie de cette famille.

2 Quelques éléments d'architecture

La figure 1 présente l'architecture classique d'une plateforme disposant d'un processeur x86. Dans le principe (on verra plus loin que cette affirmation est aujourd'hui largement erronée), la majeure partie des composants logiciels s'exécute sur le processeur : code du système d'exploitation, des applications, des éventuels moniteurs de machines virtuelles, code de maintenance s'exécutant en mode "System Management" par exemple. Dans leur mode nominal de fonctionnement (mode protégé pour les processeurs 32 bits ou IA32-e pour les processeurs 64 bits), les processeurs disposent d'un mécanisme dit de privilège processeur (encore appelé CPL ou ring [19]). Les tâches les plus privilégiées (le système d'exploitation ou le moniteur de machine virtuelle selon les architectures logicielles) s'exécutent en ring 0 et disposent des privilèges maximaux au niveau matériel. Les applications utilisateur s'exécutent elles en ring 3 et ont donc des privilèges très restreints. Ce mécanisme de ring couplé aux mécanismes de segmentation et de pagination mis en œuvre au sein de l'unité de gestion de la mémoire du processeur (MMU) permet à un système d'exploitation d'isoler son propre espace mémoire de celui des autres applications potentiellement malicieuses, mais également d'isoler le contexte d'une application donnée de celui des autres applications.

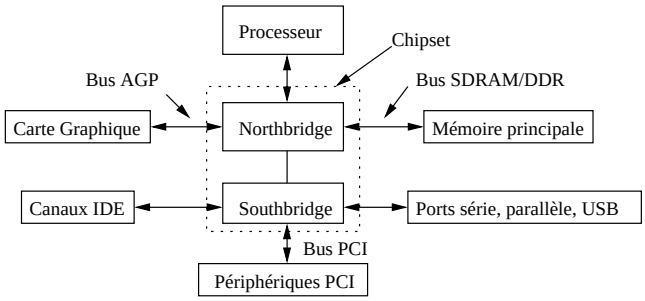

Fig. 1. Architecture simplifiée d'un système x86 (détail : Pentium® 4)

Les processeurs les plus récents munis des technologies AMD-V et VT-x [20] disposent de plus d'un mode de fonctionnement particulier permettant, lorsqu'un moniteur de machines virtuelles est mis en œuvre, d'isoler les composants de confiance des autres composants (systèmes d'exploitation invités par exemple).

Les processeurs x86 disposent enfin d'un mode de fonctionnement très privilégié appelé System Management Mode (SMM) [20] dans lequel vont s'exécuter un certain nombre de routines de traitement d'événements liés à la gestion d'alimentation de la plateforme indépendamment du système d'exploitation. Des exemples de tels événements peuvent être, en fonction des systèmes : connexion d'un réseau, passage à l'an 2000, connexion d'un périphérique USB, mise en veille

de la machine. L'emploi du mode System Management permet au concepteur de la machine de spécifier le comportement attendu du matériel en fonction des différents événements pouvant se produire, sans préjuger du système d'exploitation qui sera ensuite mis en œuvre.

3 Absence de modèle de sécurité matériel

Ce n'est que très tardivement que de réelles considérations de sécurité ont été prises en compte dans l'étude du mode de fonctionnement des composants matériels. Certes, ce sont bien des considérations de sécurité qui ont conduit à l'implémentation de mécanismes tels que le mécanisme de ring, de segmentation ou l'ajout de flag NX [4] ou XD [18] au mécanisme de pagination. Si donc les processeurs ont rapidement fourni aux systèmes d'exploitation modernes des mécanismes leur permettant a priori d'assurer leur sécurité au niveau logique, aucune étude de cohérence au niveau de la plateforme n'a jamais été menée. Des mécanismes de sécurité ont été petit à petit rajoutés à la demande au sein de différents composants sans qu'une étude de cohérence de l'ensemble n'ait été menée. Il n'y a donc aucune raison particulière pour que l'ensemble soit cohérent du point de vue de la sécurité.

L'exemple le plus patent est sans doute l'incohérence engendrée par le mécanisme dit "d'accès direct à la mémoire" (DMA). Pour des raisons de performances, il est possible pour les périphériques d'accéder directement à la mémoire sans aucun contrôle du processeur. Alors que les applications subissent un contrôle permanent effectué par le système d'exploitation, les périphériques eux, pouvaient historiquement lire ou modifier à volonté le contenu de l'ensemble de l'espace mémoire de la machine. Dès lors deux pistes d'attaques étaient possible pour tout attaquant souhaitant par exemple modifier le noyau du système d'exploitation ou retrouver des clefs de chiffrement stockées en mémoire :

- connecter un périphérique à la machine et initier depuis ce périphérique des accès DMA [14] ;
- reconfigurer logiquement, et sans accès physique à la machine, un contrôleur du chipset (par exemple le contrôleur USB [1]) pour que ce dernier réalise pour son compte des accès directs en mémoire.

Ces deux pistes d'attaque ont été prouvées exploitables. Les dernières générations de chipsets mettent cependant en œuvre des technologies dites VT-d ou I/OMMU [3] permettant de limiter les zones mémoire auxquels les périphériques ont accès via le mécanisme DMA. Cette réponse n'est que partielle, d'une part dans la mesure où ces technologies sont à l'heure actuelle assez peu utilisées, en partie car certains développeurs de périphériques et de pilotes ont pris de mauvaises habitudes qui empêchent leurs périphériques de fonctionner si la protection au niveau des accès DMA est activée, et d'autre part dans la mesure où cette réponse ne résout que le problème des accès DMA. Il a été prouvé à plusieurs reprises que des mécanismes de traduction d'adresse du chipset [1,27] pouvaient être exploités par un attaquant à des fins d'escalade de privilèges.

De même, le modèle de transition entre modes des processeurs a été assez peu étudié, et il a également été montré [16] que sous certaines conditions un attaquant pouvait générer des transitions non contrôlées vers le mode System Management lui permettant d'exécuter des commandes dans ce mode très privilégié et donc d'obtenir les privilèges maximums sur le système. Cette technique a depuis été reprise pour permettre à des virus de type rootkit de cacher certaines de leurs fonctions [5,17].

4 Conséquences de l'augmentation du nombre des fonctionnalités

Compte-tenu de ce qui précède, il aurait été raisonnable, pour les constructeurs de machines et de composants, de tenter d'établir un modèle de sécurité global de la machine de manière à corriger les problèmes de sécurité résiduels. Au lieu de cela, la tendance est aujourd'hui à la multiplication des fonctionnalités. Les processeurs sont aujourd'hui tous multi-cœurs, comportent des fonctionnalités cryptographiques avancées, supportent jusqu'à 5 modes de fonctionnement complètement différents et gèrent des registres spécifiques (en particulier les registres de type MSR Model Specific Registers [20]) différents d'une version à l'autre et pouvant avoir un impact significatif sur le sécurité de l'ensemble.

Pire, la complexité des chipsets croît de jour en jour. Un chipset actuel comporte bien entendu des contrôleurs mémoire et des contrôleurs de bus, mais aussi des contrôleurs de périphériques intégrés (carte réseau, contrôleur graphique) et plusieurs microprocesseurs génériques programmables. Ces derniers peuvent par exemple être utilisés pour émuler un composant de type TPM (Trusted Platform Module), effectuer des vérifications d'intégrité en mémoire (utilitaire Intel DeepWatch [6] présenté lors de la conférence Blackhat 2008), ou encore permettre l'administration du chipset à distance. C'est par exemple l'objet de la technologie Intel AMT [11] visant à permettre une administration d'un parc de machines à distance depuis un centre d'administration. Le centre d'administration se connecte directement à l'environnement local d'administration qui s'exécute entièrement dans le chipset (environnement de type web à base de technologies comme https et SOAP, qui ne sont pas nécessairement réputées pour leur fiabilité). Des technologies similaires existent pour la remontée d'alarme (technologie ASF [12] par exemple), qui permettent au chipset de prendre la main sur la carte réseau et de diffuser des messages d'alertes.

Globalement la vision de l'architecture présentée sur la figure 1 tend à disparaître pour être remplacée par une architecture où un seul composant, le chipset, gère la quasi totalité des périphériques, implémente des mécanismes de sécurité et de cloisonnement, et des fonctions d'administration et exécute des programmes logiciels très complexes, qui n'ont pas de raison d'être plus sécurisés que ceux qui s'exécutent sur le processeur, le tout sans modèle de sécurité apparent. Le processeur n'est plus vu que comme un coprocesseur du chipset qui gère les applications utilisateur, et sur lequel le chipset a un droit de regard et

de modification. Ce changement de philosophie explique l'intérêt grandissant de la communauté pour les questions de sécurité liées au chipset.

5 Conséquences d'un bogue ou d'un piégeage d'un processeur

Au vu de ce qui précède, il doit sembler évident que tout bogue ou piégeage d'un chipset peut avoir des conséquences dramatiques sur la sécurité de l'ensemble. Dans cette partie, nous nous concentrons maintenant sur l'étude des conséquences d'un bogue ou d'un piégeage des processeurs sur l'efficacité des mécanismes de sécurité matériels.

Lors de la conférence d'introduction aux journées C&ESAR 2007 [7], Adi Shamir a présenté l'impact sur la sécurité de l'implémentation logicielle de certains systèmes de chiffrement asymétrique, d'un bogue ou d'un piégeage de l'unité arithmétique et logique d'un microprocesseur x86 [19]. Cette présentation a ensuite été effectuée lors de la conférence Crytpo 2008. La presse grand public s'en est largement faite écho [24].

D'autres présentations ont récemment montré comment il était possible d'implémenter simplement des pièges exploitables dans les processeurs. On peut notamment citer la présentation de l'université de l'Illinois [22] qui a montré comment piéger de différentes façons un microcontrôleur de type Sparc et exploiter les pièges implémentés. Une autre présentation [13] a étudié l'ajout à un processeur ARM d'un mode semblable au mode SMM du Pentium permettant d'obtenir les privilèges maximums sur une machine.

5.1 Bogue, piégeage ou fonction non documentée ?

Les termes de bogue, piégeage ou fonctionnalité non documentée renvoient à trois notions intuitivement différentes. Un bogue correspond à une erreur non volontaire d'implémentation d'un composant qui se traduit par un dysfonctionnement dudit composant, incompatible avec les spécifications de ce dernier. La présence de bogues dans un composant matériel semble inévitable avec les méthodologies de développement actuelles, étant donnée la relative difficulté à corriger a posteriori un bogue matériel, malgré le soin extrême généralement apporté par les concepteurs.

On parle, en revanche, de fonctionnalité non documentée, lorsque le développeur a inclus dans son produit un ensemble de fonctionnalités dont il n'a pas précisé l'existence, la syntaxe ou la sémantique. Il est très fréquent pour un concepteur d'implémenter des fonctionnalités non documentées, en particulier pour ce qui est des fonctions de débogage ou d'administration. Il est courant, même si c'est intuitivement incorrect, de considérer que le secret de l'existence d'un mécanisme empêchera toute exploitation à des fins frauduleuses de ce dernier. A titre d'exemples, plusieurs fonctionnalités des processeurs x86 n'ont pas, à ce jour, été documentées. Les processeurs x86 possédaient en particulier une

instruction assembleur dite LOADALL[1] [8] qui permettait au code qui l'exécutait de charger en un cycle d'horloge l'ensemble des registres du processeur à partir d'une image en mémoire dont l'adresse était passée en paramètre à l'instruction. On imagine sans peine l'intérêt qu'une telle fonctionnalité peut avoir dans le domaine de l'analyse de fonctionnement du processeur ou du débogage, mais également quel usage un attaquant pourrait faire de cette fonctionnalité. Les processeurs x86 actuels possèdent également quelques fonctionnalités non documentées par les constructeurs, parmi lesquelles l'instruction assembleur "salc" et dont on trouve la signification par ailleurs [9].

Enfin, la notion de piégeage renvoie, elle, à une volonté manifeste, de la part du développeur ou d'une entité étant parvenu d'une manière ou d'une autre à s'immiscer dans le flot de conception du composant cible, d'introduire des fonctionnalités non documentées qui lui permettront a posteriori, lorsque le composant sera produit et distribué, d'effectuer des opérations les plus privilégiées possibles à l'insu de l'utilisateur légitime. On peut par exemple imaginer l'exemple paranoïaque d'un piégeage d'une carte réseau qui sur réception d'une trame IP particulière, passerait dans un mode actif permettant des accès arbitraires à distance en mémoire principale via le mécanisme DMA (Direct Memory Access [14]). Un autre exemple classique est celui d'une carte à puce piégée qui lorsqu'elle reçoit une donnée x, renvoie systématiquement le chiffré de x par une clef K sauf pour une valeur donnée de x pour laquelle elle renvoie K.

Bien que ces trois notions renvoient à des concepts différents l'on est malheureusement contraint, du point de vue d'une analyse sécurité, de considérer leur équivalence. En effet, un expert en sécurité qui devrait évaluer le niveau de sécurité d'un composant n'aurait connaissance ni des bogues spécifiques à ce composants (non connus du développeur), ni aux fonctions non documentées, et encore moins aux éventuels piégeages. Le principe de précaution lui impose en outre de considérer l'impact maximal de chacun des problèmes potentiels. Bien que dans la plupart des cas, les bogues puissent être anodins et non exploitables par un quelconque attaquant, dans le pire cas, un bogue peut être exploitable et permettre à l'attaquant de mettre en œuvre une escalade de privilèges sur le système. Il en va de même des fonctionnalités non documentées. Du point de vue d'une analyse sécurité, ces trois notions sont donc équivalentes.

Nous utiliserons donc dans la suite de ce document le terme piégeage pour désigner indifféremment un bogue, une fonction non documentée ou un piégeage en tant que tel, dès lors qu'ils sont exploitables par un attaquant.

5.2 Expérimentations

Afin d'étudier les conséquences d'un bogue ou d'un piégeage sur la sécurité des systèmes d'exploitation et des moniteurs de machines virtuelles, deux expérimentations ont été menées. La première (figure 2a) consistait à implémenter dans un processeur x86 un piège qui, lorsque le processeur se trouvait dans un état

[1] Cette instruction assembleur était toutefois a priori réservée au seul usage des composants s'exécutant en ring 0.

donné et exécutait une instruction particulière, donnait à la tâche courante les privilèges maximums (ring 0) sur le système. La seconde (figure 2b) implémentait un piégeage un peu plus compliqué qui fournissait en outre un moyen de contourner les mécanismes de segmentation et de pagination lorsque le piège était activé.

Pour le besoin des preuves de concept, l'émulateur libre Qemu a été modifié pour implémenter les pièges considérés.

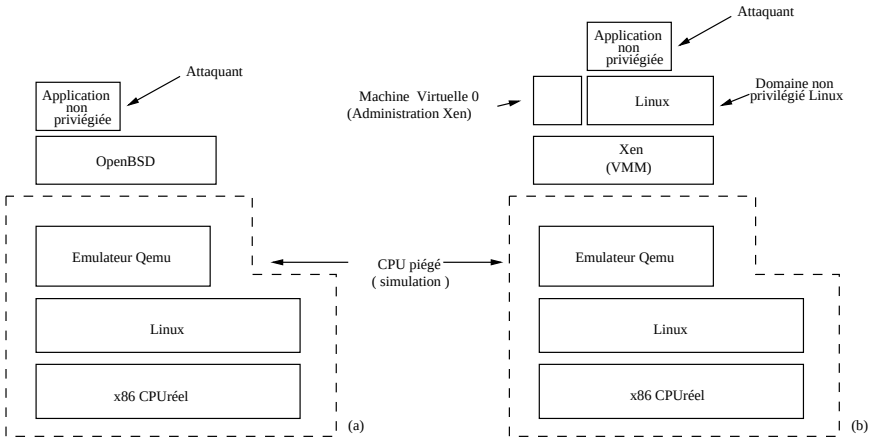

Fig. 2. Schéma du dispositif expérimental

5.3 Conclusions de l'étude

Il est apparu lors de l'étude (voir [15] pour une description plus détaillée) que le premier piégeage (figure 2a) permettait à un attaquant en ayant connaissance d'obtenir l'exécution de code arbitraire en couche noyau quel que soit le système d'exploitation mis en œuvre, depuis des privilèges très restreints (contexte d'une application non privilégiée). En revanche, il est apparu très difficile (aucun moyen satisfaisant n'a été mis en évidence jusqu'ici) d'exploiter un tel piégeage dans le cas général si un moniteur de machines virtuelles (Xen [26] par exemple) est utilisé. Le second piégeage (figure 2b) permet à l'attaquant de prendre la main sur n'importe quel système (exécuter du code en ring 0), qu'il mette en œuvre un moniteur de machines virtuelles ou non, quels que soient les mécanismes de sécurité mis en œuvre, et ce depuis des privilèges très réduits (application non privilégiée d'un système d'exploitation lui-même non privilégié).

Cette étude montre qu'il sera toujours possible à un attaquant d'intégrer dans un processeur un piégeage simple et générique qui lui permette a posteriori de prendre la main sur tous les systèmes qui mettent en œuvre ce composant dès lors qu'il est capable d'exécuter du code sur ce dernier. Cela prouve une fois encore le risque qu'il existe à faire tourner une application non maîtrisée sur une

machine même lorsque celle-ci ne bénéficie initialement que de privilèges très restreints.

5.4 Limiter l'impact des bogues et des piégeages

L'une des questions qui s'est rapidement posée est celle de la détection de tels piégeages. Une équipe de recherche a présenté [2] comment des techniques proches de celles mises en œuvre dans les attaques par canaux auxiliaires [23] pouvaient permettre de détecter des pièges intégrés en phase de fonderie dans les composants. L'efficacité pratique d'une telle méthode reste cependant à vérifier. Il est en particulier de notoriété publique que l'administration américaine, via l'agence DARPA (Defense Advanced Research Projects Agency), a lancé un grand projet de 3 ans (projet "Trust In Circuits") visant à déterminer la meilleure méthode pour détecter un piège au sein d'un composant.

Afin de limiter le risque d'exploitation d'un bogue ou d'un piège par un attaquant, il convient dans la mesure du possible que les concepteurs de système n'autorisent que des composants logiciels de confiance à s'exécuter sur la machine. Tout code non maîtrisé s'exécutant sur une machine a la possibilité d'exploiter d'éventuels bogues ou d'éventuels piégeages pour obtenir les privilèges maximaux sur une machine. La suppression des moyens de compilation ou d'exécution de code arbitraire sur une machine (macros) est également une bonne pratique.

5.5 Aspects pratiques

Il est intéressant de noter que les deux constructeurs principaux de processeurs x86 (Intel et AMD) publient régulièrement une liste [10] des bogues matériels de leurs processeurs. Cette liste est généralement relativement longue et certains bogues qui y sont répertoriés ne seront sans doute jamais corrigés, du fait de la difficulté de modifier a posteriori le fonctionnement d'un circuit microélectronique aussi complexe qu'un microprocesseur.

Un chercheur indépendant (Kris Kaspersky) a prévu de présenter [21] comment exploiter un certain nombre de ces bogues comme moyen d'escalade de privilège. Selon lui certains de ces bogues sont même exploitables à distance et ce quel que soit le système d'exploitation mis en œuvre. Ces allégations restent à l'heure où ces lignes sont écrites à vérifier mais la situation n'en est pas moins préoccupante.

6 Conclusion

En conclusion, nous avons ici abordé la question de la confiance dans les composants informatiques standards que sont les processeurs et les chipsets. Nous avons montré que cette confiance, pourtant nécessairement préalable à la mise en œuvre de tout système d'information sécurisé, était pourtant dans l'absolu difficilement atteignable. Les publications académiques et scientifiques qui démontrent les faiblesses des architectures matérielles actuelles sont de plus en

plus nombreuses et pertinentes. Face à ce constat, il est important de prendre en compte d'éventuels bogues ou piégeages des composants matériels lors de l'analyse de risque préalable à la conception de tout système destiné à la protection d'information sensibles.

Références

1. L. Absil and L. Duflot. Programmed i/o accesses : a threat to virtual machine monitors. In *Pacific security conference PacSec07*, 2007.
2. D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using ic fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 296–310, 2007.
3. AMD. I/o virtualization technology (iommu) specification. 2006. http ://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf.
4. Advanced Micro Devices (AMD). Nx flag by amd. 0. http ://www.amd.com.
5. BSDDaemon, Coideloko, and D0nAnd0n. System management mode hack : Using smm for other purposes. In *Phrack Magazine*, 2008. http ://www.phrack.org/issues.html ?issue=65&id=7#article.
6. Y. Bulygin. Insane detection of insane rootkits : Chipset-based approach to detect virtualization. In *Blackhat Briefings USA*, 2008.
7. CELAR. Computer and electronics security applications rendez-vous (c&esar 2007). 2007. http ://www.cesar-conference.fr/.
8. R. Collins. The loadall instruction. In *Tech Specialist Journal*, 1991. http ://www.x86.org/articles/loadall/tspec_a3_doc.htm.
9. R. Collins. Undocumented opcodes : Salc. 1999. http ://www.rcollins.org/secrets/opcodes/SALC.html.
10. Intel Corp. Intel core 2 extreme processor x6800 and intel core 2 duo desktop processor e6000 and e4000 sequence : Specification update. 2007. http ://www.intel.com/technology/architecture-silicon/intel64/index.htm.
11. Intel Corp. Active management technology : Open source drivers and tools. 2008. http ://www.openamt.org.
12. Intel Corp. Alert standard format : Standards-based systems management. 2008. http ://download.intel.com/design/network/papers/ASF_whitepaper.pdf.
13. F. David, E. Chan, J. Carlyle, and R. Campbell. Cloaker : Hardware supported rootkit concealment. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 296–310, 2008.
14. M. Dornseif. Owned by an ipod : Firewire/1394 issues. In *CanSecWest security conference core05*, 2005. http ://cansecwest.com/core05/2005-firewire-cansecwest.pdf.
15. L. Duflot. Cpu bugs, cpu backdoors and consequences on security. In *ESORICS 2008 : Proceedings of the 13th European Symposium on Research Computer Security*, 2008.
16. L. Duflot, D. Etiemble, and O. Grumelard. Security issues related to pentium system management mode. In *Cansecwest security conference Core06*, 2006. http ://www.cansecwest.com/slides06/csw06-duflot.ppt.

17. Shawn Embleton and Sherri Sparks. The system management mode (smm) rootkit. In *Black hat briefings*, 2008.
18. Intel Corp. Execute disable bit software developer's guide. 0. http ://cache-www.intel.com/cd/00/00/14/93/149307_149307.pdf.
19. Intel Corp. Intel 64 and ia 32 architectures software developer's manual volume 1 : basic architecture. 2007. http ://www.intel.com/design/processor/manuals/253665.pdf.
20. Intel Corp. Intel 64 and ia 32 architectures software developer's manual volume 3b : system programming guide part 2. 2007. http ://www.intel.com/design/processor/manuals/253669.pdf.
21. K. Kaspersky. Remote code execution through intel cpu bugs. In *Hack In The Box Security Conference*, 2008.
22. S. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *Proceedings of the first usenix workshop on large scale exploits and emergent threats, LEET'08*, 2008.
23. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO'99 : Proceedings of Advances in Crytology*, 1999.
24. H. Morin. Deux experts mettent en garde sur la sécurité sur internet. In *Le Monde*, 2007.
25. Trusted Computing Group. Tpm specification version 1.2 : Design principles. 2008. https ://www.trustedcomputinggroup.org/specs/TPM/MainP1DPrev103.zip.
26. University of Cambridge. Xen virtual machine monitor. 2007. http ://www.cl.cam.ac.uk/research/srg/netos/xen/documentation.html.
27. Rafal Wojtczuk. Subverting the xen hypervisor. In *Blackhat Briefings USA*, 2008.