# LES MÉCANISMES DE GESTION INTERNE D'UNE PLATE-FORME INFORMATIQUE : SMM ET ACPI

Loïc Duflot, Olivier Levillain et Benjamin Morin – Ingénieurs de recherche à l'ANSSI

***mots-clés : SMM / ACPI***

*Les ordinateurs sont devenus des machines très complexes, qui résultent de l'assemblage de nombreux composants issus de fabricants différents. Pour assurer le fonctionnement correct de l'ensemble, le système a notamment besoin de gérer l'alimentation de la plate-forme et de répondre à certains événements bas-niveau. Si ce besoin concernait initialement surtout les ordinateurs portables, il s'est généralisé à l'ensemble des systèmes informatiques.*

La manière de traiter ces événements dépend fortement de la plate-forme utilisée. Cependant, les systèmes d'exploitation sont par nature des objets génériques qui sont amenés à s'exécuter sur une multitude de plates-formes différentes et dont ils ne peuvent pas connaître toutes les spécificités. Il est donc logique de confier la gestion de la maintenance de la plate-forme matérielle à un composant logiciel fourni avec celle-ci. La fourniture de ce composant logiciel incombe au fabricant de la carte mère.

Cet article s'intéresse à la fonction de gestion des plates-formes x86 et à ses répercussions sur la sécurité. Cette fonction repose sur le mode SMM (*System Management*) et sur l'ACPI (*Advanced Configuration and Power Interface*), auxquels sont consacrés les deux premières sections. Nous analysons ensuite leur impact sur la sécurité des systèmes, en particulier du point de vue de l'informatique de confiance, puis nous évoquons des attaques qui exploitent les faiblesses de ces mécanismes. La dernière section traite des contre-mesures.

## 1 System Management Mode (SMM)

Comme nous l'avons vu dans le premier article de ce dossier, les systèmes d'exploitation utilisent normalement les processeurs x86 en mode protégé (ou en mode long en 64 bits). Le mode SMM est un autre mode de fonctionnement théoriquement dédié à l'exécution de routines de maintenance de la plate-forme, qui sont spécifiées par le concepteur de la carte mère. Ce mode a été introduit avec la génération 386 des processeurs compatibles Intel autour de 1985, et existe encore aujourd'hui comme le décrit **[1]**.

### 1.1 Fonctionnement général

La seule manière de faire basculer le processeur en mode SMM est une interruption matérielle appelée *System Management Interrupt* (SMI). Cette interruption est générée par le chipset à la survenue de certains événements sur la carte mère, tels que le réveil d'un périphérique, la surchauffe de certains composants, des notifications en provenance de la batterie ou encore diverses pannes. (**[2]** donne plusieurs exemples précis de tels événements). En théorie, seuls des événements matériels peuvent déclencher une telle interruption. Cependant, la plupart des chipsets fournissent un registre appelé APMC (*Advanced Power Management Control*) dans lequel toute écriture déclenche une SMI. Ce registre peut servir à des échanges entre différents composants en charge de la gestion de l'alimentation et de la configuration.

La routine de traitement de la SMI est écrite par le concepteur de la carte mère, afin d'exonérer les systèmes d'exploitation de la connaissance des spécificités de chaque plate-forme. Compte tenu de sa proximité avec le matériel, cette routine a besoin de privilèges élevés pour fonctionner. Le code qui s'exécute en mode SMM dispose de fait d'un accès illimité à la mémoire physique de la machine (dans la limite de 4Go) et à l'ensemble des registres de configuration ou de l'espace mémoire propre des périphériques, quel que soit le mécanisme d'accès utilisé (PMIO, configuration PCI ou MMIO). Il n'existe pour l'heure aucun mécanisme analogue à celui du mode protégé permettant au mode SMM de contrôler les opérations des routines qu'il exécute (même si c'est moralement ce en quoi consisterait le *SMM Transfer Monitor* dont il est question plus loin).

Le processeur bascule automatiquement en mode SMM à la réception d'une SMI. Il sauvegarde préalablement son contexte, en particulier l'état des registres de données et de contrôle, dans une zone de la mémoire physique appelée SMRAM (*System Management RAM*). L'adresse de base de la SMRAM est stockée dans le registre interne du CPU appelé SMBASE. Une des particularités de ce registre est qu'il n'est pas directement accessible, ni en écriture, ni en lecture. En revanche, ce registre est sauvegardé en mémoire comme les autres registres du processeur au moment de la bascule en mode SMM.

Une fois la bascule effectuée, le processeur exécute le code présent à l'adresse SMBASE + 0x8000, où est censée se trouver la routine de traitement de la SMI, qui a été copiée en mémoire par le BIOS lors du démarrage de la machine.

Lorsque la routine de traitement de la SMI exécute l'instruction assembleur rsm, le processeur restaure intégralement son contexte à partir de celui qui a été sauvegardé en SMRAM. Il est important de noter que la routine de traitement de la SMI a totalement le droit de modifier le contenu de l'état sauvegardé du processeur, et en particulier la valeur de SMBASE sauvegardée. C'est d'ailleurs la seule manière de modifier la valeur de SMBASE.

Si le contexte n'a pas été modifié explicitement par la routine de traitement de la SMI, l'état des registres est restauré à l'identique. Du point de vue du système d'exploitation qui a été interrompu par la SMI, tout se passe donc comme si aucune interruption n'avait eu lieu.

## 1.2 Protection de la SMRAM

La routine de traitement de la SMI s'exécute avec des privilèges maximaux sur le système et dans un contexte où le système d'exploitation est gelé ; ce dernier n'est donc pas en mesure d'imposer sa politique de sécurité. Un attaquant capable d'injecter du code dans cette routine pourrait alors simplement prendre le contrôle de la machine cible. Pour empêcher des *rootkits* en espace noyau d'injecter du code dans la routine de traitement de la SMI, les chipsets implantent un mécanisme de contrôle d'accès à certaines zones mémoire susceptibles d'accueillir la SMRAM (et donc la routine de traitement de la SMI). Ces zones mémoire sont au nombre de trois (*Legacy SMRAM*, *High SMRAM* et TSEG), mais nous n'entrerons pas dans les détails de leurs spécificités.

Le mécanisme de contrôle d'accès est configuré par certains bits des registres SMRAMC (*SMRAM Control*) et ESMRAMC (*Extended SMRAM Control*).

- Le bit **D_OPEN** du registre SMRAMC détermine dans quel mode de fonctionnement du processeur les zones sont accessibles. S'il vaut 0, l'accès est restreint au mode SMM. Dans le cas contraire, les zones sont accessibles sans restriction à l'OS.
- Le bit **D_LCK** contrôle quant à lui l'accès aux registres de contrôle. Lorsque celui-ci est positionné à 1, **D_OPEN** est mis à 0 et les registres SMRAMC et ESMRAMC sont verrouillés. Seul un redémarrage les rend accessibles de nouveau en écriture.

À l'aide de ce mécanisme, le BIOS (qui ne s'exécute pas en mode SMM) peut donc configurer proprement la SMI (avec le bit **D_OPEN** à 1), puis mettre le bit **D_LCK** à 1 de telle sorte qu'il ne soit plus possible à quelque composant logiciel que ce soit de modifier la routine de traitement de la SMI (à l'exception de la routine elle-même qui s'exécute en mode SMM).

La plupart des concepteurs de BIOS appliquent ce mécanisme de contrôle d'accès à bon escient depuis 2006 en intégrant la routine de traitement de la SMI dans une des trois zones protégées, puis en mettant le bit **D_LCK** à 1, mais cela n'a pas toujours été le cas. Ce mécanisme interdit en théorie toute modification de la routine de traitement de la SMI, même à un rootkit en mode noyau.

# 2 ACPI

La gestion de l'alimentation et de la configuration d'une plate-forme x86 consiste notamment en la manipulation de registres situés dans le chipset ou au sein des périphériques. Avant l'introduction de l'ACPI, ces manipulations étaient réalisées par le BIOS, selon la norme APM (*Advanced Power Management*). L'APM permettait au système d'exploitation de connaître les routines du BIOS qui devaient s'exécuter pour effectuer telle ou telle opération. Le BIOS était donc le seul composant à manipuler ces registres de configuration.

En 1996, Hewlett-Packard, Intel, Microsoft, Phoenix et Toshiba ont rédigé la spécification de l'ACPI, qui a supplanté l'APM. La dernière révision de la norme ACPI date de juin 2009 **[3]**.

Le principe général de l'ACPI est de confier la gestion de l'alimentation et de la configuration de la machine au système d'exploitation. Pour ce faire, la norme ACPI propose un langage appelé ASL (*ACPI Specification Langage*). Il sert à la description des composants d'une plate-forme et des directives permettant au système d'exploitation de les gérer. Ces directives sont transmises par le BIOS au système d'exploitation, qui les interprète. Ce fonctionnement exonère ainsi les systèmes d'exploitation de la connaissance des spécificités de toutes les plates-formes sur lesquelles ils sont censés s'exécuter, tout en leur laissant une vision haut-niveau de la gestion de l'alimentation et de la configuration ; il exige en contrepartie une confiance dans ces directives, qui sont écrites par les fabricants des plates-formes.

Dans le modèle ACPI, la plate-forme fournit ainsi :

- un BIOS ACPI ;
- des registres ACPI, liés à la gestion de l'alimentation et de la configuration de la plate-forme ;

- des directives de gestion sous la forme de tables ACPI, qui décrivent notamment les opérations à effectuer sur les registres ACPI en fonction de l'action à effectuer.

## 2.1 ACPI Specification Language (ASL)

Le langage ASL sert d'une part à décrire les registres ACPI correspondant aux différents périphériques et d'autre part à définir les méthodes sur ces objets, méthodes qui peuvent agir sur les registres ACPI via des opérations booléennes et arithmétiques. Le langage inclut également des commandes et des structures de contrôles (expressions conditionnelles et boucles), ainsi qu'une directive d'envoi de message à un sous-composant du système d'exploitation (commande **Notify()**). Sous Linux, ces notifications sont envoyées au démon **acpid** lorsque ce dernier est en écoute.

L'exemple suivant décrit un périphérique fictif **TEST** auquel on attribue une zone de registre **REG** avec l'instruction **OperationRegion()**. Cette zone contient un unique registre **CNT** sur un octet. Les deux méthodes qui suivent servent respectivement à mettre à zéro le registre et à incrémenter sa valeur.

```
Device(TEST){
   OperationRegion (REG, SystemMemory, 0x00000400, 0x01)
   Field (REG, ByteAcc, Nolock, Preserve) { CNT, 8 }

   Method (_STA, 1, NotSerialized)
   {
      Store (0x0, \_SB.PCI0.TEST.CNT)
   }
   Method (_PSR, 1, NotSerialized)
   {
      Increment (\_SB.PCI0.TEST.CNT)
   }
}
```

Les routines ACPI exprimées en ASL sont compilées sous une forme binaire dans le langage AML (*ACPI Machine Language*). C'est sous cette seconde forme que les routines sont stockées dans les tables ACPI. Un système d'exploitation compatible ACPI doit donc embarquer un interpréteur AML pour lire et exécuter les tables.

## 2.2 Les tables ACPI

Il existe de nombreuses tables ACPI. La première est la RSDT (*Root System Description Table*), dont il existe une version 64 bits, la XSDT (*eXtended System Description Table*), qui pointe vers les autres tables. Certaines ont un rôle uniquement descriptif pour l'emplacement de registres standards. D'autres contiennent des méthodes AML à interpréter, comme la DSDT (*Differentiated System Description Table*) ou les SSDT (*Secondary System Description Table*) optionnelles.

La DSDT et les SSDT décrivent les périphériques qui supportent la gestion de l'alimentation. Ces périphériques sont organisés en paquetages dans une structure arborescente. **_PR** correspond par exemple aux différents processeurs du système et **_SB.PCI0** au premier bus PCI. Les méthodes sont situées au niveau des feuilles de l'arbre. Ainsi, la méthode pour faire passer le premier contrôleur USB dans l'état actif (état S0) est **_SB.PCI0.USB0._S0**.

Ces tables sont accessibles en lecture sous Linux dans le répertoire **/sys/firmware/acpi/tables** (l'emplacement **/proc/acpi** est désormais obsolète). Réciproquement, il est possible d'imposer des tables ACPI modifiées à un système lors de la compilation du noyau.

## 2.3 Les interruptions

La norme ACPI décrit enfin le fonctionnement des interruptions matérielles liées à la gestion de l'alimentation et de la configuration. Elles sont de trois sortes :

- Les *OS Transparent Events* sont des fonctions spécifiques à la plate-forme, et dont le système d'exploitation n'est pas conscient. Ces événements sont pris en charge par des SMI, décrites précédemment.
- Les *Interrupt Events* sont des événements destinés au système d'exploitation. Si le système d'exploitation est compatible ACPI, une interruption visible par ce dernier sera levée, appelée SCI (*System Control Interrupt*). En revanche, pour un système non ACPI, la plate-forme déclenchera une SMI.
- Les *Hardware Events* sont des interruptions qui génèrent systématiquement une SCI. Elles ne sont donc pas prises en compte par un système non ACPI.

# 3 Analyse de sécurité

## 3.1 Quelques scénarios d'attaque

Les mécanismes ACPI et SMM décrits ci-dessus bénéficient tous les deux de privilèges importants. En effet, le code des tables ACPI accède à l'ensemble des registres ACPI, et est généralement exécuté en *ring* 0. Le mode SMM est quant à lui un mode d'exécution extrêmement privilégié du processeur. Il semble donc intéressant pour un attaquant d'y injecter du code.

Cependant, l'injection du code dans les tables ACPI ou dans le traitement de routine de la SMI suppose de la part d'un attaquant des privilèges élevés, généralement équivalents au super-utilisateur **root**. Pour ajouter des méthodes dans la DSDT, par exemple, il est nécessaire de modifier le noyau du système d'exploitation ou de modifier la configuration du *bootloader*. C'est encore

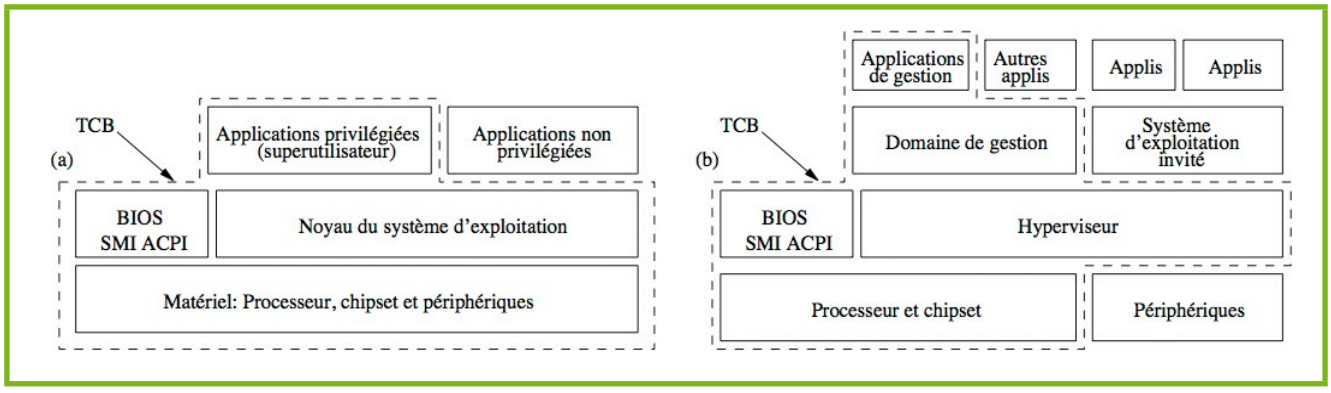


*Figure 1*

plus difficile pour le gestionnaire de SMI, comme nous le verrons dans la section dédiée aux attaques. L'injection de code malveillant dans ces éléments peut également passer par une mise à jour frauduleuse du BIOS.

Notons également que ce type d'attaque repose sur des mécanismes qui dépendent fortement de la plate-forme visée, ce qui limite l'implémentation de codes malveillants génériques.

L'intérêt essentiel que présentent ces vecteurs d'attaque réside principalement dans la furtivité qu'ils confèrent au code injecté. La routine de traitement de la SMI est conçue pour être invisible depuis le système d'exploitation. Les tables ACPI récupérées par le système sont pour leur part traitées comme des boîtes noires contenant du code auquel le système fait une confiance aveugle et qu'il exécute dans diverses circonstances (l'intérêt même de l'ACPI est de fournir de l'information à l'OS qui ne connaît pas la plate-forme).

Le modèle d'attaquant envisagé correspond donc typiquement à un rootkit ou à une porte dérobée : l'attaquant dispose ponctuellement d'un accès très privilégié à la plate-forme (mise à jour BIOS, exécution de code `root`, etc.) et souhaite conserver cet accès privilégié dans le temps de manière discrète. Bien entendu, la persistance de l'attaque varie selon que l'attaquant a modifié le BIOS ou qu'il a seulement modifié les composants en RAM.

D'autres scénarios d'attaques sont également envisageables, en cas de *bug* dans une table ACPI ou dans une routine de traitement de la SMI. Il serait alors possible à un attaquant, même non privilégié, d'exploiter ces erreurs pour réaliser une escalade de privilèges. Dans tous les cas, le succès de l'attaque dépend fortement du matériel considéré.

## 3.2 Impact sur l'informatique de confiance

De nombreuses initiatives, comme le groupement d'industriels TCG (*Trusted Computing Group* **[4]**), cherchent à accroître le niveau de confiance des utilisateurs dans les systèmes d'information. D'une manière générale, un niveau de confiance élevé passe par la maîtrise d'un ensemble restreint de composants matériels et logiciels dont le bon fonctionnement est essentiel. Cet ensemble de composants constitue le socle de confiance, appelé *Trusted Computing Base* (TCB). Intuitivement, plus ce socle de confiance est réduit, plus il est aisé de s'assurer de son bon fonctionnement. Deux exemples classiques de socles de confiance sont donnés dans la figure 1.

Plusieurs acteurs industriels ou académiques ont proposé des modèles visant à réduire le périmètre de ce socle de confiance. Les technologies Intel TxT **[5]** et AMD-V Presidio **[6]** permettent par exemple au système de déclencher à tout instant des vérifications d'intégrité de la plate-forme, sans nécessiter un redémarrage de la machine et en excluant le code de démarrage et de configuration des périphériques (le BIOS) de la zone de confiance. Ces vérifications doivent évidemment être elles-mêmes réalisées dans un environnement d'exécution intègre. Ces technologies sont décrites dans l'article suivant.

S'il paraît effectivement possible à terme d'exclure du socle de confiance une bonne partie des composants d'une machine, un certain nombre de points durs subsistent, tels que la routine de traitement de la SMI et les tables ACPI.

En effet, il existe un autre scénario d'attaque en cas d'utilisation de TxT/Presidio. L'attaquant peut profiter de la phase précédant ce redémarrage de confiance pour corrompre les tables ACPI ou la routine de traitement de la SMI. Une fois le système réputé intègre lancé, les composants modifiés par l'attaquant sont toujours utilisés par la plate-forme pour la gestion du fonctionnement interne, et peuvent donc lui servir à reprendre la main sur le système d'exploitation normalement sécurisé.

# 4 Attaques

L'ACPI et le mode SMM ont été l'objet de plusieurs attaques ces dernières années. Nous évoquons quelques-unes d'entre elles dans cette section. Nous renvoyons le lecteur aux articles originaux pour plus de détails.

## 4.1 ACPI

En 2006, John Heasman a présenté à la conférence *Black Hat* pourquoi et comment un attaquant pourrait corrompre les tables ACPI pour installer des portes dérobées. Les routines contenues dans les tables ACPI peuvent en effet modifier des zones quelconques de la mémoire centrale, en particulier la zone de code du noyau. La présentation **[7]** propose ainsi une porte dérobée sous Windows et sous Linux, en modifiant quelques instructions. Une fois mise en place, la trappe peut être activée par des moyens accessibles simplement depuis l'ACPI, en l'occurrence l'horloge interne de la plate-forme.

Des travaux menés à l'ANSSI **[8]** ont confirmé la faisabilité de telles attaques, en utilisant un autre stimulus pour activer la porte dérobée, le branchement et le débranchement du câble d'alimentation d'un ordinateur portable.

Ce type d'attaque rencontre de sérieuses limitations qui rendent peu probables des attaques de grande envergure utilisant les tables ACPI comme vecteur :

- Comme nous l'avons évoqué plus haut, la mise en place des tables corrompues nécessite un accès privilégié à la plate-forme.
- Les modifications à apporter dépendent de la plate-forme, puisqu'il faut trouver des méthodes ACPI existantes et utilisées sur un système particulier.
- La charge utile dépend du système ciblé. L'expressivité de l'AML est en effet assez réduite et ne permet pas a priori la manipulation de pointeurs. Les attaques citées modifiaient ainsi le code noyau avec des adresses inscrites en dur, ce qui limite considérablement leur portabilité.

### Note

**Nous avons indiqué que l'AML ne permettait pas la manipulation de pointeurs. En effet, les registres ACPI en mémoire centrale sont définis comme des pointeurs constants. Il est cependant possible de faire de l'arithmétique des pointeurs en créant et en chargeant à la volée des tables ACPI depuis du code AML.**

**Il est alors possible d'écrire en théorie du code utilisant des pointeurs, mais cela complexifie les tables et nécessite le chargement de tables à la volée, qui est facilement détectable, et qui peut être limité sur certains systèmes.**

## 4.2 SMM

Dans son utilisation nominale, la SMRAM est censée ne pas être accessible du système d'exploitation et a fortiori des applications en espace utilisateur. Pour garantir cela, le BIOS doit activer la protection de la SMRAM à l'aide du bit **D_LCK**.

Cependant, ce mécanisme n'est pas suffisant. En effet, Laurent Absil et Loïc Duflot ont montré en 2007 comment détourner le mécanisme d'ouverture graphique (GART, *Graphics Address Relocation Table*) du bus AGP pour modifier la SMRAM à travers la plage d'adresse projetée par la carte vidéo, y compris lorsque **D_LCK** est activé **[9]**.

Une attaque similaire a été présentée par Rafal Wojtczuk et Joanna Rutkowska en 2008, utilisant un bug lié aux projections mémoire par le BIOS sur la carte mère Intel Q35 **[10]**. Ici encore, une faille fournissait un accès indirect à la SMRAM, laquelle aurait dû être verrouillée. L'équipe de *The Invisible Things Lab* a également publié une autre attaque exploitant cette fois des vulnérabilités dans le code du gestionnaire de SMI **[11]**.

En 2009, des travaux menés à l'ANSSI ont mis en évidence un autre moyen de modifier la SMRAM en utilisant les caches du CPU (voir **[8]** et **[12]**). L'attaque consiste à modifier la politique de cache de la SMRAM de sorte que les modifications qui y sont apportées le soient de façon différée (politique dite *Write Back*). L'attaquant peut alors écrire un gestionnaire de SMI de taille réduite à l'adresse de la SMRAM, puis déclencher une SMI. Si le cache n'a pas été invalidé dans l'intervalle, c'est le code de l'attaquant qui s'exécute. Il peut alors utiliser cette SMI pour relocaliser SMBASE à une adresse mémoire sous son contrôle.

Toutes ces attaques reposent sur l'absence d'un modèle de sécurité global et cohérent dans les plates-formes x86 :

- D'une part, les mécanismes de sécurité reposent à la fois sur des éléments du CPU (mode d'exéuction) et sur des éléments du chipset (registres de contrôle de la SMRAM).
- D'autre part, le comportement en cas d'utilisation simultanée de différents mécanismes de gestion mémoire (gestion du cache, projection mémoire dans le chipset, contrôle d'accès à la SMRAM) n'est généralement pas défini.

Plus récemment, Alexandre Gazet a présenté une attaque complexe permettant d'injecter du code dans le contrôleur clavier et dans la SMRAM **[13]**. Ces travaux utilisent en particulier un débordement de tampon dans la routine de traitement de SMI. Il est intéressant de constater une nouvelle incohérence dans le modèle de sécurité : en effet, la vulnérabilité est exploitable car le code s'exécutant en SMRAM fait une confiance totale au contrôleur clavier.

Enfin, d'autres erreurs de programmation sont susceptibles d'être exploitées en SMM, par exemple si la routine de traitement des SMI utilise des pointeurs de fonctions situés en dehors de la zone protégée par **D_LCK**. Ceux-ci sont en effet modifiables par l'attaquant que nous avons considéré.

# 5 Contre-mesures et perspectives

## 5.1 ACPI

Une première solution visant à limiter la menace que constitue la corruption des tables ACPI consiste naturellement à vérifier leur **intégrité**. Les tables ACPI devraient ainsi logiquement faire partie des éléments qui sont mesurés par le TPM lors du démarrage de plate-forme, au même titre que ceux sur lesquels repose la confiance dans le système. L'article suivant décrit le principe de fonctionnement de la mesure des éléments au démarrage.

Cependant, cette vérification n'est réalisée qu'au moment du démarrage du système et ne prend pas nécessairement en compte les tables effectivement utilisées par le système. De plus, si les tables sont corrompues dès la mesure initiale, la plate-forme n'est pas protégée. Surtout, les vérifications portent sur l'intégrité des tables, et non sur leur **innocuité**, qui est le réel objectif.

Des techniques d'analyse statique et dynamique peuvent être appliquées aux tables ACPI pour vérifier leur innocuité lors de leur chargement et pour contrôler certains accès en cours d'exécution. Les règles suivantes ont par exemple été testées dans le cadre de travaux menés à l'ANSSI :

- interdire certaines instructions (chargement de tables) ;
- interdire la définition de registres ACPI dans des zones de mémoire utilisée par le système (en espace noyau ou utilisateur).

Bien que ces règles permettent de bloquer les attaques présentées, d'autres attaques plus complexes demeurent possibles, par exemple en programmant certains périphériques pour qu'ils modifient la mémoire centrale par DMA. Ce type de rebond étant spécifique à chaque périphérique, il semble difficile de détecter toutes les attaques de ce type, car cela nécessiterait une compréhension fine du fonctionnement de tous les périphériques par le système d'exploitation, ce qu'est précisément censé éviter l'ACPI. Aussi, des mécanismes complémentaires de défense en profondeur sont nécessaires pour éviter cette catégorie d'attaques, par exemple l'I/O MMU, décrite dans l'article précédent.

## 5.2 SMM

Comme nous l'avons indiqué précédemment, la protection de la SMRAM repose notamment sur la manipulation correcte du bit **D_LCK**. Cette protection est mise en œuvre par défaut dans la majorité des BIOS depuis 2006.

Les attaques reposant sur l'ouverture graphique AGP et autres mécanismes de traduction ont été rendues inopérantes, soit parce que les mécanismes ont disparu, soit parce que des mises à jour des BIOS ont corrigé les incohérences dans l'interaction des mécanismes de gestion mémoire.

Concernant l'attaque utilisant le cache mémoire du processeur, un nouveau registre a été ajouté par Intel sur ses puces en 2009. Le SMRR (*System Management Range Register*) a pour rôle de contrôler finement la politique de cache appliquée à la SMRAM. La manipulation correcte de ce registre bloque l'attaque mentionnée précédemment.

Enfin, un nouveau mécanisme appelé STM (*SMM/SMI Transfer Module/Monitor*) a été spécifié par Intel pour contrôler les opérations effectuées par les routines de traitement de la SMI. Le STM est un module logiciel exécuté en mode SMM, avec lequel le système d'exploitation négocie initialement les opérations autorisées aux routines de traitement de la SMI. Le STM dispose de privilèges supérieurs à ceux des routines SMI et peut ainsi empêcher une routine malveillante d'accéder à des ressources exclues de l'accord avec le système d'exploitation (par exemple, les pages de code du noyau). L'intérêt de la séparation entre le STM et les routines SMI réside dans le fait que le premier est un code générique réutilisable sur de nombreuses plates-formes et a priori simple (donc vérifiable), tandis que le second est spécifique à chaque plate-forme.

## Conclusion

Les mécanismes de gestion interne des plates-formes informatiques posent un certain nombre de problèmes de sécurité, parfois inhérents à la nature de ces mécanismes.

Principalement, il est difficile de contrôler le contenu des tables ACPI ou la routine de traitement des SMI. Il s'agit en effet d'informations spécifiques à la plate-forme, dont il est difficile voire impossible de vérifier l'innocuité. De plus, le contenu de la SMRAM n'est normalement pas accessible au système d'exploitation, y compris en lecture. Aussi, il semble difficile de vérifier quoi que ce soit au sujet du gestionnaire de SMI.

Par ailleurs, des résultats récents montrent que le mode SMM est parfois détourné de son usage normal par le BIOS (fonctions cachées de vérification des mots de passe **[13]**, fonctions ACPI sous-traitées), alors que sa raison d'être initiale était de répondre de manière fiable à des sollicitations urgentes du matériel. On attend donc au contraire de ce composant qu'il reste le plus simple possible. La documentation sur l'ACPI indique explicitement qu'utiliser le mode SMM pour certaines fonctions rend les implémentations difficiles à déboguer.

Enfin, il semble que les choix d'Intel et AMD concernant ces mécanismes consistent essentiellement à ajouter de nouvelles fonctionnalités pour corriger les anciennes, alors qu'il serait essentiel de reposer le modèle de sécurité à plat pour le rendre cohérent.

Ainsi, pour améliorer la confiance dans ces composants, nous avons vu que l'approche directe (vérification de l'intégrité et/ou de l'innocuité) n'était pas chose aisée. Il est donc essentiel de mettre en place les mécanismes de défense en profondeur à notre disposition (I/O MMU, TPM, et bientôt le STM) pour restreindre les privilèges maximaux accordés actuellement à l'ACPI et surtout au mode SMM. ■

## RÉFÉRENCES


**[1] www.intel.com/content/dam/doc/manual/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf**

**[2] Intel I/O Controller Hub 9 (ICH9) family datasheet, www.intel.com/Assets/PDF/datasheet/316972.pdf**

**[3] www.acpi.info/DOWNLOADS/ACPIspec40.pdf**

**[4] www.trustedcomputinggroup.org**

**[5] Grawrock - The Intel safer computing initiative: building blocks for Trusted Computing, Intel Press, 2006**

**[6] AMD64 virtualization: Secure virtual machine architecture reference manual, AMD Publication no. 33047 rev. 3.01, 2005**

**[7] Heasman, Implementing and detecting an ACPI BIOS rootkit, Blackhat federal 2006**

**[8] L. Duflot et O. Levillain, ACPI et routine de traitement de la SMI : des limites à l'informatique de confiance ?, SSTIC 2009**

**[9] L. Absil et L. Duflot, Programmed I/O Accesses: a threat to virtual machine monitors, PacSec 2007**

**[10] J. Rutkowska et R. Wojtczuk, Detecting & Preventing the Xen Hypervisor Subversions, Black Hat 2008**

**[11] R. Wojtczuk et J. Rutkowska, Attacking Intel Trusted Execution Technology, Black Hat DC 2009**

**[12] L. Duflot, O. Levillain, B. Morin et O. Grumelard, System Management Mode Design and Security Issues, IT Defense 2010**

**[13] A. Gazet, Sticky fingers & KBC Custom Shop, SSTIC 2011**