Virtualisation

3 décembre 2007

Gagnaire Laure & Lahoudere Fabien

Table des matières

Présentation

Historique

Juin 1959 : Christopher Strachey publie un papier intitulé « Time Sharing in Large Fast Computer » à la conférence internationale sur le traitement de l'information à l'UNESCO.

En **1974**, Christopher clarifie son idée présenté en 1959 dans un email adressé à Donald Knuth. Il y explique que son papier précédent porte principalement sur la multiprogrammation. Cette idée de multiprogrammation se rapporte à l'idée de ne plus attendre le chargement d'un périphérique, et de pouvoir travailler sur une autre tâche sur son ordinateur. C'est en ce terme qu'il parle de « Time Sharing ».

1960 : La première utilisation de la multiprogrammation peut être attribuer à l'*ordinateur Atlas*. Le projet Atlas est développé conjointement par l'Université de Manchester et Ferrandi Ltd.

En **1961**, les SER (Supervisor extracode routine) forment les « branches principales » du superviseur (terme employé pour décrire un hyperviseur). Ils sont activés aussi bien par des routines[1] d'interruption que par des instructions d'« extracode ». Le superviseur d'Atlas utilise alors une machine virtuelle. Une autre machine virtuelle est alors employée pour faire tourner les programmes des utilisateurs.

Dans la même année, au MIT (Massachusetts Institute of Technology), le *CTSS* (Compatible Time Sharing System) est développé sur des machines IBM 704.

Le programme de supervision du CTSS repose sur une console I/O :

- organisation via des jobs fg et bg
- stockage temporaire
- etc...

Le superviseur a le contrôle direct de toutes les interruptions collectées.

En **1963**, le projet MAC du MIT a pour but le design et l'implémentation du meilleur temps partagé (« Time Sharing ») d'un système basé sur le CTSS.

Vers **1965**, le centre de recherche d'IBM est en charge du projet M44/44X. Son but étant d'évaluer les concepts naissant sur le partage du temps.

[1] partie d'un code informatique destinée à être utilisé plus d'une fois

L'architecture est la suivante :

– une machine principale : *IBM 7044* (M44)

– et d'autres machines où chacune est une image expérimentale de la machine principale (M44)

L'espace d'adressage de 44X réside alors dans la hiérarchie de la mémoire de M44, implémenté via de la mémoire virtuelle et de la multiprogrammation.

En **1967**, l'idée de *virtualisation matérielle* est développée via le modèle 67 (System/360^{TM}) (cf. la figure 1) d'IBM. Ce modèle virtualise toutes les interfaces matérielles via la VMM (Virtual Machine Monitor). Le système d'exploitation est le *TSS* (Time-Sharing System) aussi appellé le superviseur. L'IBM 360/67 est équipé d'un « hyperviseur » appelé CP/67 capable de supporter plusieurs machines virtuelles 360/65 tournant sous des versions de système différentes. CP/67 a engendré VM/CMS (CMS correspond à la notion de système de temps partagé), qu'IBM a commercialisé pendant les années 1970-1980.

FIG. 1 – pupitre d'un modèle 360/67

La VMM tourne directement sur le « vrai » matériel, ceci permet l'existence de plusieurs machines virtuelles :VMs. Où chaque machine virtuelle tourne sur son propre système d'exploitation. Les machines virtuelles d'IBM de nos jours sont des offres très respectées et des plateformes de calcul robustes.

Durant les **années 90**, la notion de « processeur simulé » voit le jour. C'est la machine *P-code* (ou pseudo-code). C'est un langage machine qui est exécuté dans une machine virtuelle. Le langage *Java*^{TM} a suivi le modèle de P-code pour sa machine virtuelle. Ceci a permis la distribution large des programmes de Java au-dessus des architectures innombrables en mettant en communication simplement la JVM (Java Virtual Machine).

A la fin des **années 90** et au début des **années 2000**, la société VMware développe et popularise un système propriétaire de virtualisation logicielle des architectures du type x86 pour les architectures de type x86.

L'architecture IA-32 (X86) fournit alors quelques instructions pour la virtualisation. Il faut savoir que les instructions liées au mode privilégié peuvent renvoyer différents résultats en fonction du mode. Par exemple, l'instruction x86 STR s'occupe de rapporter l'état de sécurité.

Mais, la valeur qu'elle renvoie dépend du niveau de privilège .Ainsi, c'est pour cela que dès **2006** les fabricants de processeur x86 que sont AMD et Intel ont proposé dans leurs gammes de la *virtualisation matérielle.*

Utilisations

La virtualisation séduit autant les programmeurs que les enseignants pour de multiples raisons.

1. Développement
   - tracer les programmes : faire du debuggage
   - tester la portabilité : plusieurs systèmes d'exploitation sur un ordinateur
   - déploiement et migration de machines virtuelles
   - possibilité de créer des points de restauration de machines virtuelles par le biais de « snapshot » : image d'une machine virtuelle à un instant donné
2. Enseignement
   - isoler des machines entre elles : empêcher des étudiants de communiquer entre eux lors de devoirs, ou de casser des machines
3. Sécurité par l'isolation
   - s'emprisonner dans un répertoire d'un système hôte, où l'accès à ce dernier est impossible depuis le répertoire
   - restreindre l'accès à une partie de la mémoire
   - jail : enfermement dans une machine virtuelle avec un accès restreint à la mémoire

Définitions utiles

Définition 1. *Virtualisation (définition générale) : elle a pour but de donner un environnement système au programme afin de le faire croire être dans un environnement matériel.*

Définition 2. *Machine virtuelle (VM - Virtual Machine) : le sens originel de machine virtuelle est la création de plusieurs environnements d'exécution sur un seul ordinateur, dont chacun émule l'ordinateur hôte. Cela fournit à chaque utilisateur l'illusion de disposer d'un ordinateur complet alors que chaque machine virtuelle est isolée des autres. Le logiciel hôte qui fournit cette fonctionnalité est souvent dénommé superviseur ou hyperviseur. Ce concept va plus loin que celui des simples temps partagés où chaque utilisateur dispose seulement d'un espace de développement personnel, et non d'une machine simulée entière.*

Définition 3. *Hyperviseur (ou VMM - Virtual Machine Monitor) : c'est le logiciel hôte qui fournit la fonctionnalité d'exécuter une machine virtuelle sur un système. Le système d'exploitation communique directement avec le matériel.*

Système de base

Un tel système est basé sur un système d'exploitation prévu pour fonctionner avec le matériel utilisé.

Tout d'abord, l'application envoie des instructions au système d'exploitation qui permet au matériel de l'exécuter. La figure 2 décrit ce système.

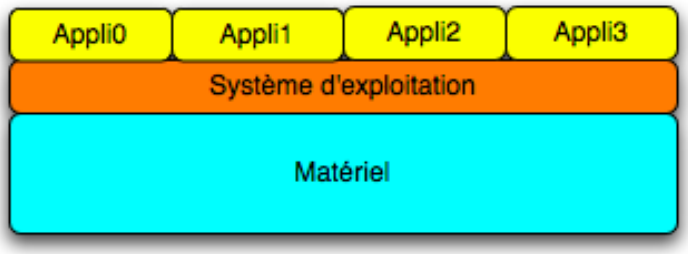

FIG. 2 – système de base

Chapitre 1

Les différents types de virtualisation

1.1 L'émulation

Dans ce cas, un environnement est créé où l'on simule les périphériques hardware. Chaque application donne des instructions via des pilotes à l'hyperviseur qui doit passer par le système d'exploitation pour exécuter celles-ci (cf. la figure 1.1).

Avantage : ce type de virtualisation permet de travailler avec des systèmes prévus pour un matériel totalement différent que celui sur lequel il tourne grâce a l'émulation du matériel.

Inconvénient : il y a une grosse perte de performance du fait du nombre de couches traversées et de la conversion nécessaire pour passer d'un langage correspondant à un matériel à l'autre.

Utilisations : en premier lieu, il est possible alors de tester des applications multiplateformes sans avoir besoin d'acquérir le matériel nécessaire. Ou alors de lancer une application prévue pour un autre matériel (jouer à des jeux d'anciennes consoles sur un ordinateur personnel). Des logiciels tels que Bochs ou QEMU l'utilisent.

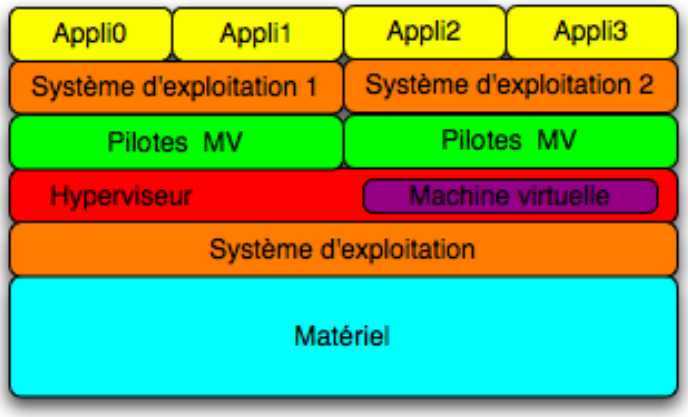

FIG. 1.1 – l'émulation

1.2 La virtualisation totale

Il s'agit d'une émulation plus performante mais plus restrictive car le processeur de la machine virtuelle doit être du même type que celui de la machine hôte. Ainsi on peut placer l'hyperviseur au niveau du système d'exploitation pour avoir un accès plus rapide au matériel (via les pilotes du système d'exploitation hôte). Les applications du système d'exploitation invité ont une couche de moins à traverser pour atteindre le matériel (cf. la figure 1.2).

Avantages : meilleures performances que l'émulation du fait de l'accès plus rapide au matériel.

Inconvénients : les performances ne sont pas optimales dans l'usage de certains périphériques : type carte accélératrice 3D. De plus, il y a réduction des systèmes d'exploitation virtualisables.

Utilisations : faire tourner des systèmes d'exploitation virtuellement en les laissant accéder aux différents périphériques au travers des pilotes installés sur le système hôte. Des logiciels tels que VMWare Workstation ou VirtualPC de Microsoft l'utilisent.

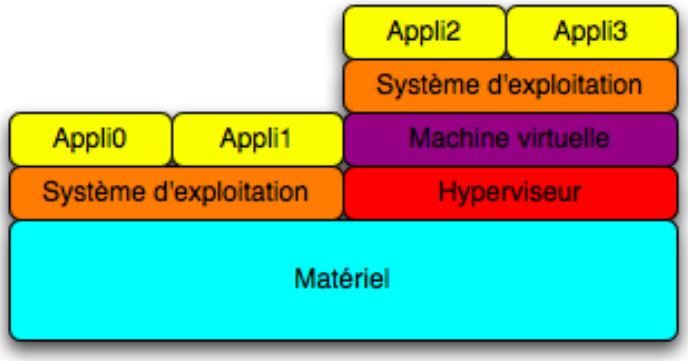

Fig. 1.2 – la virtualisation totale

1.3 La paravirtualisation

On fait tourner l'hyperviseur sur le matériel et les systèmes d'exploitation invités au dessus de ce dernier (cf. la figure 1.3).

Avantages : meilleures performances que la virtualisation totale.

Inconvénients : les systèmes d'exploitation invités doivent être modifiés afin de tourner avec l'hyperviseur. La machine virtuelle doit pouvoir tourner sur le processeur (physique) de la machine hôte.

Utilisations : Des logiciels comme Xen ou UML l'utilisent.

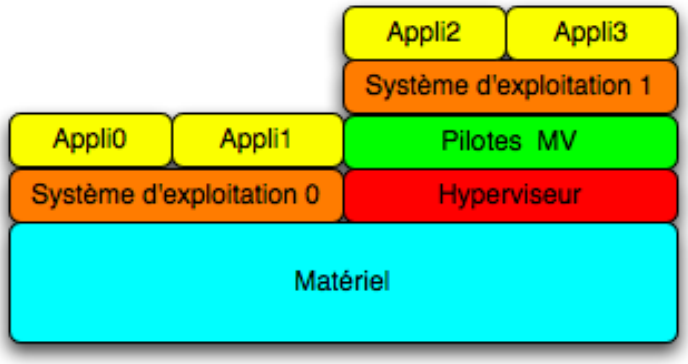

FIG. 1.3 – la paravirtualisation

1.4 La virtualisation matérielle

Des instructions sont ajoutées au processeur pour qu'il serve d'hyperviseur à l'aide du HAL (Hardware Abstraction Layer). Les systèmes d'exploitation invités sont au même niveau que ceux hôtes (cf. la figure 1.4).

Avantages : certains processeurs permettent un accès direct à la mémoire des invités. Les performances sont optimales. Les processeurs ne sont pas emulés et les systèmes d'exploitation inchangés.

Inconvénient : il faut un processeur spécifique. Ainsi, si notre machine ne comporte pas ce type de processeur, alors il est nécessaire de s'équiper d'une machine récente qui comportera celui-ci.

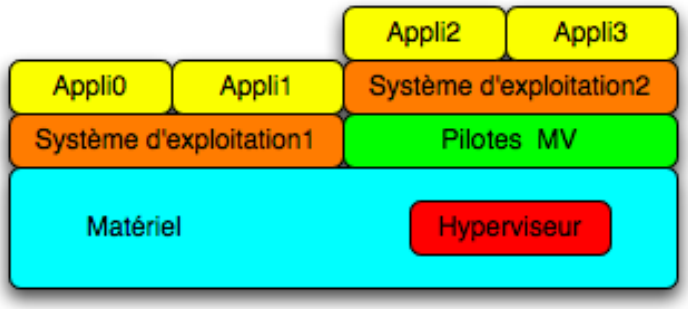

FIG. 1.4 – la virtualisation matérielle

Ce type de virtualisation sera plus détaillé dans le prochain chapitre.

Chapitre 2

Virtualisation matérielle

Ces technologies peuvent être aussi bien utilisées par Xen, VMWare ou encore User-mode Linux.

2.1 Cas d'Intel

La technologie de virtualisation développée par Intel (Intel-VT - anciennement Vanderpool) supporte aussi bien l'architecture x86 (VT-x) que celle $Itanium^R$ (VT-i). Un nouveau mode d'exécution apparait alors : VMX. Ce mode comporte un niveau racine (root) pour l'hyperviseur, correspondant à des ring inférieurs à 0, et un niveau normal pour les systèmes d'exploitation invités (machine virtuelle) (non-root), correspondant aux anciens rings de 0 à 3. Le niveau root est entièrement privilégié, alors que l'autre ne possède aucun privilège, même au « ring 0 ».

L'instruction *VMX_entry* permet de passer de VMX root à VMX non-root, tandis que l'instruction *VMX_exit* permet l'inverse. Cette technologie permet une certaine flexibilité dans la définition des instructions permettant à la machine virtuelle (système d'exploitation invité) de sortir de l'emprise de la VMM (VM_exit). La figure 2.1 montre le passage d'un niveau à l'autre via les deux instructions expliquées précédemment.

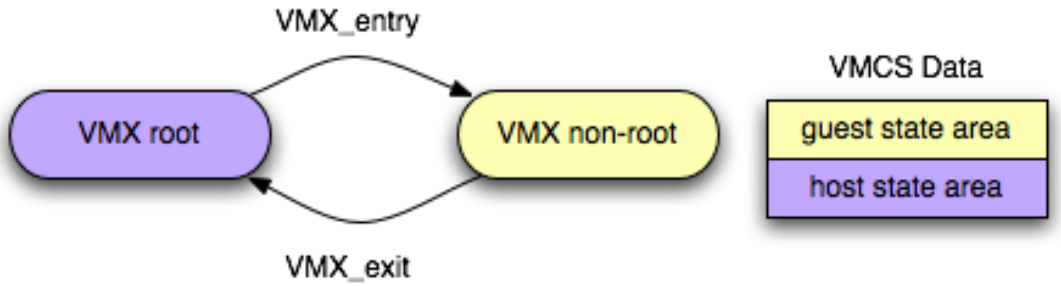

Fig. 2.1 – la technologie INTEL-VT

En mode root l'hyperviseur stocke dans une structure de données les informations relatives au matériel associé à un système invité (processeur(s), blocks mémoire(s), entrées/sorties). Sous la technologie VT-x, celle-ci se nomme *Virtual Machine Control Structure* (VMCS).

Depuis 2007, une 2ème génération de cette technologie est apparue. Elle inclut en plus un support matériel pour la gestion de la mémoire virtuelle : *Nested Page Tables* (NPT). Deux sortes de table de page existent : Virtual Memory -> Physical Memory et Physical Memory -> Virtual Memory. Cette nouveauté permet des gains significatifs de performance pour les applications faisant un usage intensif de la mémoire ou générant un grand nombre de processus système. La figure 2.2 décrit ce nouveau support.

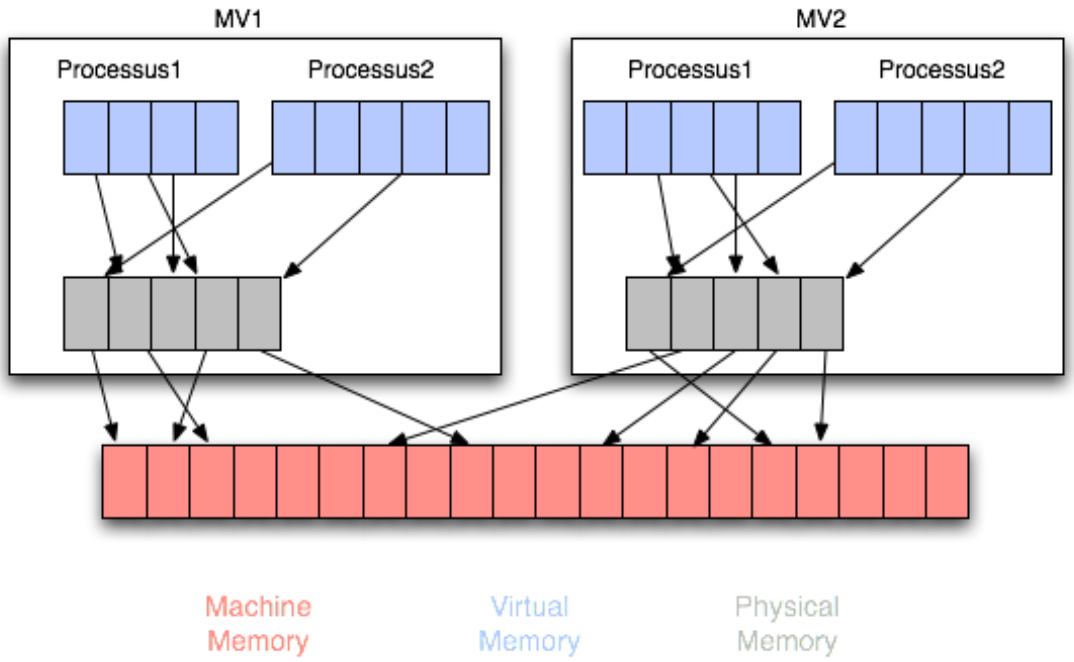

FIG. 2.2 – gestion de la mémoire virtuelle

2.2 Cas d'AMD

La technologie de virtualisation développée par AMD se nomme *AMD-V* (anciennement Pacifica).

Il faut savoir que lors du boot un processeur AMD64 démarre en mode x86_32 afin de maintenir la compatibilité avec les systèmes d'exploitation 32 bits. Le *boot-loader* d'un système d'exploitation 64 bits exécute une instruction qui bascule le processeur en mode x86_64. De manière similaire, un processeur fonctionnant avec la nouvelle technologie démarre en mode invité (toutes les fonctions de virtualisation sont alors inactives) jusqu'à ce qu'un hyperviseur compatible ne soit démarré.

La structure *Virtual Machine Control Block* (VMCB) a la même fonction que celle de *VT-x*.

Une fois la structure *VMCB* définie, l'hyperviseur bascule le processeur en mode invité, puis passe la main à l'OS invité associé qui démarre (se considérant seul au monde sur le matériel) exécutant son code privilégié en ring 0 et son code applicatif en ring 3. L'hyperviseur s'occupe alors de l'exécution et intercepte les instructions privilégiées. Quand une instruction demande un accès à une ressource définie dans la *VMCB* le processeur bascule en mode root. L'hyperviseur gère alors la requête.

L'instruction *VMMCALL* est une instruction qui permet aux systèmes d'exploitation invités de négocier des ressources avec l'hyperviseur.

La *Translation Look-aside Buffer* (TLB) est une table qui contient les références entre adresses réelles et virtuelles des pages mémoire récemment accédées. Le *Device Exclusion Vector* (DEV), quant à lui, permet à l'hyperviseur de savoir immédiatement si un accès à une page mémoire est légitime ou non.

Sous AMD-V, le support matériel pour la gestion de la mémoire se nomme : *Nested Pages* ou *Nested Page Tables* (NPT).

Chapitre 3

Virtualisation et sécurité

3.1 Notion de conteneur

On parle aussi de « virtualisation au niveau du système d'exploitation ». Le but est d'isoler des environnements de l'espace utilisateur entre eux mais en les faisant fonctionner sur le même noyau. Ces systèmes de conteneur permettent alors d'utiliser différents ensembles d'applications dans un même contexte (on peut alors envisager plusieurs distributions Linux avec un unique noyau).

Avantages : un conteneur permet d'isoler le système hôte lors d'un « crash ». Mais aussi de servir de « honeypot » (pot de miel). Le but de cette méthode est de fragiliser le système invité pour attirer les personnes malveillantes. Ainsi, depuis le système hôte, on peut alors observer les moyens de compromission des pirates informatiques, et alors se prémunir contre de nouvelles attaques. Afin qu'un « honeypot » soit efficace, il est nécessaire que le pirate ne détecte pas que le système invité est virtualisé. L'utilisation d'une telle configuration se déroule en trois étapes :

- la surveillance,
- la collecte d'information,
- l'analyse d'information.

Inconvénient : Les systèmes de conteneur ne permettent pas l'utilisation de différents systèmes d'exploitation.

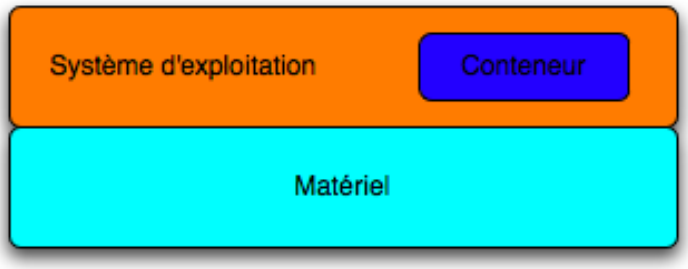

Fig. 3.1 – notion de conteneur

3.2 Cas du rootkit

Concept : mettre un système d'exploitation cible dans une machine virtuelle en contrôlant le système hôte.

3.2.1 Rootkit basé sur le virtualisation totale : l'exemple de Subvirt

Pour ce faire on installe une solution logicielle de virtualisation sur le disque de la machine. Puis, on change la séquence de démarrage pour qu'elle lance en premier lieu le système malveillant puis la solution de virtualisation dans lequel sera lancé le système cible. Ensuite, Subvirt repose seulement sur le logiciel de virtualisation une fois installé (cf. 3.2).

Il est nécessaire que quelques changements soient effectués :
- d'une part, certains antivirus empêche la modification de démarrage. SubVirt utilise donc des pilotes bas niveau pour pouvoir contourner ses protections en se plaçant au dessous des éventuelles protections,
- le système doit être modifier pour empêcher la détection du changement d'environnement par la cible.

La technique n'est pas détectable depuis un live cd du fait de la conception de ce rootkit.

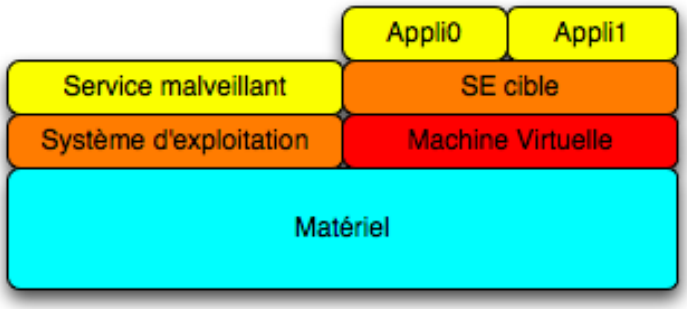

FIG. 3.2 – Rootkit virtualisé sous virtualisation totale

3.2.2 Rootkit basé sur le virtualisation matérielle : l'exemple de Blue Pill

Ce type de rootkit utilise les nouvelles technologies de virtualisation des processeurs. Blue Pill utilise lui un AMD-V. Pour commencer le rootkit met le bit *SVME* du registre *MSR EFER* à 1 pour activer la virtualisation. Ensuite on utilise l'instruction *VMRUN* pour exécuter le code de la machine virtuelle.

Il y a plusieurs paramètres à *VMRUN* contenus dans la *VMCB* :
- liste des instructions à intercepter,
- bits de contrôle pour l'environnement virtuel,
- l'état du processeur virtuel.

On passe l'adresse de cette *VMCB* en paramètre de *VMRUN* pour lancer la machine virtuelle. Pour lancer Blue Pill il faudra donc que la *VCMB* soit correctement rempli pour lancer l'hyperviseur puis la nouvelle machine virtuelle. On remarque ainsi qu'il n'est pas nécessaire de redémarrer l'ordinateur pour lancer ce rootkit. La séquence de démarrage et le système cible n'ont pas besoin d'être modifier pour que ce rootkit s'exécute ce qui le rend beaucoup plus difficile à détecter que SubVirt. Les seules changements interviennent au niveau du processeur. Il y a un inconvénient à cette furtivité : Blue Pill ne survit pas au redémarrage.

Il existe un rookit développé sous la technologie *Intel-VT* sous le nom de Vitriol.

3.3 Détection d'un environnement virtualisé

La notion de détection est très importante. Comme nous l'avons vu précédemment, le concepteur d'un « honeypot » doit faire en sorte que le pirate qui attaque le système invité ne voit pas qu'il est dans un environnement virtualisé. D'un autre côté, dans le cas d'un rootkit utilisant la virtualisation, le pirate lance le système hôte dans une machine virtuelle. L'existence de cette dernière est alors rendue transparente aux yeux de l'utilisateur de la machine.

Différents indices permettent de détecter la présence dans un environnement virtualisé :

Anomalies matérielles : présence de périphériques utilisés exclusivement pour l'environnement virtualisé.

Exemple : dans le cas de l'utilisation d'un environnement UML, la commande **cat /proc/cpuinfo** nous donne les informations suivantes :
- vendo_id : User Mode Linux,
- model name : UML.

On constate alors que l'on se trouve dans un environnement UML.

Etrangetés dans la mémoire : présence du nom de l'environnement virtualisé dans celle-ci.

Exemple : dans le cas de l'utilisation du logiciel VMWare, on remarque la présence du mot **vmware** dans la réponse à la commande **ls /proc/kcore**.

Originalité à l'exécution : en utilisant **gdb**, on peut alors remarquer que certaines adresses mémoires ont des valeurs plus hautes ou plus basses que d'habitude.

« Timing attacks » : on regarde le temps d'une requête. Si le temps est anormalement élevé par rapport à un autre système équivalent, alors il y a des chances pour que le système soit virtualisé.

Exemple : la commande **time ls** nous renverrait une valeur anormale dans un environnement virtualisé.

Il est nécessaire de noter que certains indices ne sont pas efficaces selon l'outil de virtualisation utilisé. Par exemple, dans le cas d'une virtualisation matérielle, l'étude des adresses mémoires ne fonctionne alors plus.

Des « timing attacks » peuvent être utilisées pour détecter la présence de rootkits virtualisés matériellement. Dans le cas de Vitriol, sous *Intel-VT* beaucoup d'instructions assembleur entraînent une instruction *VM_exit*. Ainsi, si on se trouve dans un monde virtuel, ces appels mettront beaucoup de temps.

Conclusion

Il n'y a pas un type de virtualisation à utiliser plus qu'un autre. Il s'agit plutôt de sélectionner la virtualisation correspondante à nos besoins.

La virtualisation matérielle est une technologie en constante évolution. Le support permettant la gestion de la mémoire virtuelle ne datant que de 2007.

En ce qui concerne les rootkits dernière génération utilisant la virtualisation eux aussi évoluent. Des rootkits comme Blue Pill ou encore Vitriol ont été développés sous virtualisation matérielle presque dès l'arrivée des processeurs permettant la virtualisation matérielle.

Il est nécessaire de noter qu'en ce qui concerne les rootkits virtualisés il n'existe pas actuellement de techniques génériques pour les détecter.

Bibliographie

[1] Thomas Ptacek. Detecting Virtualized Rootkits.
http ://www.matasano.com/log/680/detecting-virtualized-rootkits/. 24 Janvier 2007

[2] Amit Singh. An Introduction to Virtualization.
http ://www.kernelthread.com/publications/virtualization/. 2003

[3] David Delavennat, Stéphane Aicardi. Technologies informatiques récentes et Supports matériels. 2007.
www.mathrice.org/rencontres/mars.2007/presentations/evolutions_materielles_recentes.pdf.

[4] Virtualisation.
http ://fr.wikipedia.org/wiki/Virtualisation.

[5] Julien Bachmann, Sébastien Bombal. Rootkits et Virtualisation. Magazine Misc.
Septembre/Octobre 2007.

[6] Rootkits et antirootkits. http ://infomars.fr/forum/index.php ?showtopic=377. 2007

[7] Christophe Bardy. Virtualisation : VT-x et AMD-V ne servent à rien... .
http ://idg3.typepad.com/infrastructure/2006/11/vtx_et_amdv_ne_.html. 2006.