42 MARS/AVRIL 2009

France Métro 8 € / DOM 8,80 € / TOM Surface 990 XPF / TOM Avion 1300 XPF / CH 15,50 CHF BEL. LUX, PORT.CONT 9 Eur / CAN 15 $CAD

MISC

Multi-System & Internet Security Cookbook

100 % SÉCURITÉ INFORMATIQUE

INFOWAR

Défense et sécurité nationales : Où en est la guerre de l'information en France ?
(p. 4)

DOSSIER

LA VIRTUALISATION :

VECTEUR DE VULNÉRABILITÉ OU DE SÉCURITÉ ?

- Microsoft Hyper-V
- Mécanismes internes de Xen
- Bonnes pratiques avec VMWare ESX

FICHE TECHNIQUE

Émulation d'architectures réseau avec Dynamips/Dynagen/GNS3
(p. 71)

SCIENCE

Utilisation des méthodes formelles pour le test des algorithmes cryptographiques
(p. 77)

PROGRAMMATION

Contourner les techniques avancées et complexes d'obfuscation de code
(p. 50)

L 19018 - 42 - F: 8,00 € - RD

ÉDITO ▶ Discours xyloglottes et autres logorrhées

Parfois, l'inspiration, telle l'eau, coule de source. Je me laisse porter par des envolées lyriques qui confirment, s'il en était besoin, que le ver est bien dans le poème. C'est donc d'une main tremblante que je prends la parole pour vous adresser cet édito enflammé qui, je l'espère, rafraîchira l'atmosphère – si elle en avait besoin. C'est avec la langue que je mets donc les pieds dans le plat.

Je mettrai aussi de l'eau dans mon vin et, pour ne pas jeter d'huile sur le feu, je traiterai d'un ton atone, mais néanmoins juste et précis, des difficultés de la communication. La population homogène et disparate des acteurs du monde de la sécurité est murée dans un grand espace infini aux limites duquel la compréhension se heurte parfois à un obstacle imprévu.

Prenons l'opposition qui rassemble les *techos* et les certifiés 27001. Certes, il s'agit souvent d'un combat sanglant. D'où l'importance de tirer la chaîne et tisser les liens qui initieront le mouvement perpétuel et jetteront au sol (pleureur et meunière) les fondations du boulot. Mais, le dialogue entre ces corps, que dis-je, ces troncs, s'appuie sur un non-dit largement sous-entendu : chacun détient la vérité. C'est là l'arbre qui cache le buisson.

Pour les premiers, la vérité est dans l'assembleur, dans l'octet, dans le bit qui marque le rythme de toute partition. Il est question de fouiner dans les recoins de chaque élément d'un système d'informations, et de chercher comment détourner le flux des battements numériques. Une telle eurythmie n'est pas un joli rêve, mais la réalité quotidienne et abracadabrantesque de nombreux *nerds* dont la survie dépend du bon vouloir d'une hiérarchie pas toujours, voire rarement, complaisante.

Pour les seconds, tout est processus, normes et procédures. Les volumes de papiers, bien rangés dans des placards obscurs et abscons, portent d'une voix forte, insidieuse et affirmative, les interrogations que tout un chacun doit se poser pour répondre aux questions sur son besoin de protection. Il s'agit de stigmatiser la griffe de l'angoisse qui saisit à la gorge le cœur de leurs incertitudes.

Les premiers reprochent aux seconds de négliger la réalité de la vraie vie. Les mesures proposées par les seconds reviendraient à se tirer une balle de plus dans le pied, à se mettre une épée de Damoclès au-dessus de la tête, puis à couper le crin de cheval qui la retient.

Les seconds reprochent aux premiers de négliger le cadre structurant et rassurant qui entoure notre domaine. La magie irrationnelle et mystérieuse pratiquée par les premiers s'oppose à la rigueur apaisante des seconds qui partent du principe que tout exercice doit être répété quatre ou cinq fois jusqu'à ce qu'il soit réussi du premier coup.

Cette incompréhension, véritable épine dans le pied, coupe régulièrement les bras et étrangle l'ossature indispensable au bon fonctionnement de notre activité. Il est nécessaire de lancer des ponts qui soutiendront les passerelles entre les berges au-dessus du grand fossé abyssal qui nous sépare. Nous ferions mieux de marcher main dans la main et cheveux au vent, en nous serrant les coudes pour former un front uni. Mais, seul l'avenir nous dira ce que le futur nous réserve, alors attendons pour y voir clair à la lumière du temps qui passe.

Sans rapport, mais pas abstinent pour autant, je rappelle qu'il est temps de prendre ses jambes à son cou pour que l'œil vaillant veille, le doigt aux aguets : les places pour SSTIC seront bientôt en vente (si ce n'est déjà le cas lorsque vous lirez, l'eau à la bouche et l'œil haletant, ces lignes).

Bonne lecture,

Fred Raynal, en direct et fortement imprégné de l'air de Champignac.

SOMMAIRE

INFOWAR [04 - 07]
> Vers une version française de la guerre de l'information ?

DOSSIER [08 - 49]
[LA VIRTUALISATION : VECTEUR DE VULNÉRABILITÉ OU DE SÉCURITÉ ?]
> Introduction à la virtualisation / 08 → 11
> Inside Microsoft Hyper-V / 12 → 18
> Voyage dans l'antre de Xen / 19 → 27
> Bonnes pratiques pour la virtualisation avec VMware ESX / 28 → 39
> Détection opérationnelle des rootkits HVM (Partie 1) / 40 → 49

PROGRAMMATION [50 - 59]
> L'obfuscation contournée (Partie 2)

SYSTÈME [60 - 67]
> La sécurité des clés USB (partie 2)

FICHE TECHNIQUE [71 - 76]
> Émulation d'architectures réseau : présentation de Dynamips/Dynagen/GNS3

SCIENCE [77 - 82]
> Sûreté de fonctionnement et sécurité des algorithmes cryptographiques

ABONNEMENTS/COMMANDE [68 - 70]

Rendez-vous au 07/05/2009 pour le n°43 !

MISC
est édité par Les Éditions Diamond
B.P. 20142 - 67603 Sélestat Cedex

Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : cial@ed-diamond.com
Service commercial : abo@ed-diamond.com
Sites : www.ed-diamond.com
www.miscmag.com

LES ÉDITIONS DIAMOND

Imprimé en France
Dépôt légal : à parution
N° ISSN : 1631-9036
Commission Paritaire : K 81190
Périodicité : Bimestrielle
Prix de vente : 8 Euros

Membre April
www.april.org

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Frédéric Raynal
Relecture : Dominique Grosse
Secrétaire de rédaction : Véronique Wilhelm
Conception graphique : Kathrin Troeger
Responsable publicité : Tél. : 03 88 58 02 08
Service abonnement : Tél. : 03 88 58 02 08
Impression : SIB Imprimerie (Boulogne-sur-Mer / France)
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. **Tél.** : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. **Tél.** : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 61 72 76 24

Charte du magazine : MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information et les problèmes de sécurité qui l'accompagnent.
MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate.
MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

Daniel Ventre – CNRS/CESDIP – daniel.ventre@gern-cnrs.com

VERS UNE VERSION FRANÇAISE DE LA GUERRE DE L'INFORMATION ? (PARTIE 1)

mots-clés : *guerre de l'information / temple des opérations d'information / maîtrise de l'information / sécurité / défense nationale*

La guerre de l'information est généralement définie en France par la désormais célèbre formule « par, pour et contre l'information » [1], laquelle est d'ailleurs reprise dans le récent rapport sur la Cyberdéfense [2] (publié dans la foulée du Livre blanc sur la défense et la sécurité nationale 2008) [3]. Mais la France ne dispose toujours pas d'un document unique de référence en matière de guerre de l'information, qui définirait à la fois le concept, les enjeux, les outils et les rôles respectifs des acteurs civils et/ou militaires. Alors, chacun l'utilise un peu à sa guise, tantôt synonyme de guerre ou intelligence économique, de manipulation de l'information, du rôle des médias, d'information dans la guerre, de cyberguerre et de cyberattaques, d'espionnage, d'opérations d'information, d'opérations d'influence, voire de cybercriminalité. Prise semble-t-il dans la tourmente des vagues de cyberattaques qui ont jalonné les deux dernières années, la France a placé la sécurité des systèmes d'information au rang d'enjeu de défense et de sécurité nationale (§1). Les menaces telles qu'elles sont identifiées et perçues (§2) justifient une stratégie fondée sur la maîtrise de l'information, des systèmes, et la mise en œuvre de moyens de lutte informatique (à suivre dans le prochain numéro de MISC).

1. L'espace informationnel, enjeu de souveraineté

1.1 Un contexte nouveau

Le dernier *Libre blanc sur la défense et la sécurité nationale*, publié en juin 2008, fait désormais une large place à l'information numérique, aux systèmes d'information. Ce n'était pas le cas dans le précédent *Livre blanc* de 1994 **[4]**. À nouvelle époque, nouvelle vision du monde, nouveau contexte, nouvelles contraintes. Le monde nouveau, c'est la fin de l'après guerre froide marquée par les attentats du 11 septembre 2001, la mondialisation qui « *ne crée un monde ni meilleur ni plus dangereux…* », mais « *nettement plus instable* » (à preuve le choc financier de l'automne 2008), un monde qui reste dominé par la puissance des États-Unis, mais avec un rééquilibrage au bénéfice de l'Asie, la multiplication des menaces (terrorisme, missiles balistiques, attaques contre les systèmes d'information, espionnage, réseaux du crime organisé, risques naturels) qui a également fait disparaître la distinction entre sécurité intérieure et extérieure, une nécessaire approche globale des problèmes, davantage de complexité et d'incertitude qui rendent notre environnement et ses menaces difficilement appréhensibles, l'augmentation des dépenses militaires dans le monde, un système de sécurité collective fragile. Ce monde nouveau, c'est aussi celui de l'affirmation du cyberespace comme système vital, système nerveux de notre modèle de société. L'information y est diffusée plus largement, plus rapidement, avec pour conséquences une accélération de l'action, une augmentation de la puissance des médias **[5]**, un flux non maîtrisé des idées, notamment celles de la contestation idéologique, religieuse, radicale, un pouvoir accru des acteurs non étatiques, et une réduction de l'expression de la capacité de contrôle et de la souveraineté des Etats : « *L'accélération foudroyante de la circulation de l'information… fragilise la capacité d'intervention autonome des États* »**[6]**.

> *L'information est diffusée plus largement avec pour conséquences une réduction de l'expression de la capacité de contrôle et de la souveraineté des Etats...*

La société de l'information n'a donc pas apporté avec elle son seul lot de bienfaits et, si les aspects positifs structurent la société, les aspects négatifs quant à eux sont devenus une menace pour l'État lui-même. La sécurité de l'espace informationnel est une question politique, une question stratégique, une question de défense et de sécurité nationale. La sécurité des systèmes d'information (SSI) est, selon le rapport Lasbordes, « *un enjeu à l'échelle de la Nation toute entière* […] *Pour l'État il s'agit d'un enjeu de souveraineté nationale. Il a en effet la responsabilité de garantir la sécurité de ses propres systèmes d'information, la continuité de fonctionnement des institutions et des infrastructures vitales pour les activités socio-économiques du pays et la protection des entreprises et des citoyens* » **[7]**.

Le sentiment d'insécurité, la peur d'attaques et de déstabilisation des sociétés ont été exacerbés par l'affaire estonienne de 2007 qui est devenue la figure symbolique de cette menace contre le cyberespace. Si la menace semble immense, à en croire le discours sécuritaire officiel, elle n'en est pas pour autant totalement nouvelle. La cybercriminalité a depuis plus de dix ans déjà introduit le sentiment d'insécurité face aux réseaux. Quant à cette menace, de laquelle découleront ensuite les justifications d'un encadrement plus formalisé de l'utilisation des réseaux, elle est déjà inscrite, de manière visionnaire, dans un texte d'Anatole France publié en 1905 : « *La télégraphie et la téléphonie sans fil étaient alors en usage d'une extrémité de l'Europe à l'autre et d'un emploi si facile que l'homme le plus pauvre pouvait parler, quand il voulait et comme il voulait, à un homme placé sur un point quelconque du globe. […] Ce fut la suppression des frontières. Heure critique entre toutes ! […] La République française, la République allemande […] la suisse même et la belge, exprimèrent chacune, par un vote unanime de leur parlement et dans d'immenses meetings, la résolution solennelle de défendre contre toute agression étrangère le territoire national et l'industrie nationale. Des lois énergiques furent promulguées,[…] réglementant avec sévérité l'usage du télégraphe sans fil* » **[8]**. Il n'y est bien sûr point question de guerre de l'information, ni de cyberespace, mais on y perçoit déjà la formulation de nos préoccupations que l'on pense parfois modernes sur la sécurité. Y est également décrite l'attitude des gouvernants pour maîtriser la sphère informationnelle.

1.2 Pourquoi sommes-nous sensibles à la menace ?

La France, dans cet espace informationnel, serait en situation de vulnérabilité **[9]**. À cela, il y aurait plusieurs raisons :

- Parce qu'elle serait mal préparée pour affronter les menaces qui pèsent sur elle : la France serait en retard, les pays voisins feraient mieux, seraient mieux défendus.
- Parce qu'en France la culture de la sécurité ferait défaut.
- Parce que les politiques pourtant initiées voici près de 20 ans, notamment dès 1986 **[10]** avec une série de textes réglementaires organisant la sécurité des systèmes d'information, puis en mars 2004 avec l'adoption d'un plan triennal de renforcement de la sécurité des systèmes d'information de l'Etat **[11]**, n'auraient pas eu jusqu'ici l'efficacité escomptée.
- Par manque d'autorité ou de crédibilité des structures responsables.
- Par manque de moyens. Cet argument est le principal du rapport du Sénateur Romani. Pourtant, le nombre de structures, acteurs, programmes, opérations traitant à divers niveaux de la sécurité des systèmes d'information est impressionnant : la DCSSI (SGDN) qui intègre le COSSI (Centre Opérationnel de la Sécurité des Systèmes d'Information), lequel à son tour intègre le CERTA (le CERT français) **[12]**, le CELAR (DGA), la DGSIC (Direction Générale des Systèmes d'Information et de Communication), la DGSE (Direction Générale de la Sécurité Extérieure), la DPSD (Direction de la Protection et de la Sécurité de la Défense) , la DCRI (Direction Centrale du Renseignement Intérieur, ex-DST), l'OCLCTIC, un haut-fonctionnaire de défense rattaché auprès de chaque ministère, des fonctionnaires de sécurité des systèmes d'information (FSSI), des AQSI (autorités qualifiées en sécurité des

systèmes d'information), mais encore le CERT-IST [13], le CERT-Renater [14], le CERT-LEXSI [15], le plan PIRANET, des réseaux sécurisés comme RIMBAUD [16], ISIS [17], MAGDA [18], etc. Toute cette architecture est relayée ou coordonnée avec des initiatives internationales : travaux de l'ONU, de l'UIT, de l'OCDE, du G8, de l'OTAN, de l'UE, coordination des CERT via le FIRST, l'ENISA [19]...

- Par manque de coordination des moyens existant. « *Assurer notre défense et notre sécurité nécessite de percevoir puis de comprendre les dangers et les menaces* ». Or, malgré cette pléthore d'acteurs, il semble que « *la France est, aujourd'hui, dépourvue des outils nécessaires lui permettant d'appréhender, d'analyser et de traiter tout ce que l'on entend aujourd'hui par 'sécurité globale'* » [20].

La sensibilité à la menace n'est bien entendu pas le seul fait du système français. Le rapport Romani sur la cyberdéfense [21] dresse un inventaire de ces raisons communes à tous les États modernes : interconnexion des systèmes, leur relation à l'Internet, caractère contaminant de l'internet (tout ce qui entre en contact avec Internet devient faillible), dépendance de la société aux systèmes d'information, perméabilité de l'espace informationnel en raison des équipements mobiles (menace à l'intégrité des réseaux), faiblesses du protocole internet, utilisation d'applications (de briques sur étagères) qui ajoutent à la complexité et aux failles de sécurité, contamination des briques les plus solides par les briques les plus faibles, etc.

2. Identifier et désigner la menace

Les rapports et livres blancs publiés ces dernières années évoquent l'affaire estonienne (printemps 2007), les cyberattaques dénoncées par les États-Unis, la Nouvelle-Zélande, l'Allemagne, le Royaume-Uni, les atteintes subies par la France (le CEA – Commissariat à l'Énergie Atomique – victime en 2006 [22], les mails des diplomates français piratés ces derniers mois...). Ces incidents auraient « *matérialisé de manière très concrète une menace encore mal identifiée sur notre continent, particulièrement en France* » [23].

2.1 La figure de l'ennemi

2.1.1 Les agressions

« *Les attaques sont une réalité* » [24], la menace qui pèse sur les systèmes d'information et la sécurité nationale est présentée comme une « *évidence* » [25], les agressions quotidiennes subies par les systèmes d'information iront croissantes et la forte probabilité « *d'attaques informatiques majeures* » dans les 15 années à venir contre les systèmes d'information du pays est là encore une évidence [26]. Ces attaques informatiques majeures sont inscrites au nombre des 6 scénarios illustratifs proposés par le *Livre blanc* de 2008, faisant l'objet des mêmes égards (ou inquiétudes) que les menaces pouvant impliquer l'OTAN, les pandémies massives à forte létalité, les catastrophes naturelles, les crises dans un département d'outre-mer, et l'engagement de la France dans un conflit régional majeur. Le scénario sur les menaces ou conflits pouvant impliquer l'Alliance Atlantique [27] considère la cyberguerre comme l'un des moyens d'attaque permettant de contourner les défenses classiques des Alliés.

...les attaques informatiques majeures sont à « probabilité forte » et d'« ampleur faible à forte »...Les atteintes aux systèmes d'information peuvent ainsi être considérées comme des actes d'agression,...

Sur une échelle de hiérarchisation des risques et menaces, les attaques informatiques majeures sont dites à « *probabilité forte* » (au même niveau que les attaques terroristes, la criminalité organisée) et d'« *ampleur faible à forte* » (n'atteignant donc jamais sur cette graduation le niveau « *sévère* » réservé au terrorisme, aux pandémies, aux catastrophes naturelles et aux armes balistiques) [28]. Les atteintes aux systèmes d'information peuvent ainsi être considérées comme des actes d'agression, objet de crises et de conflits majeurs.

2.1.2 Définition de « l'attaquant »

« *Il est convenu d'appeler 'attaquant' toute personne physique ou morale (État, organisation, service, groupe de pensée, etc.) portant atteinte ou cherchant à porter atteinte à un système d'information, de façon délibérée et quelles que soient ses motivations* » [29]. Cette définition large de l'attaquant regroupe ainsi le délinquant, le cybercriminel, le terroriste, le réseau mafieux, l'agresseur militaire. Le caractère déterminant n'est pas a priori l'identité de l'individu ou du groupe, mais :

- L'acte commis : l'atteinte réalisée de manière intentionnelle.
- Les objectifs de l'attaquant : désinformer, empêcher l'accès à une ressource, prendre le contrôle du système, récupérer de l'information, utiliser le système pour des attaques rebond. L'attaque peut être motivée par des intérêts ludiques, cupides, terroristes, stratégiques (États, groupes).

Le rapport Romani dresse le portrait des attaquants. Il s'agirait selon lui d'individus ou groupes :

- Professionnels : « *à l'évidence les attaques informatiques actuelles ne peuvent être imputées à de simples amateurs* » [30] parce qu'elles se font plus ciblées et utilisent des techniques plus sophistiquée.
- Agissant en réseaux.
- « *groupes organisés, voire (...) services de renseignement* » [31].
- Motivés par l'argent, puisqu'ils revendraient leurs services à des États.
- Cyberterroristes [32] ayant recours aux services de la cybercriminalité, même si « *on sait [...] que les organisations terroristes ont acquis une maîtrise significative des outils informatiques* »
- Maîtrisant l'utilisation de « l'arme » informatique.
- Bien identifiés par la presse internationale.
- Localisés en Chine [33], en Russie.

Chacune des lignes de cette description est bien sûr discutable.

- Seuls des « professionnels » seraient-ils capables d'utiliser des technologies « sophistiquées » ? La marque du professionnel doit-elle être l'utilisation des méthodes sophistiquées et l'agissement en réseaux ?
- Qui détient des preuves du recours par les États à des réseaux mercenaires, d'une relation entre des gouvernements, des groupes terroristes et leur main armée que serait devenue la cybercriminalité organisée ? La presse, qui « *a fait état de l'existence de tels groupes en Russie* » [34] ?
- Les États motivés n'ont-ils pas les moyens de développer eux-mêmes leurs moyens d'agression et doivent-ils donc avoir recours à une forme de sous-traitance criminelle ?
- Quant à la notion « d'arme » informatique, pourquoi n'est-elle pas définie, alors qu'elle est pourtant lourde de conséquences sur la manière d'appréhender les mesures de sécurité et les moyens d'affronter les menaces : à quel moment l'outil informatique devient-il une arme ?
- La vision de la répartition de la menace dans le monde n'est-elle pas conditionnée par les médias et par le discours américain, qui désignent la Chine et la Russie comme États agresseurs majeurs. L'agresseur n'est-il plus américain ? Les grandes antennes qui interceptent les communications des satellites se sont-elles éteintes ? Les sous-marins qui interceptent les communications des câbles internet sous-marins restent-ils au port ? Les États les plus respectueux du droit ne savent-ils pas, eux non plus, se servir de ces technologies « *si peu détectables comme arme de renseignement* » [35] ?

La vision de la répartition de la menace dans le monde n'est-elle pas conditionnée par les médias et par le discours américain...

2.1.3 La menace terroriste

Tout le monde en parle, mais « *il faut souligner que cela n'a encore jamais été rapporté* » [36]. Le cyberterrorisme est alors objet de scénario. Le *Livre blanc du gouvernement sur la sécurité intérieure face au terrorisme* [37], texte de doctrine dans le prolongement de la loi du 23 janvier 2006 relative à la lutte contre le terrorisme, propose un scénario intitulé « Attentats diversifiés transfrontaliers » [38] qui inscrit les cyberattaques à son programme : l'une des équipes terroristes impliquées dans des attentats tente de désorganiser les secours en s'attaquant aux systèmes informatiques. Le plan PIRANET [39] serait mis en œuvre pour contrer cette cyberattaque.

Plus simplement, le terrorisme est perçu comme un utilisateur de l'internet, qui profite de l'anonymat qui y est offert. Pour lutter contre le terrorisme, « *les services de renseignement doivent pouvoir identifier et sélectionner les informations dignes d'intérêt dans la masse de celles disponibles sur le volet ouvert d'internet. Ils doivent aussi pouvoir accéder, sous certaines conditions, à celles qui circulent sur le volet fermé* » [40]. Ces mesures, que certains qualifient de « liberticides », s'insèrent dans la construction plus large d'un système de surveillance dont les fins sont officiellement justifiées : autoriser l'accès des services de renseignement aux fichiers gérés par le ministère de l'Intérieur (cartes d'identité, passeports, visas, titres de séjour, cartes grises, permis de conduire), aux fichiers des compagnies aériennes, maritimes, ferroviaires, autoriser le croisement de toutes ces données entre elles ou avec des données biométriques, de vidéosurveillance, permettre leur conservation dans un fichier unique pour plus d'efficacité [41], etc. La guerre contre le terrorisme passe par la maîtrise de l'information.

A suivre... ■

Les références de cet article sont disponibles sur miscmag.com/ref42.

Éric Lacombe – goretux@gmail.com, eric.lacombe@security-labs.org

INTRODUCTION À LA VIRTUALISATION

mots-clés : *virtualisation de systèmes / paravirtualisation / virtualisation matérielle / hyperviseur*

Nous expliquons dans cet article le concept de « virtualisation », pour ensuite nous intéresser plus particulièrement à la virtualisation de systèmes. Une classification est alors présentée. Il s'ensuit une introduction aux extensions matérielles développées par Intel et AMD pour faciliter la virtualisation de leurs architectures.

Les systèmes informatiques sont construits selon une hiérarchie d'interfaces qui séparent différents niveaux d'*abstractions*. Chaque couche d'abstraction propose, au niveau supérieur, une interface qui permet de simplifier l'interaction avec le niveau inférieur. Par exemple, au niveau du système d'exploitation, les *fichiers* constituent une abstraction qui rend possible une interaction simplifiée avec la mémoire de masse pour les applications.

Définissons à présent le concept de « virtualisation ». On virtualise un composant (comme un processeur, la mémoire ou un périphérique d'E/S) à un niveau d'abstraction donné lorsqu'on propose une interface et des ressources, construites à partir des siennes, aux composants se situant au-dessus de ce niveau d'abstraction. Le composant virtualisé est alors vu comme un ou plusieurs composants virtuels. Ces derniers proposent soit une interface et des ressources différentes du composant virtualisé (on parle dans ce cas d'émulation) ou bien proposent une interface et des ressources identiques. Il est important de noter que la virtualisation se distingue de la notion d'« abstraction » (ou couche d'abstraction), car son rôle n'est pas de simplifier l'interaction avec le système sous-jacent, mais d'en proposer une vision différente. Un exemple de virtualisation très répandu dans la plupart des OS actuels est celui de la mémoire. Cette virtualisation s'opère via la mise à disposition d'un espace d'adressage identique pour chaque processus (grâce à l'unité matérielle de pagination que l'on retrouve dans la plupart des architectures). Enfin, dans le cas où le composant virtualisé est un système complet, on appelle ce composant virtuel, une *Machine Virtuelle* (VM – *Virtual Machine*).

1. Les deux classes principales de virtualisation

La virtualisation se découpe généralement en deux classes suivant qu'elle s'effectue au niveau applicatif ou au niveau système. Dans le premier cas, il s'agit de virtualiser uniquement l'environnement des applications par le biais d'un virtualiseur nommé *runtime software* comme la JVM (*Java Virtual Machine*), laquelle exporte de surcroît sa propre ISA (*Instruction Set Architecture*). Les applications qui s'exécutent dans cet environnement disposent alors chacune d'un ensemble privé de ressources, toujours agencées de la même façon, mais dont les quantités peuvent varier. Dans le cas de la virtualisation de système, il s'agit de virtualiser complètement l'environnement matériel au sein d'une machine virtuelle pour qu'elle puisse accueillir un système d'exploitation au complet (ou au minimum son « espace utilisateur » comme Linux-Vserver). Les virtualiseurs mis en jeu dans ce type de virtualisation sont couramment appelés *Virtual Machine Monitor* (VMM) ou encore *hyperviseur*. Dans la suite de l'article, nous analysons uniquement le cas de la virtualisation de systèmes.

2. La virtualisation de systèmes

Nous venons de voir qu'un hyperviseur met en place une ou plusieurs machines virtuelles dans lesquelles sont exécutés des systèmes d'exploitation. Ces systèmes sont alors appelés « systèmes invités » (*guest system*).

Rappelons qu'une machine virtuelle doit proposer une virtualisation des différents éléments matériels nécessaires à l'exécution d'un système d'exploitation. Les différents composants à virtualiser sont la CPU, la mémoire et les périphériques d'E/S.

2.1 Les deux types d'hyperviseurs qui mettent en œuvre cette virtualisation

La virtualisation de systèmes est mise en œuvre au travers de la gestion de machines virtuelles, orchestrée par l'un des deux types suivants d'hyperviseurs.

2.1.1 Hyperviseur de type I

Un hyperviseur de type I s'exécute directement sur le matériel et propose des machines virtuelles aux systèmes invités en s'appuyant uniquement sur l'interface et les ressources matérielles sous-jacentes. Il doit ainsi implémenter, entre autres, la gestion mémoire complète des machines virtuelles et leur ordonnancement. En d'autres termes, il doit implémenter la plupart des services que fournissent les noyaux de systèmes d'exploitation courants. Les hyperviseurs de ce type réduisent toutefois leur complexité, en employant souvent une machine virtuelle privilégiée qui abrite les pilotes des périphériques du système (et autorise également le contrôle de l'hyperviseur).

2.1.2 Hyperviseur de type II

Un hyperviseur de type II, aussi appelé *host-based hypervisor*, s'installe sur un système d'exploitation (lequel est alors appelé « système hôte ») et profite de ses abstractions et fonctionnalités pour créer les environnements virtuels. Pour la plupart, l'abstraction des processus des OS sert de support aux machines virtuelles et ainsi l'ordonnancement de ces machines est effectué directement par l'ordonnanceur de l'OS. La gestion mémoire s'appuie également sur les services de l'OS, simplifiant par conséquent l'implémentation d'un hyperviseur de ce type.

2.2 Les trois grandes classes de virtualisation de systèmes

2.2.1 La virtualisation complète

La virtualisation complète ou virtualisation native fournit un environnement virtuel (interface et ressources) censé représenter une architecture réelle et cela sans aucune simplification (ou plus généralement sans aucune modification) de l'interface et des ressources associées à cette architecture. Ainsi, un système d'exploitation développé pour être exécuté sur une architecture particulière, s'exécute de façon native (c'est-à-dire sans aucune modification) dans une machine virtuelle qui virtualise complètement cette architecture. Lorsque l'architecture virtualisée est différente de l'architecture matérielle sous-jacente, on qualifie la virtualisation d'émulation matérielle, car il s'agit alors d'une *imitation* d'architecture construite à partir d'une autre architecture.

Comme solution de virtualisation complète, on remarque par exemple :

- *KVM* (*Kernel Based Virtual Machine*) qui est un hyperviseur de type II fonctionnant sous Linux et qui propose une virtualisation complète de l'architecture IA-32 et IA-64 ;
- *Qemu* qui virtualise complètement de multiples architectures ;
- VMware ESX et VMware Workstation respectivement de type I et II.

2.2.2 La paravirtualisation

La paravirtualisation ne cherche pas à présenter une architecture matérielle complète aux systèmes invités. À la différence de la virtualisation complète où le système invité croit idéalement s'exécuter à même le matériel, elle instaure une collaboration entre l'hyperviseur et ses systèmes invités, afin d'en faciliter leur gestion et ainsi d'obtenir de très bonnes performances d'exécution. Cela entraîne bien évidemment la modification des systèmes invités qui font appel à l'hyperviseur pour effectuer des opérations privilégiées sur le matériel (cas par exemple des systèmes invités Linux utilisant les opérations de l'infrastructure générique `paravirt_ops` – laquelle remplace les opérations privilégiées d'un OS par des appels à l'hyperviseur – afin de communiquer avec l'hyperviseur) ou bien faciliter l'interception de ces opérations (par exemple pour l'architecture x86, en plaçant leur noyau dans le ring 1 moins privilégié que le ring 0).

Ces approches donnent de bonnes performances avec des architectures matérielles qui ne sont pas virtualisables telles quelles, comme le x86 d'origine (avant l'arrivée des extensions matérielles de virtualisation). Par exemple, le x86 ne définit

pas `sidt` (instruction qui écrit en mémoire la valeur du registre `idtr`) comme une instruction privilégiée, et empêche ainsi son interception par l'hyperviseur. Cette interception est cependant nécessaire à une virtualisation complète, car l'architecture x86 ne dispose que d'un seul registre `idtr`, et virtualiser ce registre suppose d'intercepter les différents accès qui peuvent s'y produire. Pour pallier les déficiences de ce type d'architecture en matière de virtualisation, des traitements logiciels lourds sont nécessaires si l'on ne souhaite pas mettre en place de la paravirtualisation (parce que l'on ne souhaite pas faire confiance à l'OS invité par exemple). Certaines instructions du code du système invité peuvent être par exemple remplacées en mémoire avant exécution afin que l'hyperviseur puisse les intercepter.

On trouve, par exemple, comme hyperviseurs appartenant à cette classe :

- Xen et lguest qui emploient `paravirt_ops` ;
- User Mode Linux qui relocalise le noyau Linux sur x86 en ring 1 ;
- Qemu avec le module noyau kqemu ;
- VMware Workstation associé aux VMware tools.

Notons que kqemu et les VMware tools sont en gros des modules noyau qui s'installent sur le système invité et qui remplacent les opérations privilégiées de ce système par des appels directs à l'hyperviseur. Ainsi, Qemu et VMware Workstation associés à leur module noyau respectif fournissent la paravirtualisation.

2.2.3 La virtualisation au niveau du système d'exploitation

Également appelée « virtualisation par *containers* », la virtualisation au niveau du système d'exploitation autorise l'instanciation de multiples « espaces utilisateurs » sur un même noyau, et de les isoler les uns des autres par le biais de différents espaces de noms (chaque container dispose de son propre espace de PID par exemple). Dans ce type de virtualisation, il s'agit de rendre la plupart des structures du noyau instanciables afin de renforcer l'illusion de l'existence de multiples machines. Les *Jails* de BSD peuvent être vus comme des précurseurs des containers.

Des solutions de ce type de virtualisation sont par exemple :

- les *Zones* de Solaris ;
- Linux Vserver ou OpenVZ/Virtuozzo, lesquels peuvent faire tourner différentes distributions GNU/Linux sur un même noyau.

À noter toutefois que l'isolation entre les « machines virtuelles » de ce type de virtualisation dépend du niveau d'« instanciabilité du noyau » (afin qu'il y ait le moins d'interactions possibles au sein du noyau entre les différentes machines virtuelles).

3. Le support matériel pour la virtualisation

Nous avons mentionné le fait que certaines architectures comme le x86 d'origine ne sont pas virtualisables de façon native sans un lourd traitement logiciel. Intel et AMD ont pallié cela en ajoutant des extensions matérielles facilitant la virtualisation complète de leurs architectures. Ces extensions sont donc principalement employées dans le cas de la virtualisation complète (KVM et Xen par exemple), mais peuvent également être mises à profit dans la paravirtualisation (par exemple, une instruction supplémentaire est définie par Intel afin de faire appel directement à l'hyperviseur depuis un système invité). Nous abordons par la suite uniquement le cas d'Intel avec ses extensions VT-x (pour la virtualisation du processeur et de la mémoire – support matériel pour les *shadow page tables*) et VT-d (pour la virtualisation des E/S) sur architecture IA32, les technologies d'AMD étant similaires.

3.1 La technologie de virtualisation VT-x

L'extension matérielle VT-x rend possible la virtualisation partielle des processeurs de type IA32 (cf. la section suivante sur VT-d). Cette extension supporte deux types de logiciels : l'hyperviseur qui se comporte comme un hôte réel et qui a le contrôle complet du processeur et des autres parties matérielles ; et le système invité, qui est exécuté dans une machine virtuelle. Chaque machine virtuelle s'exécute indépendamment des autres et utilise la même interface pour le processeur, la mémoire, la mémoire de masse, la carte graphique et les E/S fournies par la plateforme physique.

> *...Le support processeur pour la virtualisation est fourni par une forme d'opérations processeur appelées « opérations VMX »...*

Le support processeur pour la virtualisation est fourni par une forme d'opérations processeur appelées « opérations VMX ».

Il y a deux types d'opérations VMX : les opérations *VMX root*, pour l'exécution de l'hyperviseur, et les opérations *VMX non-root* pour l'exécution des systèmes invités. Le comportement du processeur en est quasiment le même dans les deux modes avec la différence qu'un ensemble de nouvelles instructions est disponible. Le comportement du processeur en mode VMX non-root est restreint et modifié pour faciliter la virtualisation. À la place de leur comportement habituel, certaines instructions et évènements causent des transitions vers l'hyperviseur, appelées *VM-exits*. Comme ces VM-exists viennent se substituer au comportement habituel, les fonctionnalités du logiciel en mode VMX non-root sont donc limitées. Ces limitations permettent à l'hyperviseur de garder le contrôle des ressources du processeur. Comme les opérations VMX placent des restrictions même sur le logiciel qui s'exécute au niveau le plus privilégié (le ring 0), le logiciel en mode invité peut s'exécuter au même niveau de privilège que celui pour lequel il a été conçu à l'origine. Cette particularité peut notamment simplifier le développement d'un hyperviseur.

3.2 La technologie de virtualisation VT-d

VT-d ou *Virtualization Technology for Directed I/O* permet la virtualisation des E/S sur les architectures IA32. Ainsi, l'hyperviseur peut contrôler l'accès des périphériques à la mémoire principale, et l'accès des machines virtuelles aux périphériques. Il peut notamment associer exclusivement certains périphériques à une machine virtuelle en particulier. Afin de parvenir à ce résultat, VT-d implémente une I/O MMU. Il s'agit d'une unité de gestion mémoire (MMU – *Memory Management Unit*) qui connecte un bus d'E/S (supportant le DMA – *Direct Memory Access*) à la mémoire principale. De la même façon qu'une MMU traditionnelle, la I/O MMU, s'occupe de la correspondance entre les adresses d'E/S et les adresses physiques de la mémoire. Les tables de traductions se trouvent dans la mémoire principale et doivent être maintenues par l'hyperviseur.

Grâce à cette unité, l'hyperviseur est capable de proposer une configuration des transferts DMA, identique à toutes les machines virtuelles. En effet, les adresses mémoire « physiques » que fournit un système invité à un périphérique capable de DMA peuvent être relogées ailleurs en mémoire, par le biais d'une traduction d'adresses qui s'effectue via la I/O MMU et sous le contrôle de l'hyperviseur.

Conclusion

La virtualisation est un domaine assez ancien. Les premiers travaux datent des années 70. L'objectif était alors d'exploiter les énormes ressources des mainframes. Mais, l'arrivée des ordinateurs personnels a vu la recherche dans ce domaine disparaître progressivement. L'engouement actuel pour la virtualisation s'explique par l'apparition de nouveaux domaines d'application comme la sécurisation des systèmes informatiques ou encore l'amélioration de la flexibilité de la gestion des ressources pour les serveurs.

Dans la suite de ce dossier sur la virtualisation, nous introduisons l'architecture de la solution de virtualisation de Microsoft nommée Hyper-v, pour ensuite nous approprier le fonctionnement de l'hyperviseur Xen. Ce voyage se poursuit avec quelques bonnes pratiques pour la virtualisation de serveurs avec VMware ESX, et s'achève sur un développement sur les techniques de détection d'exécution au sein de machines virtuelles, lesquelles sont applicables à la détection de la présence de *rootkits* hyperviseur. ■

ivanlef0u@security-labs.org

INSIDE MICROSOFT HYPER-V

mots-clés : *Hyper-V / virtualisation / hyperviseur / hypercalls*

Hyper-V est le dernier-né des produits de virtualisation de Microsoft. Il se veut être un hyperviseur orienté serveurs utilisant les dernières technologies hardware de virtualisation. C'est aussi la figure de proue de Microsoft sur le marché grandissant de la virtualisation. Cet article a pour but de présenter en profondeur l'architecture d'Hyper-V, notamment l'utilisation d'API appelées « hypercalls ».

1. Introduction

Ce document visant un public aussi large que possible, nous commencerons par faire quelques rappels sur la virtualisation et les technologies qu'elle met en œuvre. Ceux qui sont déjà à l'aise avec le sujet peuvent sauter cette section. Le lecteur est supposé avoir des bases sur l'architecture x86 et celle des systèmes d'exploitation.

1.1 Windows Server 2008

Tout d'abord, quelques mots sur l'OS qui supporte Hyper-V : tout comme Windows Vista [1], Windows Server 2008 [2] repose sur un noyau NT 6.0 SP1. Il est donc aussi bien disponible pour les processeurs x86 que x64. La RTM (*Release To Manufacturing*) fut officiellement lancée en février 2008. La version R2 Bêta a été lancée en janvier 2009. Windows Server 2008 est le successeur de Windows Server 2003 [3], il est donc principalement destiné au marché professionnel.

Windows Server 2008 introduit notamment le concept de « *Server Manager* » [4], qui unifie la gestion des ressources de la machine en proposant une interface standard sous forme de rôles et de *features*. Un rôle définit une fonction basique du serveur qui regroupe des services, des évènements, des ressources en ligne comme des *best-practices*. Une feature décrit plutôt une fonction secondaire de la machine. Par exemple, BitLocker [5] (outil de chiffrement de disque) est une feature alors que le serveur Web IIS [6] est un rôle.

Bien sûr, il existe de nombreuses autres fonctionnalités introduites par Windows Server 2008, mais celles-ci n'ont pas leur place dans cet article.

1.2 Hyper-V

Hyper-V [7] est un hyperviseur 64 bits uniquement disponible sous Windows Server 2008 x64. La RTM de Hyper-V a été lancée en juin 2008 et la dernière version est parue dans la Bêta du SP2 en décembre 2008. Durant la même période, une version « *stand-alone* » est sortie, appelée « Microsoft Hyper-V Server 2008 ». Il s'agit d'un Windows Server 2008 « *Core* » possédant donc le strict minimum pour faire tourner Hyper-V et s'administrant via la ligne de commande Windows et des scripts PowerShell. Cette version d'essai gratuite permet donc de manipuler Hyper-V, mais est réservée aux plus courageux ! (L'auteur de cet article tient à préciser qu'il n'est pas courageux).

Les principaux points-clés de Hyper-V sont :

- virtualisation à l'aide des jeux d'instructions SVM et VMX sur architecture x64 ;
- support des *guests* 32 bits (x86) et 64 bits (x64) ;
- support de mémoire large, jusqu'à 64 Go par VM ;
- support du SMP (*Symmetric Multiprocessing*) pour les VM (jusqu'à 4 cœurs) ;
- gestion des « *snapshots* » ;
- support des disques, du réseau, des entrées souris/clavier et de la vidéo ;
- services d'intégration des composants ;
- mise en place de VLAN et de NLB (*Network Load Balancing*) ;
- gestion avancée du partage des périphériques.

Hyper-V se présente donc sous la forme d'un rôle pour Windows Server 2008. La figure 1 montre les différents composants du rôle Hyper-V.

Hyper-V n'affecte en rien le fonctionnement des autres rôles. Il assure uniquement la bonne marche des machines virtuelles, ce qui veut dire qu'il est possible d'avoir une VM Windows Server 2003 qui propose des services sur le réseau sans que l'hôte et son guest ne se connaissent.

1.3 Gestion des machines virtuelles

La gestion des machines virtuelles de Hyper-V ressemble beaucoup à celle qu'on trouve sur des produits concurrents comme VMWare Workstation [8]. Cet article n'a pas pour but de résumer les différentes fonctions de gestion des VM sous Hyper-V. Qui plus est, le net regorge de bonne documentation sur le sujet [9].

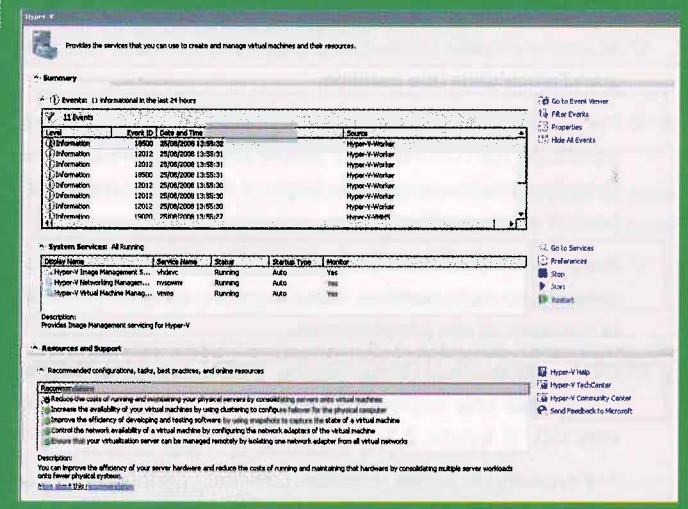
Hyper-V propose des *Integration Components* pour ses guests. Ce sont l'équivalent des VMware Tools. Comme avec VMware, il s'agit d'une image ISO qu'on retrouve dans `%systemroot%\system32\vmguest.iso`. Ces services sont :

- Synchronisation du temps (*Time Synchronization*) : permet de donner une vision du temps uniforme à chaque VM.
- *Heartbeat* : mécanisme qui permet à Hyper-V de savoir si un guest est toujours actif.
- Arrêt (*Shutdown*) : permet d'arrêter un guest de manière correcte à l'aide des fonctions standards comme `NtShutdownSystem`. Cela évite de « couper » le fonctionnement du guest de manière brutale.
- Échange de clés de registre (*Key/Value Pair Exchange*) : le guest et l'hôte partagent des clés permettant de connaître leurs versions et leurs architectures.
- *Volume Shadow Copy Service* (VSS) [10] : lors d'un *back up* de l'hôte avec VSS, ce service s'active aussi pour les VM qui le supportent.

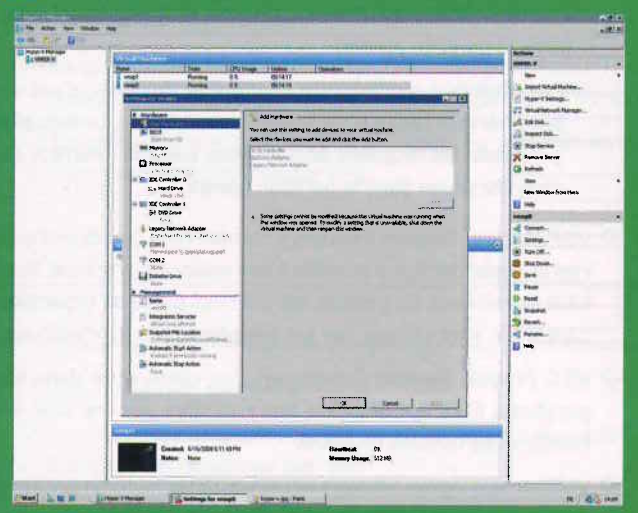
Les *Integration Services* servent aussi à optimiser les performances. Par exemple, ils permettent à l'hyperviseur d'optimiser l'utilisation des TLB (*Translation Lookaside Buffer*) en lui fournissant un ensemble d'entrées à *flusher* plutôt que de tout flusher lors du passage dans l'hyperviseur [11].

Cependant, la version actuelle de Hyper-V ne supporte complètement que les OS suivants comme guest :

- Windows XP SP3 x86 et x64 ;
- Windows Vista SP1 x86 et x64 ;
- Windows Server 2000 SP4 ;
- Windows Server 2003 SP3 x86 et x64 ;
- Windows Server 2008 x86 et x64 ;
- Suse Linux Entreprise Server 10 SP1 x86 et x64.



1 *Hyper-V et son rôle dans Windows Server 2008*



2 *Paramétrages d'une VM Hyper-V*

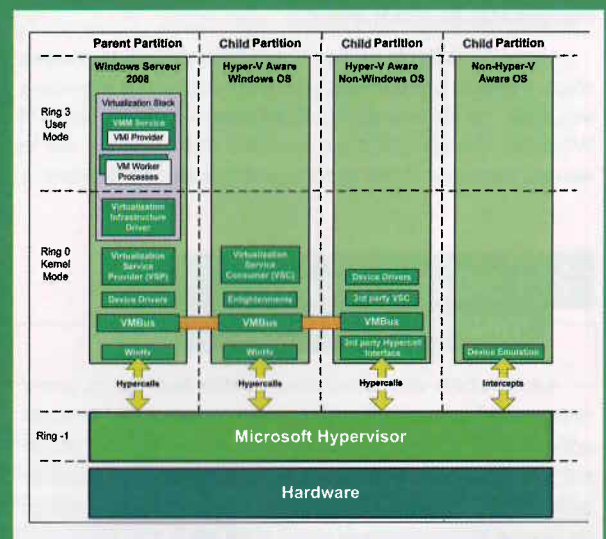
2. Architecture de Hyper-V

Hyper-V possède la même architecture que Xen, les guests accèdent au matériel en passant par les pilotes d'un guest administrateur. Même si Microsoft appelle cela une architecture « *micro-kernel* », il s'agit bien d'un hyperviseur monolithique [12].

Commençons par définir quelques composants propres à Hyper-V [13] :

- Hyperviseur (*Hypervisor*) : couche logicielle située au-dessus du matériel. Son but premier est de fournir des environnements isolés du point de vue matériel et logiciel, qu'on appelle « partitions ». L'hyperviseur a tous les droits sur le matériel.
- Partition (*Partition*) : une partition représente pour Hyper-V une entité isolée. C'est un ensemble de ressources physiques constitué de processeurs logiques, de mémoire et de périphériques.
- Machine virtuelle (*Virtual machine*) : une machine virtuelle prend place dans une partition.
- Partition parent (*Parent partition*) : la partition parent est constituée par l'OS Windows Server 2008 x64. Elle contient l'interface d'administration de Hyper-V et possède donc tout pouvoir sur les partitions filles.
- Partition fille (*Child partition*) : une partition fille ne peut pas créer de nouvelle partition. Elle n'a pas accès directement à la mémoire, ni aux périphériques.
- Périphérique virtuel (*Virtual device*, VDEV) : un périphérique visible par une machine virtuelle qui va donc recevoir des I/O. Il existe 2 types de périphériques virtuels :
  - Périphérique émulé (*Emulated device*) : périphérique qui est montré à une machine virtuelle par l'hyperviseur. Il émule un périphérique physique existant, mais facilement reconnaissable par un guest. Cette émulation se fait dans l'un des *VM Worker Process* nommé `vmwp.exe` qui est situé dans la partition parent.
  - Périphérique synthétique (*Synthetic device*) : c'est un périphérique proxy pour la machine virtuelle. Lorsqu'elle y accède, les requêtes sont routées sur une interface de contrôle située dans la partition parent.
- VSP (*Virtual Service Provider*) : un serveur qui fournit une interface standard pour accéder à une ressource physique. Il est situé *kernel-land*. En général, les VSP font partie de la partition parent. Ils sont utilisés par les périphériques synthétiques.
- VSC (*Virtual Service Consumer*) : un client situé dans les partitions filles qui effectue des requêtes sur les VSP en fonction des besoins du guest.
- VMBus : bus logiciel à travers lequel les partitions communiquent avec un protocole basé sur des IPC et de la mémoire partagée.
- Hypercall : interface de communication standard avec l'hyperviseur. Les hypercalls sont basés sur les instructions `VMMCALL` (AMD) et `VMCALL` (Intel) des jeux d'instructions de virtualisation et permettent de sortir du guest (de manière contrôlée bien sûr :]).
- Éclairements (*Enlightements*) : améliorations dans une partition qui lui permettent d'être consciente de l'hyperviseur. Elles fournissent aussi des optimisations notamment sur la gestion de la mémoire du guest par l'hyperviseur. Ils sont installés à l'aide des Integration Services et permettent l'accès aux périphériques synthétiques.

Le schéma suivant résume les points que nous venons d'énoncer :



3 *Architecture de Hyper-V*

En fait, l'architecture de Hyper-V met en place ce qu'on appelle une pile de virtualisation (*Virtualization Stack*) [14]. Cette pile désigne l'ensemble des composants qui servent à la gestion des machines virtuelles dans la partition parent. On y retrouve les composants suivants :

- Le service VMMS (*Virtual Machine Management Service*) qui est le service de contrôle de toutes les machines virtuelles. Il est là pour fournir une interface WMI à la console de gestion (*Management Console*) et pour créer les *Worker Processes* associés à chaque VM.
- Le *driver* de virtualisation de l'infrastructure (*Virtual Infrastructure Driver* ou VID). Il fournit une interface noyau de communication avec l'hyperviseur. Il a pour rôle aussi d'assurer l'émulation des composants bas niveau des VM comme la ROM du BIOS et les Memory Mapped I/O.
- Un *Worker Process* par VM. Ce processus contient l'état de la machine virtuelle, ainsi qu'une carte-mère virtuelle avec l'ensemble de ses VDev. Il fournit aussi la fonctionnalité de snapshot.

Pour résumer, Hyper-V met donc en place une architecture d'hyperviseur monolithique. L'hyperviseur n'est là que pour fournir un ensemble d'environnements isolés qu'on appelle des partitions et qu'on peut confondre avec des machines virtuelles.

Il existe une partition parent qui maintient l'état de toutes les VM, elles-mêmes placées dans des partitions filles. De cette notion de hiérarchie découle le fait que toutes les requêtes hardware doivent passer par la partition parent pour être traitées. Les partitions communiquent entre elles à l'aide d'un VMBus.

Les périphériques des VM peuvent être complètements émulés ou bien agir comme des proxies par rapport aux pilotes natifs.

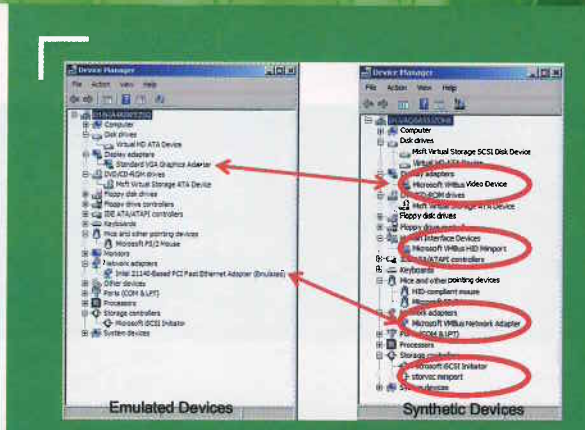
2.1 Périphériques synthétiques vs périphériques émulés

Les périphériques synthétiques sont la grande nouveauté de Hyper-V [15]. Par défaut, sans Integration Component, une VM possède les périphériques suivants qui sont émulés par son Worker Process dans la partition parent :

- driver de contrôleur de disque (Intel 440 BX Controller) ;
- driver de carte réseau (Intel/DEC 21140 Network Card) ;
- driver de carte vidéo (S3 Trio Video Card).

Dans la VM, on retrouve donc un driver natif standard qui croit communiquer avec un vrai périphérique, alors qu'en fait le comportement est émulé dans la partition parent. Ils sont utilisés pour les OS qui ne disposent pas des Integration Components.

Avec les Integration Components, les pilotes de périphériques deviennent des proxies s'interfaçant avec le VMBus à l'aide d'hypercalls, ce sont des VSC. Les VSC envoient leurs requêtes sur les VSP qui sont dans la partition parent. Ces derniers maintiennent un état de VDev pour chaque VM. Ils sont ensuite capables de transférer les I/O sur les pilotes ou services natifs de la partition parent.



4 *Différences entre les périphériques synthétiques et émulés*

Avec cette seconde architecture, on évite à l'hyperviseur de gérer les I/O effectuées par les pilotes natifs de VM. De plus, on évite un changement de contexte du kernel-land vers l'user-land pour se retrouver dans le Worker Process.

2.2 Boot

Quelques mots sur le démarrage de Hyper-V. Lors du *boot* de la machine, le driver `%systemroot%\system32\hvboot.sys` est lancé en tant que pilote prioritaire. Il a pour rôle de :

- Détecter si un hyperviseur est déjà lancé.
- Vérifier que le CPU supporte bien les extensions de virtualisation VMX et SVM. En fonction du CPU, `hvboot.sys` charge le driver correspondant :
  - Pour AMD : `%systemroot%\System32\Hvax64.exe` ;
  - Pour Intel : `%systemroot%\System32\Hvix64.exe`.

L'hyperviseur va donc virtualiser l'OS à la volée !

La commande `BCDEDIT` permet de définir les paramètres de démarrage de l'hyperviseur :

- Le paramètre `/set {current} hypervisorlaunchtype <auto|off|on>` permet de définir si l'on souhaite activer l'hyperviseur ou non.
- En utilisant `/hypervisorsettings`, on règle les paramètres de débogage de l'hyperviseur [16].

3. Hypercalls

Les hypercalls sont très importants pour Hyper-V, car ils assurent la communication d'une partition avec l'hyperviseur. Nous allons voir qu'il est possible de les utiliser de 2 manières. Mais d'abord, quelques mots sur leur fonctionnement.

Voici un tableau résumant les différences de terminologie entre un système « physique » et un système virtualisé (voir tableau page suivante).

Les hypercalls servent à :

- Ajuster l'activité du processeur en spécifiant par exemple la charge CPU maximale par VM.
- Gérer les quotas mémoire d'un guest.
- Envoyer des messages sur le VMBus pour les communications inter-partitions notamment en utilisant le SynIC.

Physique	Virtuel
Système & OS	Partition & OS invité
Processeur logique	Processeur virtuel
APIC(*Advanced Programmable Interrupt Controller*)	Virtual APIC + Contrôleur d'interruption Synthétique (SynIC)
Adresse virtuelle	*Guest Virtual Address* (GVA)
Adresse physique (*System Physical Address* ou SPA)	*Guest Physical Address* (GPA)

- Contrôler les interruptions virtuelles délivrées aux VM.
- Fournir un contrôle de l'état de la partition.
- Donner l'accès à l'état du processeur virtuel.

En fait, toute la pile de virtualisation de la partition repose sur l'utilisation des hypercalls pour administrer les VM. De la même manière, les drivers proxies des périphériques synthétiques vont utiliser des hypercalls pour communiquer sur le VMBus avec la partition parent.

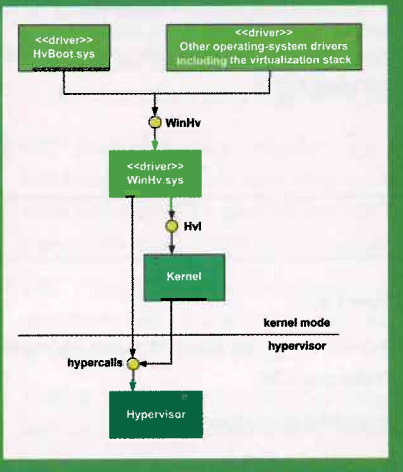
Les hypercalls ne sont disponibles qu'en ring 0 et peuvent s'utiliser de 2 façons :

1. De manière native : l'utilisateur met en place une interface autour des instructions `VMMCALL` ou `VMCALL` qui respecte les normes de l'hyperviseur.
2. À l'aide d'un *wrapper* fourni avec les Integration Components. Ce wrapper n'est disponible que pour les guests de la famille Windows NT. Il s'agit d'un driver nommé `WinHv.sys` qui exporte un ensemble de fonctions.

On peut ainsi schématiser les hypercalls par : voir Figure 5.

Les hypercalls sont documentés dans le WDK [17]. Les fonctions natives sont de la forme Hv* alors que les fonctions *wrappées* sont de la forme WinHv*.

En fait, il existe 2 types d'hypercalls, les hypercalls « simples » et les « répétitifs ». Un hypercall simple réalise une opération atomique au sein de l'hyperviseur. Un hypercall répétitif peut se transformer en hypercall simple. L'inverse n'est pas vrai. Quand on parle d'hypercall répétitif, cela veut dire qu'on peut effecteur plusieurs hypercalls simples dans un seul appel. Par exemple, l'hypercall natif `HvDepositMemory` [18] qui permet d'ajouter de la mémoire à une partition est répétitif, car il prend en paramètre un tableau d'adresses de *frames*.



5 *Architecture des hypercalls*

De plus, les hypercalls répétitifs peuvent être interrompus pendant l'appel. L'hyperviseur tente de réaliser les hypercalls en moins de 50 µs. Comme certains hypercalls répétitifs ne peuvent tenir dans cet intervalle, l'hyperviseur repasse dans la VM, afin de traiter les interruptions pendantes. Lors de cette action, le pointeur d'instruction est remis sur l'instruction `VMCALL` (ou `VMMCALL`). Ainsi, le même hypercall est appelé lorsque le traitement des interruptions du guest est terminé. C'est pour cela qu'il existe 2 valeurs pour les hypercalls répétitifs, une contenant le nombre total d'hypercalls à effectuer et l'autre le nombre d'hypercalls réalisés.

Nous allons maintenant nous intéresser à l'utilisation des hypercalls aussi bien à l'aide du wrapper `WinHv.sys` qu'avec l'interface native.

3.1 Utilisation des hypercalls

Il est très simple de savoir si un OS est virtualisé par Hyper-V. En effet, un appel à l'instruction `CPUID` avec le registre EAX à 1 renvoie le bit 31 d'ECX à 1 si Hyper-V est présent. Il est donc très facile pour un *malware* de détecter Hyper-V, mais il faut bien avoir en tête que Hyper-V n'a pas été conçu pour faire de la lutte anti-malware :]

L'instruction `CPUID` permet aussi d'obtenir des informations sur la version de l'hyperviseur.

Une fois que la présence de Hyper-V a été détectée, nous pouvons commencer à utiliser les hypercalls. Il existe 2 interfaces, l'interface native et le driver `WinHv.sys`. Commençons par cette dernière.

3.2 Hypercalls avec WinHv.sys

La dernière version du WDK (6001.18002 octobre 2008) ne fournit pas les *headers* `winhv.h` pour cette bibliothèque. En revanche, la version 6001.18000 (décembre 2007) les fournissait, mais sans le fichier `winhv.lib`. Il était donc possible de compiler un driver utilisant les WinHv*, mais on ne pouvait toujours pas le *linker*. Bien évidemment, cela ne suffit pas à nous arrêter :]

On sait que `winhv.sys` exporte des fonctions depuis sa table des exportations (EAT). On connaît leur prototype grâce à `winhv.h`. Il nous manque donc le fichier `.lib` pour permettre au linker de mettre en place la table des imports (IAT de notre driver). Il existe un outil, `lib.exe` (*Library Manager*) [19], qui permet de créer un fichier `.lib` à partir d'un fichier de définition `.def`.

Il nous suffit donc de *dumper* les fonctions exportées de `winhv.sys` avec un outil comme LordPE [20] et de créer un fichier `hyperv.def` de la forme :

```
LIBRARY winhv.sys
EXPORTS
        WinHvAdjustFeaturesState
        WinHvAllocatePartitionSintIndex
        WinHvAllocatePortId
        WinHvAllocateSingleSintIndex
        WinHvAssertVirtualInterrupt
        WinHvCancelTimer
[…]
```

Ensuite, nous utilisons `lib.exe` à travers la commande :

```
lib.exe /def:hyperv.def /OUT:hyperv.lib /MACHINE:x64 /VERBOSE
```

On obtient un joli `hyperv.lib` qui est reconnu par le linker `link.exe` de Microsoft [21]. Nous sommes donc en mesure de manipuler les API WinHv*.

Malheureusement, la documentation pour ces API a disparu dans la version 6001.18002 du WDK. On peut toujours l'obtenir dans la version 6001.18000.

3.3 Hypercalls natifs

Nous nous plaçons dans le cas où le driver `winhv.sys` n'est pas disponible soit parce que nous n'avons pas installé les Integration Components dans un guest Windows NT, soit simplement parce que nous ne sommes pas dans une VM Windows NT.

L'établissement d'une interface d'hypercall [22] repose essentiellement sur les MSR (*Model Specific Register*), car il est facile de les virtualiser et donc de les contrôler du point de vue de l'hyperviseur. On distingue en gros 3 étapes :

1. Détecter si l'hyperviseur est présent à l'aide de l'instruction `CPUID`.
2. Lire le MSR `HV_X64_MSR_HYPERCALL` (0x40000001). On obtient une structure de 64 bits qui possède les membres `Enable` et `GpaPageNumber`. Le bit `Enable` est normalement mis à jour par le code du guest pour savoir s'il a déjà mis en place cette interface. `GpaPageNumber` contient l'indice du frame que doit mapper le guest dans son espace virtuel pour effectuer des hypercalls.
3. Le guest mappe donc cette page dans son espace virtuel et met le bit `Enable` à 1. Cette page contient juste le code nécessaire pour réaliser un hypercall.

Pour information, le code qu'on trouve dans la page d'hypercall peut être obtenu avec le débogueur Microsoft KD :

```
// Lit le MSR HV_X64_HYPERCALL
kd> rdmsr 0x40000001
msr[40000001] = 00000000`025e1001

// Récupère le contenu du membre GpaPageNumber (bit 12:63 du MSR)
et le multiplie par la taille d'une page.
kd> ? (025e1001>>0xC)*0x1000
Evaluate expression: 39718912 = 025e1000

// Désassemble le code situé à l'adresse physique (GPA) 0x25e1000
kd> up 025e1000
025e1000 0f01c1    vmcall
025e1003 c3        ret
```

Comme on peut le constater, le code n'est qu'un simple appel à `VMCALL` avec un `RET`. Évidemment, on retrouve soit `VMCALL`, soit `VMMCALL` en fonction du CPU.

À partir de là, nous sommes en mesure d'effectuer des hypercalls. Reste à connaître la manière dont on passe les paramètres et leur format.

On se place dans le cas où l'on réalise des hypercalls dans un environnement x64. La convention d'appel de fonction x64 veut que nous passions les arguments par les registres RCX, RDX, R8 et R9. Dans le cas où il y a plus de 4 arguments, les suivants sont mis sur la pile.

Ici, l'hyperviseur fonctionne quelque peu différemment. Les hypercalls ne possèdent que 3 arguments au maximum. On utilise donc les registres RCX, RDX et R8. Dans le cas d'un environnement x86, on a par convention : RCX=EDX:EAX, RDX=EBX:ECX et R8=EDI:ESI.

Le premier argument d'un hypercall est une structure `HV_X64_HYPERCALL_INPUT`. Elle provient du fichier d'en-tête `hvgdk.h` fourni par le WDK et défini comme :

```
typedef union _HV_X64_HYPERCALL_INPUT
{
  struct
  {
    UINT32 CallCode        : 16; // Least significant bits
    UINT32 IsFast          : 1; // Uses the register based form
    UINT32 Reserved1       : 15;
    UINT32 CountOfElements : 12;
    UINT32 Reserved2       : 4;
    UINT32 RepStartIndex   : 12;
    UINT32 Reserved3       : 4; // Most significant bits
  };
  UINT64 AsUINT64;
} HV_X64_HYPERCALL_INPUT, *PHV_X64_HYPERCALL_INPUT;
```

On retrouve, dans cette structure, l'index de notre hypercall (`Callcode`). Les membres `CountOfElements` et `RepStartIndex` sont là pour les hypercalls répétitifs. Le membre `IsFast` désigne la manière dont on passe les arguments :

Si `IsFast` est à 0, alors l'hyperviseur considère que les 2 arguments suivants qui sont dans RDX et R8 contiennent des GPA. RDX pointe sur une page contenant les arguments d'entrée alignés sur 8 octets. R8 pointe sur une page qui va recevoir les valeurs de sorties, alignée aussi sur 8 octets. Évidemment, c'est à l'utilisateur de fournir ces pages.

Si IsFast est à 1, alors on n'a que des valeurs d'entrée pour notre hypercall. RDX contient la première valeur et R8 la seconde.

Dans les 2 cas, la valeur de retour de l'appel est mise dans RAX (ou EDX:EAX). Il s'agit d'une structure de la forme :

```
typedef union _HV_X64_HYPERCALL_OUTPUT
{
  //
  // Output: The result and returned data size
  //
  struct
  {
    UINT16 CallStatus;       // Least significant bits
    UINT16 Reserved1;
    UINT32 ElementsProcessed : 12;
    UINT32 Reserved2         : 20; // Most significant bits
  };
  UINT64 AsUINT64;
} HV_X64_HYPERCALL_OUTPUT, *PHV_X64_HYPERCALL_OUTPUT;
```

Le CallStatus est de type HV_STATUS. Il permet de connaître le code de retour de l'appel. Ces statuts sont définis dans hvgdk.h.

Maintenant, nous sommes en mesure de coder notre propre interface d'hypercall. Voici un exemple qui utilise l'API HvInitializePartition :

```
HV_STATUS WrapHvInitializePartition(HV_PARTITION_ID PartitionId)
{
      ULONG i;
      PVOID HyperCallPage;
      HV_X64_HYPERCALL_INPUT HyperCallIn;
      HV_X64_HYPERCALL_OUTPUT HyperCallOut;
      PHYSICAL_ADDRESS paHyperCallPage;
      HV_STATUS (*HyperCall)(ULONG64 HyperCallInValue, ULONG64 Arg1,
      ULONG64 Arg2);

      RtlZeroMemory(&HyperCallIn, sizeof(HyperCallIn));
      RtlZeroMemory(&HyperCallOut, sizeof(HyperCallOut));

      HyperCallIn.CallCode=HvCallInitializePartition;
      HyperCallIn.IsFast=0;

      paHyperCallPage=SetupHvInterface();
      if(paHyperCallPage.QuadPart==0)
          return HV_STATUS_OPERATION_DENIED ;
       // Mappe la page d'hypercall dans l'espace mémoire noyau de notre
       guest
      HyperCallPage=MmMapIoSpace(paHyperCallPage, PAGE_SIZE, MmCached);
      if(HyperCallPage==NULL)
      {
          DbgPrint("Error while mapping HyperCallPage with MmMapIoSpace\n");
          return HV_STATUS_OPERATION_DENIED;
      }

      HyperCall=HyperCallPage;

      HyperCallOut.AsUINT64=HyperCall(HyperCallIn.AsUINT64,
      MmGetPhysicalAddress(&PartitionId).QuadPart, 0);

      MmUnmapIoSpace(HyperCallPage, PAGE_SIZE);
      return HyperCallOut.CallStatus;
}
```

Remarquez que, dans cet exemple, nous sommes en IsFast=0. Nous n'avons donc pas besoin d'allouer de pages pour l'appel.

Nous sommes donc maintenant en mesure de nous interfacer avec l'hyperviseur. Il nous est donc possible de recoder nos propres fonctionnalités pour travailler avec Hyper-V.

Conclusion

Dans cet article, nous avons vu l'architecture de Hyper-V. Même si celle-ci n'innove pas dans la matière, elle propose quelques améliorations intéressantes qui augmentent les performances des hyperviseurs monolithiques.

Nous nous sommes ensuite concentrés sur l'interface avec le cœur de Hyper-V, son hyperviseur. Cette interface est quelque peu difficile à manipuler, mais elle permet d'accéder à l'ensemble des fonctionnalités de l'hyperviseur.

Un mot sur la version R2 de Hyper-V **[23]** qui intégrera la technologie SLAT (*Second Level Address Translation*) – fondée sur EPT **[24]** pour Intel et NPT **[25]** pour AMD – et qui ajoute un niveau de virtualisation sur la pagination. On trouvera aussi la gestion des IOMMU **[26]** afin d'augmenter la sécurité des accès DMA. Enfin, Hyper-V R2 implémentera les techniques SRTM (*Static Root of Trust Measurement*) et DRTM (*Dynamic Root of Trust Measurement*) qui mettent en place des chaînes de confiance sur le code qui est exécuté depuis le démarrage jusqu'au fonctionnement. Ces méthodes utilisent les technologies Intel TXT (*Trusted eXecution Technology*), AMD-SVM ainsi que le composant TPM **[27]**.

Avec Hyper-V, Microsoft espère bien rattraper son retard sur le marché de la virtualisation. Il tire partie des dernières technologies en matière de virtualisation et de sécurité matérielle et restera à la pointe dans ce domaine. Hyper-V a donc toutes les chances d'être, dans le futur, un produit incontournable de virtualisation. ■

Remerciements

Je tiens à remercier toute l'équipe d'*EADS Innovation Works*, plus particulièrement mon maître, Nicolas Ruff, qui m'a donné l'occasion d'étudier un sujet aussi intéressant. Merci aussi à toutes les personnes du net avec qui j'échange et découvre de nouvelles choses tous les jours.

Les références de cet article sont disponibles sur miscmag.com/ref42.

Mickaël Salaün <mic@digikod.net>

VOYAGE DANS L'ANTRE DE XEN

mots-clés : *virtualisation / hyperviseur / Xen / rootkit / canal caché*

Xen est aujourd'hui l'un des systèmes de virtualisation parmi les plus utilisés en matière d'hébergements mutualisés. Bien que ce système soit fréquemment mis en œuvre, ses rouages internes sont encore méconnus.

Ce logiciel libre est né à l'université de Cambridge. Il est depuis passé entre les mains de XenSource qui est maintenant la propriété de Citrix. Le code de Xen est donc disponible, ce qui représente une source de confiance non négligeable et une relative garantie de pérennité. Pour autant, il n'est pas exempt de défauts, notamment en matière de sécurité, comme nous le verrons par la suite.

Un moniteur de machines virtuelles comprend en effet différentes parties qui doivent communiquer entre elles. Les communications doivent pourtant être contrôlées si l'on souhaite mettre en place une politique de cloisonnement. Il existe pour cela des mécanismes pouvant être utilisés pour appliquer une politique de sécurité censée fixer des règles infranchissables. D'éventuelles erreurs peuvent rendre ces mécanismes inefficaces.

Il est toujours intéressant de comprendre le fonctionnement des technologies que l'on souhaite exploiter. Cet article est une introduction à Xen et au monde qui l'entoure. Pour commencer cette exploration, je vais essayer de résumer l'architecture de ce système. Quelques mécanismes essentiels de la gestion des machines virtuelles par le virtualiseur seront expliqués pour introduire les concepts fondamentaux. Ceci permettra ensuite un rapide survol des fonctionnalités de sécurité que Xen permet de mettre en œuvre. Enfin, la dernière partie fera le tour des failles découvertes et des points faibles de cette architecture, notamment en ce qui concerne le cloisonnement.

1. Mécanismes internes de Xen

1.1 L'hyperviseur

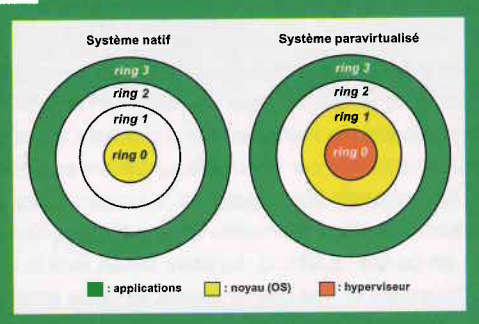
Il existe deux types de solutions de virtualisation appelés « type 1 » et « type 2 ». Le type 1 est basé sur un hyperviseur, également appelé *Virtual Machine Monitor* (VMM), directement en contact avec le matériel ; il est dit « natif » ou *bare metal*[1]. L'architecture se divise en trois parties : le matériel, l'hyperviseur et ensuite les différents systèmes invités (virtualisés). La solution de type 2 implique l'utilisation d'un virtualiseur dit « hébergé » par un système d'exploitation standard[2]. On obtient alors un empilement plus complexe comprenant le matériel, l'OS hôte, le virtualiseur et les instances invitées.

Xen est un système de paravirtualisation de type 1. C'est-à-dire qu'il comprend un hyperviseur implémentant des machines virtuelles (VM). Les OS ont conscience d'être virtualisés et sont conçus pour communiquer avec l'hyperviseur pour certaines actions nécessitant des droits qu'ils n'ont pas. Ces machines invitées sont appelées « PV » (paravirtualisé). Il est également

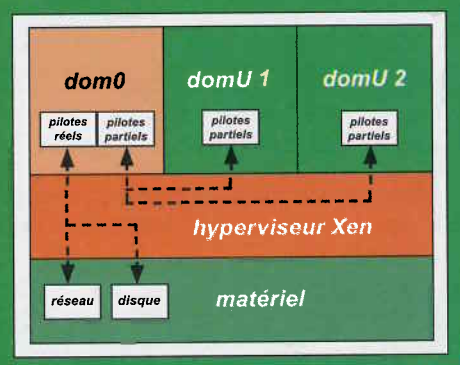
possible de faire fonctionner des systèmes d'exploitation qui ne sont pas spécialement adaptés, grâce au contrôleur de périphérique emprunté à l'émulateur Qemu (qemu-dm). Ce sont alors des *Hardware Virtual Machines* (HVM).

Un processeur de type x86 32 bits comprend quatre niveaux d'exécution. Ils sont nommés *ring* et représentent une décroissance de droits (voir Figure 1). Le code s'exécutant en *ring 0* a tous les droits sur le matériel et est habituellement le noyau du système d'exploitation (*kerneland*). Les autres rings ont des privilèges restreints. Le ring 3 contient le code le moins privilégié au niveau du bon fonctionnement du système : les applications utilisateur (*userland*). Les ring 1 et 2 ne sont pas utilisés (sauf par quelques systèmes d'exploitation exotiques). Dans un système paravirtualisé tel que Xen, c'est l'hyperviseur qui se place en ring 0. Les systèmes d'exploitation s'exécutent dans le ring 1 et les applications en ring 3. Le rôle de l'hyperviseur est d'effectuer des actions privilégiées pour du code de ring 1 tout en veillant à bien respecter la configuration choisie.

Un hyperviseur doit présenter une plate-forme virtuelle à ces OS invités, se protéger vis à vis des instances invitées et assurer une isolation correcte entre les instances invitées.



1 *Utilisation des niveaux d'exécution dans une architecture x86 32 bits hors virtualisation matérielle*



2 *Communications entre les domaines passant par l'hyperviseur*

Cette isolation est importante en cas de problème accidentel (bogue) ou intentionnel (attaque) au sein d'un système invité, celui-ci ne devant avoir aucun impact sur un autre système invité ou sur le moniteur de machines virtuelles lui-même. La conséquence au niveau du logiciel de virtualisation est qu'il doit assurer le contrôle de tous les moyens d'échanges d'informations (processeurs, mémoires, périphériques...) utilisés par les OS invités. Il doit également prendre en compte le partage des ressources matérielles garantissant le bon fonctionnement de l'ensemble et si possible offrir un mécanisme de qualité de service afin de gérer au mieux les ressources et éviter les attaques en saturation.

1.2 Les domaines

Les domaines correspondent à chaque machine virtuelle (voir Figure 2). Le « domaine 0 », abrégé en dom0, est chargé d'assurer la gestion de Xen et ses relations avec les périphériques. En pratique, il est censé être utilisé uniquement par l'administrateur du système entier afin de gérer les autres domaines virtualisés.

Il contient les pilotes qui permettent de communiquer avec le matériel. En effet, ceux-ci n'ont pas à être spécialement conçus pour Xen, mais simplement pour le système d'exploitation du dom0. Celui-ci peut être un système Linux, FreeBSD, NetBSD ou encore OpenSolaris. Il est donc possible d'utiliser un grand nombre de matériels.

L'autre utilité du dom0 est de fournir les outils permettant de contrôler les domaines. Les différents utilitaires fournis sont accessibles à partir de la commande xm (voir Figure 3). Elle permet de lister, créer, accéder, modifier et supprimer les différents domaines.

Les autres domaines sont appelés « domaines utilisateur » et abrégés en domU. Ce sont les machines virtuelles qui sont utilisables par les utilisateurs. Elles communiquent avec le dom0 par l'intermédiaire de l'hyperviseur et peuvent être contraintes par la politique de sécurité de Xen suivant le besoin.

```
# xm list
Name                                ID   Mem VCPUs      State   Time(s)
Domain-0                             0   747     2     r-----   1236.1
dom1                                 3   128     1     -b----      8.2
dom2                                 4   128     1     -b----      7.6
```

3 *Liste des domaines existants*

1.3 Les hypercalls

Toute communication entre les OS invités et l'hyperviseur Xen s'effectue via des *hypercalls*. Ils offrent la possibilité de communiquer avec l'hyperviseur pour effectuer des actions et recevoir des informations. Chacun d'eux est identifié par un numéro unique et associé à une définition.

L'appel à un hypercall se fait de la même manière que les mécanismes traditionnels d'appel système (syscall). On doit invoquer une interruption après avoir initialisé le registre EAX avec le numéro de l'hypercall correspondant. Les paramètres nécessaires sont transmis à l'aide d'autres registres.

À la troisième version de Xen, les appels aux hypercalls ont étés remaniés pour une utilisation plus propre et surtout une plus grande souplesse. L'hyperviseur copie une table des appels dans l'espace d'adressage de l'OS invité (voir Figure 4). De cette façon, un hypercall est transformé en fonction, ce qui offre une plus grande flexibilité, permettant ainsi de modifier simplement l'appel (dans le domaine courant).

La figure 5 montre comment est invoqué, depuis l'OS, un hypercall simple permettant de récupérer la version de l'hyperviseur courant. La table des appels comprend des entrées de 32 octets pour chaque hypercall. Dans cet exemple, le décalage de 17 entrées indique le numéro de l'hypercall `xen_version`. Il faut ensuite passer à la fonction le numéro de la commande permettant de récupérer le numéro de version[3].

```
for ( i = 0; i < (PAGE_SIZE / 32); i++ )
{
    p = (char *)(hypercall_page + (i * 32));
    *(u8  *)(p+ 0) = 0xb8;    /* mov  $<i>,%eax */
    *(u32 *)(p+ 1) = i;
    *(u16 *)(p+ 5) = 0x82cd;  /* int  $0x82 */
    *(u8  *)(p+ 7) = 0xc3;    /* ret */
}
```

4 *Écriture par Xen dans l'invité des fonctions appelant les hypercalls (standard)*

```
asm volatile("call hypercall_page + (17 * 32)"
             : "=a" (result), "=b" (ignore1), "=c" (ignore2)
             : "1" ((long)(0)), "2" ((long)(NULL))
             : "memory" );
```

5 *Exemple d'hypercall récupération de la version de Xe*

1.4 Processus de boot

Lors de l'élaboration d'un système d'exploitation compatible avec Xen, la première étape est de déclarer différentes informations dans le noyau permettant de faire le lien entre celui-ci et Xen. Le noyau d'un OS invité est un fichier ELF (voir Figure 6) lié statiquement et souvent compressé (format GZIP).

L'ELF d'un OS doit avoir une structure spécifique pour être compatible. On la retrouve dans l'en-tête sous forme de section (`__xen_guest`). Ce champ contient une chaîne de caractères précisant différentes informations (voir Figure 7) dont l'offset est l'adresse de la table des hypercalls, ainsi que des champs indiquant le niveau de compatibilité avec l'hyperviseur.

Plus loin dans le code de l'invité, on réserve de la place où Xen pourra *mapper*[4] les fonctions d'appel aux hypercalls.

1.5 Allocations mémoire

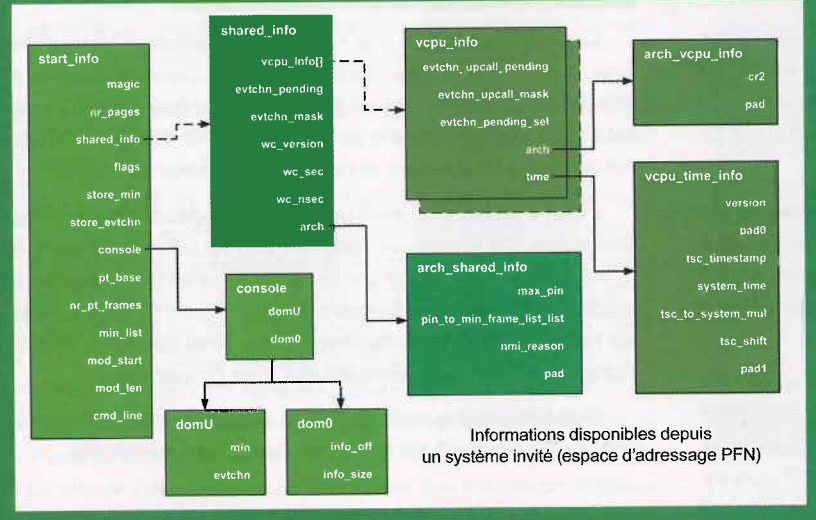
Lors de son initialisation, Xen alloue de la mémoire pour son propre fonctionnement, mais également pour les espaces d'adressage virtuels. Ces allocations se situent à des endroits

```
$ readelf -l mini-os

Elf file type is EXEC (Executable file)
Entry point 0x0
There are 2 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000080 0x00000000 0x00000000 0x10b0c 0x1dd24 RWE 0x20
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x4

 Section to Segment mapping:
  Segment Sections...
   00     .text .rodata .data .bss
   01
```

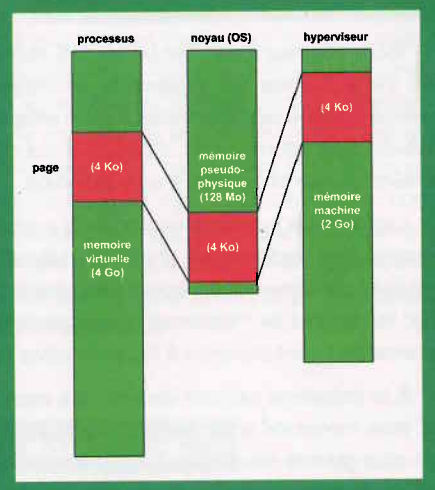
6 *En-tête partiel d'un noyau minimal*

```
.section __xen_guest
    .ascii"GUEST_OS=Mini-OS"
    .ascii",XEN_VER=xen-3.0"
    .ascii",VIRT_BASE=0x0"
    .ascii",ELF_PADDR_OFFSET=0x0"
    .ascii",HYPERCALL_PAGE=0x2"
    .ascii",PAE=no"
    .ascii",LOADER=generic"
    .byte  0
```

7 *Section d'informations de l'OS invité*



8 *Hiérarchie de la structure start_info*



9 *Exemple d'organisation de la mémoire (architecture 32 bits paravirtualisée)*

fixes de la mémoire, ce qui permet, par la suite, d'étendre l'espace d'adressage des invités. Xen doit également connaître le propriétaire de chaque zone mémoire afin de pouvoir assurer les propriétés d'isolation. Chaque domaine a donc sa zone initiale prédéfinie, mais il a également la possibilité de l'agrandir dynamiquement jusqu'à une certaine limite grâce au Balloon Driver. Ce nouveau procédé n'est toutefois pas nécessaire pour un fonctionnement standard et il n'est pas forcément activé.

Lors de l'initialisation de l'invité, une zone d'information (*Shared Info*) est copiée à un emplacement prédéterminé de la mémoire. Ces données permettent de récupérer toutes les informations nécessaires à une communication avec Xen. On y retrouve notamment le détail des différentes pages mémoire du domaine courant (longueur, adresse...)[5].

1.6 Mémoire pseudo-physique

Contrairement au fonctionnement traditionnel d'un système d'exploitation qui gère la mémoire en deux niveaux (adresse virtuelle et adresse physique), un système paravirtualisé peut se servir d'un niveau supplémentaire (adresse machine) correspondant à la mémoire de la machine réelle (voir Figure 9).

Cette nouvelle étape d'abstraction agit sur la fragmentation de la mémoire. Le système invité a cependant la possibilité de gérer lui-même cette transition. On peut ainsi avoir une allocation mémoire de l'OS invité optimisée. En effet, le système virtualisé peut allouer l'espace nécessaire de manière contiguë au niveau de la mémoire physique, pour éviter la fragmentation qui serait un frein à l'exécution de sa machine virtuelle.

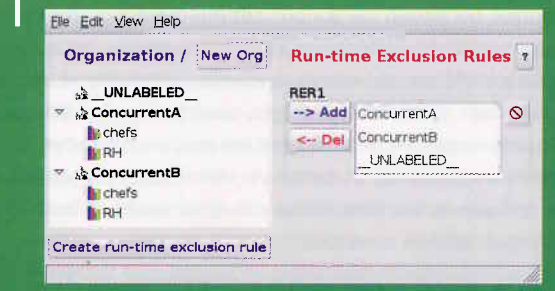
2. Fonctions de sécurité

2.1 Politique de sécurité

L'architecture de Xen en matière de sécurité est (en partie) basée sur l'architecture de sécurité de l'hyperviseur sHype[6] [1, 2]. À ce titre, elle repose sur un système de labelisation des ressources de la machine. C'est un système de contrôle d'accès obligatoire, *Mandatory Access Control* (MAC), qui permet de mettre progressivement une politique de sécurité en place.

Un des systèmes les plus connus reposant sur un MAC est SELinux. Celui-ci est spécifique à Linux et réputé pour être particulièrement complexe à mettre en œuvre. Au contraire, le système de Xen se veut plus simple en fournissant un contrôle moins fin, mais très robuste, et qui peut être considéré comme multiplateforme étant donné qu'il intervient sur les machines virtuelles.

Xen peut fixer des règles entre les domaines avec le contrôle de la mémoire partagée et des évènements interdomaines, mais également avec les ressources (stockage, réseau). Il est ainsi possible d'avoir différentes machines virtuelles, effectuant des tâches complémentaires, capable de communiquer entre elles, mais isolées des autres domaines.



10 *Outil basique de création de règles (EZPolicy d'ACM)*

Le contrôle d'accès de Xen se découpe en deux parties : la politique de contrôle d'accès, *Access Control Policy* (ACP), et le module de contrôle d'accès, *Access Control Module* (ACM). La politique de sécurité est mise en place par un mécanisme de labelisation qui définit des règles d'accès pour les fichiers de ressource (périphérique, disque, interface réseau). Le module d'accès est alors chargé d'interpréter ces règles quand les domaines ont besoin d'accéder à des ressources[7]. L'ACP donne deux façons d'utiliser les labels :

- *The simple type enforcement* : Les labels sont interprétés pour la décision de l'accès aux communications interdomaines et pour l'accès aux ressources virtuelles ou physiques. Par défaut, ces échanges sont interdits et peuvent seulement être autorisés lorsque la politique de sécurité le spécifie explicitement. L'affectation d'un label à un domaine contrôle le partage d'information entre domaines (directement à travers le canal de communication ou de façon indirecte par la mémoire partagée). C'est une politique de liste blanche.
- La muraille de Chine : Les labels sont interprétés pour décider quel domaine peut interagir avec quel autre sur un même système. On peut ainsi définir des règles d'exclusion qui permettent de prévenir les canaux cachés et minimisent les risques causés par une imperfection dans l'isolation des domaines. Il peut cependant être impensable de ne pas faire s'exécuter des machines virtuelles en même temps.

Les ACM permettent de compléter l'hyperviseur en y apportant des fonctionnalités de sécurité. Xen Security Modules (XSM), équivalent à *Linux Security Modules* (LSM), est un *framework* de sécurité qui fournit une interface permettant de personnaliser des fonctionnalités de sécurité et d'alléger l'hyperviseur quant à l'implémentation de modèles spécifiques. Il y a actuellement deux architectures de sécurité qui peuvent être utilisées : sHype et Flask. sHype est originellement développé par IBM et implémenté dans le module XSM-ACM. Flask est, quant à lui, développé par la NSA (laboratoire NIARL) et son implémentation se trouve dans le module XSM-FLASK. On peut noter que l'architecture Flask est également celle utilisée par SELinux.

Les buts prônés par ces modules sont :

- une isolation forte, intermédiaire de partage et de communication entre les instances de machines virtuelles (différentes politiques de sécurité) ;
- une attestation de l'intégrité de l'hyperviseur et de ses invités (technologie TPM [3]) ;
- un contrôle des ressources et une comptabilisation de leur utilisation ;
- des services sécurisés.

Par défaut, Xen n'intègre pas d'ACM. Il est donc nécessaire de le compiler avec le module d'accès choisi : `FLASK_ENABLE` ou `ACM_SECURITY` (sHype). Ces fonctionnalités de sécurité sont toujours en cours de perfectionnement en particulier sur l'ajout de fonctionnalités (contrôle du trafic réseau, contrôle d'usage...) et sur les outils de configuration (voir Figure 10).

2.2 Découpage du virtualiseur

Une des fonctionnalités en cours de développement, visant à découper le système de virtualisation, est l'apparition des *stub domains* [4]. Le principe consiste à contenir une partie du système dans un domaine PV dédié.

Actuellement, le dom0 contient un grand nombre de composants (de Xen), tels que le constructeur de domaine [5], le chargeur de démarrage (de domaine), l'émulation de périphérique (*ioemu device model*) ou encore les pilotes réels de périphérique. Ces composants sont souvent exécutés avec les droits *root*, ce qui peut être gênant d'un point de vue sécurité. Un deuxième point contraignant est la répartition de charge qui n'est pas gérée par l'hyperviseur lui-même, mais par le dom0 qui est autonome. Le but des stub domains est donc de répartir ces fonctions dans des domaines séparés, gérés par l'hyperviseur.

Pour arriver à cela, un stub domain est composé d'un mini-système d'exploitation (MiniOS) reposant sur des normes standards facilitant le portage des composants tels que les applications ou les pilotes. Ceci permet de prendre en compte le fait que certains pilotes de périphérique peuvent être bogués ou provenir de source non sûre.

Les domaines de périphérique (device domains) doivent permettre d'isoler un tel code dans une machine virtuelle. On aurait alors la possibilité de contenir la stabilité et l'intégrité des autres domaines. Si un pilote se termine de manière inattendue, son domaine peut être redémarré par l'hyperviseur. De cette façon, même si un pilote contient des erreurs de conception et

qu'une exploitation de faille est possible, seul le domaine chargé de ce pilote sera compromis. Les périphériques sont alors dédiés à une seule instance, peu importe leurs particularités (carte réseau, carte graphique...).

Ce confinement n'est pas encore appliqué et malheureusement pas forcément efficace étant donné l'évolution lente des périphériques et de leurs pilotes. En effet, sans une technologie de type IOMMU (*Input/Output Memory Management Unit* d'AMD ou VT-d d'Intel), un périphérique peut avoir accès à des zones mémoire en dehors de son domaine. Il serait également possible à un périphérique partageant des ressources (bus, interruptions ou encore espaces d'adressage I/O) avec un autre de lire des données qui ne lui sont pas destinées et également d'y écrire à sa place.

Dans la dernière version de Xen (v3.3), le principe des stub domains a été appliqué à l'émulation de périphérique (ioemu via qemu-dm) et au chargeur de démarrage (PyGrub).

Il est maintenant possible d'avoir des domaines d'émulation HVM (qemu-on-minios) découpant ainsi l'émulation de périphérique du dom0, ce qui permet également d'améliorer les performances des domaines HVM. En pratique, on a donc un domaine HVM qui communique avec le périphérique émulé du stub domain. Ce dernier dialogue avec le dom0 grâce aux processus de communication disponibles pour les domaines PV : hypercalls, évènements et zones de mémoire partagées. La section critique qu'est l'émulation est donc isolée et le dom0 ne reçoit que des flux contrôlés.

La partie critique qu'est PyGrub est remplacée par PVGrub. Cette évolution est très appréciable, car elle consiste à copier le système de démarrage dans le nouveau domaine qui peut par la suite s'auto-charger. Le cloisonnement est alors respecté du début à la fin de la vie d'un domaine.

3. Imperméabilité de la virtualisation

3.1 Rootkit inside

Il est intéressant de constater que plusieurs vulnérabilités ont déjà étés mises en évidence dans ce système de virtualisation. En octobre 2007, une vérification manquante permettait à un domU de lire la mémoire d'un autre domaine sur une architecture IA64[8]. Deux semaines plus tard, une autre vulnérabilité découverte permettait d'interrompre l'hyperviseur à cause d'un oubli de gestion d'un registre de débogage[9]. D'autres problèmes sont également apparus suite à des erreurs dans Qemu, qui peut être (partiellement) utilisé par Xen pour l'émulation de périphériques.

La première faille majeure de Xen a été celle de PyGrub[10]. Ce *bootloader* écrit en Python est exécuté par le dom0 pour charger une machine virtuelle. Le problème résidait dans l'interprétation du fichier de configuration du domU qui était mal *parsé* lors de sa lecture. De ce fait, il était possible de prendre le contrôle du dom0 et donc de la machine physique par l'intermédiaire d'une machine virtuelle non privilégiée.

...prendre le contrôle [...] de la machine physique par l'intermédiaire d'une machine virtuelle non privilégiée...

Plus récemment, lors des conférences Black Hat de l'été dernier, l'équipe d'*Invisible Things Lab* a rendu public des preuves de concept s'appuyant sur l'architecture de Xen pour s'infiltrer dans un système [6]. Parmi les trois conférences qui ont été présentées, la première a montré comment modifier Xen à la volée à partir d'un domaine privilégié, ce qui revient à patcher ce qui est déjà accessible. Dans un deuxième temps, ils ont présenté une « faille » qui peut être utilisée pour effectuer ces actions à partir d'un domaine non privilégié. Enfin, la dernière partie consistait à expliquer la virtualisation de virtualiseurs (via XenBluePill), afin d'insérer un *rootkit* avant la machine virtuelle.

Si on a la main sur le dom0, ce qui n'est pas forcément une mince affaire, on a alors un contrôle total des autres domaines et éventuellement de l'hyperviseur. Cependant, si une politique de sécurité est mise en place, elle peut être contraignante pour un attaquant[11].

Il existe deux manières pour prendre le contrôle de l'hyperviseur. La première méthode consiste à changer des parties de l'hyperviseur courant en réécrivant son code à partir de la mémoire physique. Ceci peut être possible grâce à un périphérique qui utiliserait du *Direct Memory Access* (DMA) comme peut le faire un disque dur ou une carte réseau[12]. Il est alors possible à partir du dom0 d'accéder à l'endroit souhaité de la mémoire de la machine physique. La deuxième méthode pour prendre le contrôle de l'hyperviseur est, comme pour tout logiciel, d'en exploiter une faille...

Une fois qu'on a la possibilité de réécrire l'hyperviseur à sa guise, une technique simple consiste à réécrire un hypercall. Par exemple, un argument ajouté à un hypercall standard pourrait permettre d'exécuter une action définie par un domaine non privilégié. Il est également possible de modifier Xen pour le contrôler par le réseau. Il s'agit alors de capturer certains paquets réseau de façon à y lire les commandes et à le cacher aux autres domaines[13].

Une démonstration concernant directement Xen peut quant à elle être considérée comme très critique. En effet, comme expliqué précédemment, il existe actuellement deux modules de sécurité pour Xen : Flask et ACM. Le premier est développé par la NSA, ce qui peut conforter sa position en matière de sécurité. Pourtant, c'est dans ce module qu'une erreur a été découverte ouvrant alors la porte à une utilisation très gênante[14]... Flask n'est pas activé par défaut, mais, dans un contexte ou la sécurité est importante, il y a des chances qu'il y soit. L'erreur trouvée est un *buffer overflow* : un manque de vérification de la taille d'une donnée et l'utilisation d'un sscanf[15] ! Cette vulnérabilité était exploitable à partir d'un domU, ce qui représentait un risque majeur, car cela touchait le processus de l'hyperviseur.

Dans un autre contexte, suite à un buffer overflow découvert dans *Xen Para Virtualized Frame Buffer* (PVFB) servant à l'affichage dans les domaines, une exécution de code par le dom0 était rendue possible pour un simple utilisateur. Ce pilote d'affichage échange ses informations à l'aide du système de communication interdomaines XenStore pour transférer le rendu graphique sur le poste physique. Il se trouve que le framebuffer communique, en plus des images, différentes valeurs comme la résolution de l'affichage. C'est sur celle-ci qu'il y avait un manque de vérification, ce qui permettait de déborder dans le code du pilote, autrement dit, d'écrire dans le dom0[16].

...l'aspect critique du code de l'hyperviseur, mais également de celui du dom0...

Ces différentes failles montrent l'aspect critique du code de l'hyperviseur, mais également de celui du dom0. Dans un premier temps, il est donc important d'auditer le code de Xen et d'éviter d'augmenter sa taille. Dans un second temps, c'est au dom0 qu'il faut prêter attention ; c'est ce couple qui représente le virtualiseur. Plus un logiciel dispose de fonctionnalités, plus il grossit, ce qui induit une plus grande complexité de son audit. Bien que le projet Xen vise à créer un hyperviseur minimal (afin d'avoir la possibilité d'auditer le code), il comprend quand même près de 312 000 lignes de code source[17], ce qui est conséquent, mais nécessaire, étant donné les fonctionnalités offertes. Le compromis entre l'utilisabilité et la sécurisation est, comme souvent, à prendre en considération.

3.2 Virtualisation et cloisonnement

Les offres de mutualisation sont par nature des offres de virtualisation. Les différentes solutions vont de l'isolation de contexte (OpenVZ, VServer...) jusqu'à de l'émulation (Qemu, Bochs...) en passant par de la paravirtualisation (Xen, Hyper-V...) ou un mixte de ces méthodes. Dans le cas des solutions de virtualisation, il est primordial pour le fournisseur de pouvoir cloisonner les différentes machines qu'il loue à ses clients. Sauf besoin spécial, personne ne souhaite exposer ses données à des tiers sous prétexte qu'ils sont hébergés au même endroit.

De la même manière, un échange de données entre deux machines virtuelles peut être intéressant s'il est contrôlé, mais, dans le cas contraire, cela devient problématique.

Dans l'hypothèse d'une compromission de la machine d'un concurrent, comme dans toute attaque ayant pour but l'espionnage, l'objectif prioritaire de l'attaquant est de rester le plus longtemps possible en place. Autrement dit, la *backdoor* installée (par un procédé sur lequel je ne m'étendrai pas) doit être la plus discrète possible. Diverses méthodes permettent de rester discret, avec notamment les procédés consistant à exécuter entièrement en mémoire un programme tout au long de sa vie[18].

Pour être utile, une backdoor doit être en mesure de communiquer avec l'extérieur. Dans certains cas, le seul moyen est de passer par le réseau. Dans le cas de machines virtuelles, une nouvelle méthode est disponible à l'attaquant : la communication entre instances virtuelles hébergées sur une même machine physique. Dans les solutions d'hébergement, il n'est pas courant que l'hébergeur mette de telles fonctionnalités à disposition de ses clients sachant qu'ils n'ont pas donné leur accord.

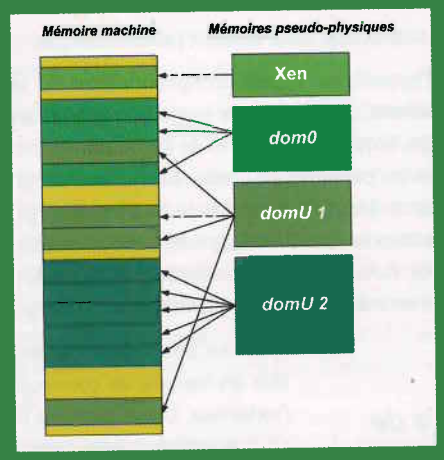
Une backdoor ayant une furtivité au niveau du système l'exécutant peut tout de même être détectée lors d'échanges réseau si ceux-ci sont correctement analysés[19]. On se retrouve alors dans le cas où une communication discrète entre deux machines virtuelles devient critique vis-à-vis de la détection d'intrusion et donc de la sécurité des systèmes exécutés. C'est ce cas que je vais aborder maintenant.

3.3 Exemple pratique de communications furtives : XenCC

Il existe un mécanisme de communication entre invités appelé « XenStore » qui permet de partager des plages mémoire en lecture/écriture. On peut alors échanger diverses informations entre invités suivant les droits qui leurs sont accordés. Cependant, il existe un autre moyen, dépendant de l'architecture de Xen, d'échanger des données entre invités.

L'analyse du fonctionnement de la mémoire m'a permis de réaliser un outil reposant sur le même principe que XenStore, mais sans « contrainte » de la politique de sécurité. XenCC est une preuve de concept permettant d'établir un canal caché pour communiquer entre invités indépendamment du domaine (dom0 ou domU). Pour cela, on doit avoir les droits qui permettent d'insérer un module noyau, autrement dit de modifier le noyau de notre domaine invité, ce qui permet d'utiliser des hypercalls.

Le principe repose sur l'utilisation de la table de *mapping* entre les adresses machines et les adresses pseudo-physiques de l'hyperviseur. Afin de pouvoir l'utiliser, on doit passer par un hypercall (`mmu_update`) pour établir des relations (voir Figure 11).



11 *Exemple d'organisation de la mémoire*

Il est important de constater que cette table est lisible par tous les invités étant donné qu'il n'y en a qu'une seule. Évidemment, seul le détenteur d'une plage d'adresse peut la modifier. On parle ici seulement d'adresses, mais, en aucun cas, des données vers lesquelles elles pointent. En effet, si on veut pouvoir lire à un emplacement, il faut pouvoir le mapper en mémoire. Ici, on ne peut mapper que nos propres données (pages mémoire). Cette table de relation contient normalement des adresses, mais, étant donné qu'il n'y a aucune vérification, on peut écrire ce que l'on souhaite.

Le principe du canal caché est simple : il faut écrire un tag pour (notamment) que le domaine avec qui on souhaite communiquer puisse trouver l'emplacement correspondant au message, et ensuite inscrire les données que l'on veut passer à notre correspondant. Une fois ces informations écrites, n'importe quel invité qui connaît le tag peut retrouver et extraire les informations. On obtient ainsi un canal *half-duplex* dès que deux invités connaissent mutuellement leurs tags. Ce nouveau canal de communication fonctionne actuellement pour Linux 2.6 x86-32. Les lecteurs intéressés pourront se référer au code du PoC pour plus d'informations[20].

Cette constatation n'est pas nouvelle, puisqu'elle a été relevée en 2006 sur la *mailing-list* de xen-devel[21] ; cependant, les participants ont conclu que le passage de la théorie à la pratique était difficile et que cette menace restait donc, jusqu'à présent, théorique. Au final, comme le pensaient les développeurs, cette méthode permet en effet d'outrepasser la politique de sécurité de Xen.

Actuellement, aucune solution ne permet de détecter ce type de communication. Afin d'empêcher ces échanges, une première idée pour Xen serait de vérifier la validité des adresses de type PFN (*Pseudo-physical Frame Number*) qui sont inscrites dans la page. De cette façon, on limite les possibilités d'écriture, mais avec une perte de temps à chaque appel due à la vérification. Cependant, il serait assez simple d'outrepasser cette mesure avec un algorithme adapté qui respecterait cette nouvelle règle en rendant légitimes de telles adresses.

Il existe un autre moyen de gérer la pagination avec l'utilisation des *shadow page tables*. Dans ce cas, l'invité virtualisé n'a pas à s'occuper de l'organisation du mapping de la mémoire machine. Le mapping qu'effectue le système virtualisé est intercepté et remplacé par celui de Xen. Il suffit pour cela que la table des pages soit marquée en lecture seule, ce qui va conduire à lever des exceptions que l'hyperviseur interceptera ensuite. Le principe est donc de laisser l'hyperviseur gérer la mémoire[22]. En contrepartie, il en découle une baisse de performance si la translation des adresses est logicielle.

La table faisant le lien entre les adresses pseudo-physiques et les adresses machines est une bonne idée pour augmenter les performances de mapping de mémoire, mais il serait judicieux d'avoir une table pour chaque invité. Elles auraient alors la même protection que la mémoire ordinaire et seraient inaccessibles à un autre invité.

La solution la plus appropriée est donc de donner à chaque domaine une table personnalisée. Chaque invité aurait alors seulement les adresses le concernant mises à jour et il n'aurait aucune vue des mappings des autres. La mise en œuvre de cette méthode a pour coût la commutation de table à chaque changement de contexte entre OS. L'occupation mémoire ne devrait pas poser de problèmes étant donné la faible taille de chaque table.

Si cette remarque était prise en compte par les développeurs, l'exploitation des canaux cachés comme le fait XenCC serait alors impossible. Cette méthode pour « isoler » les invités pourrait également servir à les empêcher de lire la table et d'en déduire une cartographie basique de la mémoire physiquement utilisée.

Tout système mutualisé n'ayant pas d'objectif de partage de l'information entre instances se doit d'isoler celles-ci entre elles. Tout manquement à cette règle entraînerait des effets de bord imprévus. On a, dans ce cas, une communication que l'on peut appeler « canal caché », possible entre plusieurs instances virtualisées complices. Ce « détail » peut ne pas choquer, mais si une backdoor utilise cette méthode, le problème de la communication devient crucial.

Par nature, les canaux cachés ne pourront jamais être complètement exclus de certains environnements. En effet, pour exclure cette possibilité, il faudrait un système entièrement isolé (du monde extérieur), ce qui veut dire n'ayant aucun moyen de communication, le rendant alors inutile... Il est cependant possible de réduire significativement l'utilisation des canaux cachés par une étude et une conception attentive à ce problème. Que ce soit avec les hyperviseurs ou plus globalement avec les systèmes d'exploitation, le contrôle des transferts de données est primordial si la politique de sécurité se veut cohérente.

Conclusion

Xen est un système de virtualisation parmi les plus intéressants et performants. Ses possibilités vont en grandissant et le nombre de ses utilisateurs également. Il ne faut cependant pas négliger la sécurité au profit des performances. Actuellement, l'implémentation que je viens de décrire ne semble intéresser qu'une petite minorité des développeurs de ce projet, ce qui est regrettable.

A contrario, dans la dernière version de Xen, de nouvelles fonctionnalités très prometteuses sont apparues. En ce qui concerne la sécurité, l'apparition des stub domains a permis de mettre en place les domaines d'émulation HVM permettant de déléguer à des domaines dédiés la gestion de l'émulation d'un pilote matériel. De cette façon, même si l'émulation n'est pas exempte de bogues, seul le domaine chargé de ce pilote pourrait être compromis (et réinitialisé au besoin). Un principe comparable a été appliqué pour l'initialisation des domaines PV avec PVGrub. Enfin, l'utilisation des technologies de virtualisation matérielle sont prises en charge pour les domaines HVM.

Il est évident que le point le plus critique sur une installation de Xen n'est pas l'hyperviseur en lui-même, mais bien tout ce qui gravite autour. Comme on l'a vu, les pilotes de périphérique sont des points d'entrée très exposés dans un système virtualisé (ou natif). Il est donc important de prêter une grande attention au dom0 qui est chargé de piloter tout le système par l'administration des machines virtuelles. Il existe des outils adaptés pour durcir un système d'exploitation et c'est le cas pour un système basé sur Linux. À ce titre, on peut trouver des solutions pour maîtriser et cloisonner un maximum tout ce qui s'exécute sur le domaine administrateur. Les solutions les plus connues sont SELinux, GrSecurity, et, bien sûr, une configuration soignée est recommandée[23]. Ces évolutions doivent à terme permettre la décomposition du dom0 afin de cloisonner au mieux les zones sensibles pour limiter toute tentative de corruption du système global. ■

Notes

[1] Virtualiseurs de type 1 : Xen, VMware ESX, Hyper-V...

[2] Virtualiseurs de type 2 : Qemu, VirtualBox, VMware Workstation, Parallels...

[3] Les actions effectuées à l'intérieur de la fonction appelée consistent principalement à mettre le registre EAX à 17 et le registre EBX à 0 avant d'appeler l'interruption 0x82.

[4] Voir la fonction `hypercall_page_initialise` dans le fichier `xen/arch/x86/x86_32/traps.c`

[5] Voir la structure `start_info` dans le fichier `xen/include/public/xen.h`

[6] Voir la description détaillée dans le fichier `tools/security/policy.txt`

[7] Les droits d'accès des domaines sont déterminés par leurs labels de sécurité et non par leur nom ou leur identifiant.

[8] CVE-2007-6207

[9] CVE-2007-5906

[10] CVE-2007-4993

[11] La politique de sécurité de Xen est chargée au démarrage de la machine physique comme l'est le dom0.

[12] Ceci est vrai s'il n'y a pas de protection matérielle (IOMMU/VT-d) ou si une faille matérielle est utilisée (voir la faille de l'alliance BIOS/chipset Q35).

[13] Pour plus de détails, voir l'implémentation des backdoors DR et Foreign.

[14] Les adeptes de la théorie du complot feront le lien avec le concepteur.

[15] À titre d'information, les 25 erreurs de programmation les plus dangereuses ont étés référencées : **http://www.sans.org/top25errors/**

[16] Une preuve de concept (fonctionnant aussi malgré SELinux, un espace d'adressage aléatoire et le bit NX activé) a été publiée : **http://invisiblethingslab.com/resources/misc08/xenfb-adventures-10.pdf**

[17] Fichiers source de l'hyperviseur Xen (v3.3.1) sans les outils `wc -l xen/**/*.[chsS]` → 311969

[18] *Sanson the Headman* en est un très bon exemple : **http://sanson.kernsh.org**

[19] Il existe bien sûr des mécanisme de canaux cachés pouvant rendre une communication réseau extrêmement difficile, voire impossible à détecter suivant les moyens employés et le débit utilisé.

[20] XenCC sous licence libre (GPL v3) disponible sur la liste de Xen devel ou sur **http://digikod.net/public/XenCC**

[21] **http://lists.xensource.com/archives/html/xense-devel/2006-02/msg00001.html**

[22] La prise en compte de la virtualisation par les processeurs permet d'avoir une autre solution en leur déléguant la gestion de cette table de traduction.

[23] N'installer que ce qui est strictement nécessaire, restreindre les accès physiques, réseau et utilisateur, etc.

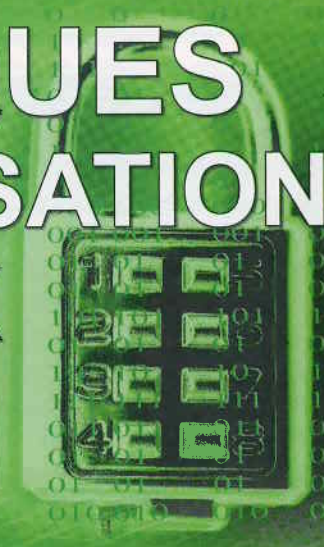
Remerciements

Je tiens à remercier le labo SSI/SSY du CELAR pour m'avoir donné l'occasion d'explorer le monde de la virtualisation, et les relecteurs de cet article pour leurs critiques.

Les références de cet article sont disponibles sur miscmag.com/ref42.

Olivier Revenu – EdelWeb – Groupe ON-X – olivier.revenu@edelweb.fr

BONNES PRATIQUES POUR LA VIRTUALISATION AVEC VMWARE ESX



mots-clés : *sécurité / virtualisation / bonnes pratiques / retour d'expérience / VMware*

La virtualisation a connu une croissance exponentielle ces dernières années. Elle apporte de nombreux avantages indéniables – consolidation, réduction des coûts, simplification de la gestion de l'architecture – mais également des inconvénients dont la sécurité, mal appréhendée, fait parfois partie. Cet article se veut être un recueil de bonnes pratiques pour intégrer au mieux la sécurité dans un projet de virtualisation en entreprise. Ce recueil n'est bien évidemment pas exhaustif et il conviendra de le compléter et de l'adapter au contexte de son projet. Enfin, ces recommandations sont tirées d'un retour d'expérience d'un projet de virtualisation de 200 serveurs auquel nous avons participé.

La virtualisation reste une solution complexe à maîtriser lorsqu'elle est déployée en entreprise. En effet, les solutions disponibles sur le marché permettent de virtualiser aussi bien le système que le réseau, ce qui peut avoir des impacts non négligeables pour l'entreprise sur ses moyens techniques et organisationnels, alors inadaptés. De plus, les problématiques sécurité sont souvent sous-estimées par les équipes en charge de ce type de projet.

La terminologie suivante sera employée dans cet article :

- **Machine hôte** : ordinateur physique hébergeant la solution de virtualisation et les machines virtuelles (VM).
- **Système hôte** : système d'exploitation de la machine hôte disposant d'un contrôle et d'un accès complet sur le matériel et sur les VM. Dans le cas de la solution VMware ESX, il n'y a pas de système hôte à proprement parler. Mais, par abus de langage, cette appellation désigne, sous ESX, la couche de virtualisation composée de l'hyperviseur et des VMM.
- **Hyperviseur** : composant en charge du partage des ressources (CPU, mémoire, disque et réseau) et des périphériques, du contrôle et de l'isolation des VM.
- **VMM** *(Virtual Machine Monitor)* : composant faisant l'interface entre l'hyperviseur et la VM en charge d'émuler le matériel (couche d'abstraction matérielle) pour le système invité.
- **Machine virtuelle** *(ou* **VM***)* : ordinateur virtualisé au sein d'une machine hôte.
- **Système invité** : système d'exploitation qui est exécuté au sein d'une VM.

Nous aborderons, en premier lieu, la problématique de l'isolation d'une VM au sein de son hôte.

Puis, la suite de l'article sera organisée selon les grandes étapes d'un projet de virtualisation :

- L'**étude de contexte** qui a pour objectif de réaliser un état des lieux du parc informatique et de définir le périmètre de la virtualisation en fonction de différents critères.
- L'**analyse de risques** qui permet d'identifier les points de faiblesse de la future architecture et de définir les contre-mesures à appliquer.
- La **définition de l'architecture** tenant compte des résultats de l'étape précédente et la rédaction des **spécifications techniques de sécurité** pour couvrir les risques inacceptables.
- La phase de **déploiement** de l'architecture de virtualisation et de **migration** des machines physiques vers des VM.
- La phase de **définition de l'organisation, des procédures et la formation des équipes** qui permet d'adapter l'entreprise aux évolutions techniques et aux nouveaux rôles et responsabilités induits par ce projet.
- La **recette finale** qui offre la garantie que la plateforme est conforme aux exigences fonctionnelles et de sécurité.
- La mise en **exploitation de la plateforme** qui s'accompagne d'un suivi de la menace et de l'application de correctifs de sécurité.
- Les **audits de sécurité** qui sont réalisés périodiquement afin de contrôler la conformité de la plateforme et la présence de vulnérabilités.

Nous étudierons dans le cadre de cet article : l'étude du contexte (2.), l'analyse de risques (3.), la définition de l'architecture et des spécifications techniques de sécurité (4.), et la définition de l'organisation, des procédures et la formation des équipes (5.).

1. Virtualisation, isolation et sécurité

Théoriquement, un processus exécuté dans une VM est cloisonné : il ne peut pas observer, affecter ou communiquer avec d'autres processus de la station hôte ou d'une autre VM.

En réalité, une VM communique en permanence avec son hôte via un certain nombre de canaux pour des besoins conceptuels (par exemple : les accès DMA aux périphériques virtuels par le système invité sont interceptés par l'hyperviseur) et fonctionnels, par l'intermédiaire d'une *backdoor* non documentée (terminologie VMware). Celle-ci est « invoquée » depuis une VM en adressant un port I/O spécifique et est utilisée pour remplacer certaines fonctions du BIOS virtuel, pour le fonctionnement de certains pilotes de périphériques, et pour les *VMware Tools* qui offrent une interaction entre le système invité et l'hôte ou le client VMware. Par exemple, cette backdoor permet d'utiliser des messages *setinfo* pour changer la résolution d'écran de la VM, obtenir des informations comme la version de VMware, la fréquence du processeur de l'hôte, la position du curseur [1].

Le cloisonnement offert par une solution de virtualisation repose encore aujourd'hui en grande en partie sur des mécanismes logiciels et donc faillibles[1].

note

[1] Nous évoquerons en conclusion les nouveaux jeux d'instructions des processeurs, Intel TXT et AMD SVM, améliorant l'isolation dans la virtualisation et introduisant un contrôle dans le code virtualisé.

Ainsi, on dénombre, depuis 2006, pas moins de 25 avis de sécurité et 152 vulnérabilités pour la version 3 de VMware ESX [2]. Ce chiffre reste à rationaliser, car la plupart des vulnérabilités concernent des composants tiers non maintenus par VMware et présents dans le Service Console, un système d'exploitation virtualisé, dérivé d'une Red Hat Linux (Enterprise Linux 3 pour ESX 3.x), et servant à administrer et surveiller le serveur ESX.

Les vulnérabilités les plus critiques ont pour impact :

- de compromettre le système hôte depuis une VM (*attaque baptisée « VM escape » : un programme s'exécutant dans une VM parvient à s'extraire de la couche « virtualisation » pour compromettre son hôte*) ;
- d'obtenir une élévation de privilèges sur le système invité d'une VM ;
- de réaliser un déni de service sur le système hôte ou sur une VM.

Parmi ces vulnérabilités rendues publiques depuis 2006, au moins 3 d'entre elles pourraient permettre de réaliser une attaque de type « VM Escape » et au moins 3 autres de réaliser un déni de service sur le système hôte.

C'est le cas, par exemple, des vulnérabilités suivantes :

- CVE-2007-4496 : une faille non spécifiée peut permettre à un compte privilégié dans un système invité de provoquer une corruption de la mémoire dans un processus VMware de l'hôte, pour ainsi exécuter du code arbitraire sur ce dernier.
- CVE-2007-4497 : une erreur non spécifiée peut permettre depuis un système virtualisé de provoquer un dysfonctionnement dans un processus VMware de l'hôte.

À ce jour, il n'existe pas de programme de démonstration publique pour l'exploitation de ces vulnérabilités sous VMware ESX. En revanche, c'est le cas pour des failles similaires sur les produits de la gamme Workstation.

Une mauvaise implémentation du produit peut permettre d'échanger des informations entre deux VM. Par exemple, VMware propose des fonctionnalités permettant de « rompre » l'isolation d'une VM depuis un client VMware :

- *drag and drop* pour échanger des fichiers entre un client et une VM ;