Les systèmes de détection d'intrusions

David Burgermeister, Jonathan Krier

Date de publication : 22/07/2006
Date de mise à jour : 22/07/2006

Sites : http://dbprog.developpez.com
http://krierjon.developpez.com

Sommaire

I. Introduction

Les systèmes d'information sont aujourd'hui de plus en plus ouverts sur Internet. Cette ouverture, a priori bénéfique, pose néanmoins un problème majeur : il en découle un nombre croissant d'attaques. La mise en place d'une politique de sécurité autour de ces systèmes est donc primordiale.

Outre la mise en place de pare-feux et de systèmes d'authentification de plus en plus sécurisés, il est nécessaire, pour compléter cette politique de sécurité, d'avoir des outils de surveillance pour auditer le système d'information et détecter d'éventuelles intrusions.

Ce que nous appelons intrusion signifie pénétration des systèmes d'information mais aussi tentatives des utilisateurs locaux d'accéder à de plus hauts privilèges que ceux qui leur sont attribués, ou tentatives des administrateurs d'abuser de leurs privilèges.

Au cours de ce document nous verrons comment se protéger efficacement face à ces intrusions, mais aussi les problèmes techniques déduits de ces outils, nouvellement apparus dans le monde informatique.

Mais avant cela, il est important, pour comprendre le rôle précis de ces systèmes, de faire un rappel des principales attaques existantes à l'heure actuelle.

II. Les différents types d'attaques

L'informatique étant un domaine très vaste, le nombre de vulnérabilités présentes sur un système peut donc être important. Ainsi, les attaques visant ces failles peuvent être à la fois très variées et très dangereuses. C'est pourquoi nous allons dans un premier temps analyser ce que nous appellerons « l'anatomie d'une attaque », puis dans un second temps, nous caractériserons ces attaques et observerons leur déroulement.

Un nombre important de termes techniques vont être employés dans cette partie. Ceux-ci ne seront pas toujours détaillés, afin de ne pas surcharger la lecture ; mais une annexe comportant un rappel sur les notions fondamentales liées aux protocoles des réseaux (TCP, UDP et IP) est disponible à la fin de ce document.

1) Anatomie d'une attaque

Fréquemment appelés « les 5 P » dans la littérature, ces cinq verbes anglophones constituent le squelette de toute attaque informatique : Probe, Penetrate, Persist, Propagate, Paralyze.

Observons le détail de chacune de ces étapes :

- **Probe :** consiste en la collecte d'informations par le biais d'outils comme whois, Arin, DNS lookup. La collecte d'informations sur le système cible peut s'effectuer de plusieurs manières, comme par exemple un scan de ports grâce au programme Nmap pour déterminer la version des logiciels utilisés, ou encore un scan de vulnérabilités à l'aide du programme Nessus.

  Pour les serveurs web, il existe un outil nommé Nikto qui permet de rechercher les failles connues ou les problèmes de sécurité.

  Des outils comme firewalk, hping ou SNMP Walk permettent quant à eux de découvrir la nature d'un réseau.

- **Penetrate :** utilisation des informations récoltées pour pénétrer un réseau. Des techniques comme le brute force ou les attaques par dictionnaires peuvent être utilisées pour outrepasser les protections par mot de passe. Une autre alternative pour s'infiltrer dans un système est d'utiliser des failles applicatives que nous verrons ci-après.

- **Persist :** création d'un compte avec des droits de super utilisateur pour pouvoir se réinfiltrer ultérieurement. Une autre technique consiste à installer une application de contrôle à distance capable de résister à un reboot (ex : un cheval de Troie).

- **Propagate :** cette étape consiste à observer ce qui est accessible et disponible sur le réseau local.

- **Paralyze :** cette étape peut consister en plusieurs actions. Le pirate peut utiliser le serveur pour mener une attaque sur une autre machine, détruire des données ou encore endommager le système d'exploitation dans le but de planter le serveur.

Après ces cinq étapes, le pirate peut éventuellement tenter d'effacer ses traces, bien que cela ne soit rarement utile. En effet, les administrateurs réseaux sont souvent surchargés de logs à analyser. De plus, il est très difficile de supprimer entièrement des traces.

2) Les attaques réseaux

Ce type d'attaque se base principalement sur des failles liées aux protocoles ou à leur implémentation. Les RFC[1] ne sont parfois pas assez spécifiques, et un choix particulier d'implémentation dans les différents services ou clients peut entraîner un problème de sécurité.

Observons quelques attaques bien connues.

i. Les techniques de scan

Les scans de ports ne sont pas des attaques à proprement parler. Le but des scans est de déterminer quels sont les ports ouverts, et donc en déduire les services qui sont exécutés sur la machine cible (ex : port 80/TCP pour un service HTTP). Par conséquent, la plupart des attaques sont précédées par un scan de ports lors de la phase *Probe* qui est comme nous l'avons vu, la première phase des 5P's dans le déroulement d'une attaque.

Il existe un nombre important de techniques de scan. Idéalement, la meilleure technique de scan est celle qui est la plus furtive afin de ne pas alerter les soupçons de la future victime. Voici une description des techniques de scan les plus répandues :

- Le scan simple : aussi appelé le scan connect(), il consiste à établir une connexion TCP complète sur une suite de ports. S'il arrive à se connecter, le port est ouvert ; sinon, il est fermé. Cette méthode de scan est très facilement détectable.

- Le scan furtif : aussi appelé scan SYN, il s'agit d'une amélioration du scan simple. Ce scan essaie également de se connecter sur des ports donnés, mais il n'établit pas complètement la connexion : pas de commande ACK (acquittement) après avoir reçu l'accord de se connecter. Grâce à ceci, la méthode est bien plus furtive que le scan normal.

- Les scans XMAS, NULL et FIN : se basent sur des détails de la RFC du protocole TCP pour déterminer si un port est fermé ou non en fonction de la réaction à certaines requêtes. Ces scans sont moins fiables que le scan SYN mais ils sont un peu plus furtifs. La différence entre ces trois types de scan se situe au niveau des flags TCP utilisés lors de la requête.

- Le scan à l'aveugle : s'effectue via une machine intermédiaire et avec du spoofing (voir plus bas). Le système attaqué pense que le scan est réalisé par la machine intermédiaire et non par le pirate.

- Le scans passif : est la méthode la plus furtive. Consiste à analyser les champs d'en-tête des paquets (TTL, ToS, MSS, …) et les comparer avec une base de signatures qui pourra déterminer les applications qui ont envoyé ces paquets.

Remarque : l'utilitaire incontournable pour réaliser des scans de ports se nomme *Nmap*.

ii. IP Spoofing

But : usurper l'adresse IP d'une autre machine.

[1] Les **request for comment** (**RFC**, littéralement *demande de commentaires*) sont une série de documents et normes concernant l'Internet, commencée en 1969. Peu de RFC sont des standards, mais tous les standards de l'Internet sont enregistrés en tant que RFC.

Finalité : se faire passer pour une autre machine en truquant les paquets IP. Cette technique peut être utile dans le cas d'authentifications basées sur une adresse IP (services tels que rlogin ou ssh par exemple).

Déroulement : il existe des utilitaires qui permettent de modifier les paquets IP ou de créer ses propres paquets (ex : hping2). Grâce à ces utilitaires, il est possible de spécifier une adresse IP différente de celle que l'on possède, et ainsi se faire passer pour une autre « machine ».

Cependant, ceci pose un problème : en spécifiant une adresse IP différente de notre machine, nous ne recevrons pas les réponses de la machine distante, puisque celle-ci répondra à l'adresse spoofée. Il existe toutefois deux méthodes permettant de récupérer les réponses :

- **Source routing :** technique consistant à placer le chemin de routage directement dans le paquet IP. Cette technique ne fonctionne plus de nos jours, les routeurs rejetant cette option.

- **Reroutage :** cette technique consiste à envoyer des paquets RIP aux routeurs afin de modifier les tables de routage. Les paquets avec l'adresse spoofée seront ainsi envoyés aux routeurs contrôlés par le pirate et les réponses pourront être également reçues par celui-ci.

iii. ARP Spoofing (ou ARP Redirect)

But : rediriger le trafic d'une machine vers une autre.

Finalité : grâce à cette redirection, une personne mal intentionnée peut se faire passer pour une autre. De plus, le pirate peut rerouter les paquets qu'il reçoit vers le véritable destinataire, ainsi l'utilisateur usurpé ne se rendra compte de rien. La finalité est la même que l'IP spoofing mais on travaille ici au niveau de la couche liaison de données.

Déroulement : pour effectuer cette usurpation, il faut corrompre le cache ARP de la victime. Ce qui signifie qu'il faut lui envoyer des trames ARP en lui indiquant que l'adresse IP d'une autre machine est la sienne. Les caches ARP étant régulièrement vidés, il faudra veiller à maintenir l'usurpation.

iv. DNS Spoofing

But : fournir de fausses réponses aux requêtes DNS, c'est-à-dire indiquer une fausse adresse IP pour un nom de domaine.

Finalité : rediriger, à leur insu, des Internautes vers des sites pirates. Grâce à cette fausse redirection, l'utilisateur peut envoyer ses identifiants en toute confiance par exemple.

Déroulement : il existe deux techniques pour effectuer cette attaque.

- **DNS Cache Poisoning :** les serveurs DNS possèdent un cache permettant de garder pendant un certain temps la correspondance entre un nom de machine et son adresse IP. Le DNS Cache Poisoning consiste à corrompre ce cache avec de fausses informations. Ces fausses informations sont envoyées lors d'une réponse d'un serveur DNS contrôlé par le pirate à un autre serveur DNS, lors de la demande de l'adresse IP d'un domaine (ex : www.ledomaine.com). Le cache du serveur ayant demandé les informations est alors corrompu.

- **DNS ID Spoofing :** pour communiquer avec une machine, il faut son adresse IP. On peut toutefois avoir son nom, et grâce au protocole DNS, nous pouvons obtenir son adresse IP. Lors d'une requête pour obtenir l'adresse IP à partir d'un nom, un numéro d'identification est placé dans la trame afin que le client et le serveur puissent identifier la requête. L'attaque consiste ici à récupérer ce numéro d'identification (en sniffant le

réseau) lors de la communication entre un client et un serveur DNS, puis, envoyer des réponses falsifiées au client avant que le serveur DNS lui réponde.

Remarque : Une attaque que nous allons voir ci-après, le Déni de Service, peut aider à ralentir le trafic du serveur DNS et ainsi permettre de répondre avant lui.

v. Fragments attacks

But : le but de cette attaque est de passer outre les protections des équipements de filtrage IP.

Finalité : en passant outre les protections, un pirate peut par exemple s'infiltrer dans un réseau pour effectuer des attaques ou récupérer des informations confidentielles.

Déroulement : deux types d'attaque sur les fragments IP peuvent être distingués.

Fragments overlapping : quand un message est émis sur un réseau, il est fragmenté en plusieurs paquets IP. Afin de pouvoir reconstruire le message, chaque paquet possède un offset. Le but de l'attaque est de réaliser une demande de connexion et de faire chevaucher des paquets en spécifiant des offsets incorrects. La plupart des filtres analysant les paquets indépendamment, ils ne détectent pas l'attaque. Cependant, lors de la défragmentation, la demande de connexion est bien valide et l'attaque a lieu.

Tiny fragments : le but de l'attaque est de fragmenter une demande de connexion sur deux paquets IP : le premier paquet de taille minimum (68 octets selon la RFC du protocole IP) ne contient que l'adresse et le port de destination. Le deuxième paquet contient la demande effective de connexion TCP. Le premier paquet est accepté par les filtres puisqu'il ne contient rien de suspect. Quand le deuxième paquet arrive, certains filtres ne le vérifient pas pensant que si le premier paquet est inoffensif, le deuxième l'est aussi. Mais lors de la défragmentation sur le système d'exploitation, la connexion s'établit !

De nos jours, une grande majorité des firewalls[2] sont capables de détecter et stopper ce type d'attaques.

vi. TCP Session Hijacking

But : le but de cette attaque est de rediriger un flux TCP afin de pouvoir outrepasser une protection par mot de passe.

Finalité : le contrôle d'authentification s'effectuant uniquement à l'ouverture de la session, un pirate réussissant cette attaque parvient à prendre possession de la connexion pendant toute la durée de la session.

Déroulement : dans un premier temps, le pirate doit écouter le réseau, puis lorsqu'il estime que l'authentification a pu se produire (délai de n secondes par exemple), il désynchronise la session entre l'utilisateur et le serveur. Pour ce faire, il construit un paquet avec, comme adresse IP source, celle de la machine de l'utilisateur et le numéro d'acquittement TCP attendu par le serveur. En plus de désynchroniser la connexion TCP, ce paquet permet au pirate d'injecter une commande via la session préalablement établie.

[2] Un firewall, ou pare-feu est un dispositif logiciel ou matériel qui filtre le flux de données sur un réseau informatique. Il est parfois appelé coupe-feu.

3) Les attaques applicatives

Les attaques applicatives se basent sur des failles dans les programmes utilisés, ou encore des erreurs de configuration. Toutefois, comme précédemment, il est possible de classifier ces attaques selon leur provenance.

i. Les problèmes de configuration

Il est très rare que les administrateurs réseaux configurent correctement un programme. En général, ils se contentent d'utiliser les configurations par défaut. Celles-ci sont souvent non sécurisées afin de faciliter l'exploitation du logiciel (ex : login/mdp par défaut d'un serveur de base de données).

De plus, des erreurs peuvent apparaître lors de la configuration d'un logiciel. Une mauvaise configuration d'un serveur peut entraîner l'accès à des fichiers importants, ou mettant en jeu l'intégrité du système d'exploitation. C'est pourquoi il est important de bien lire les documentations fournies par les développeurs afin de ne pas créer de failles.

ii. Les bugs

Liés à un problème dans le code source, ils peuvent amener à l'exploitation de failles. Il n'est pas rare de voir l'exploitation d'une machine suite à une simple erreur de programmation. On ne peut toutefois rien faire contre ce type de problèmes, si ce n'est attendre un correctif de la part du développeur.

iii. Les buffer overflows

Les buffer overflows, ou dépassement de la pile, sont une catégorie de bug particulière. Issus d'une erreur de programmation, ils permettent l'exploitation d'un shellcode[3] à distance. Ce shellcode permettra à une personne mal intentionnée d'exécuter des commandes sur le système distant, pouvant aller jusqu'à sa destruction.

L'erreur de programmation est souvent la même : la taille d'une entrée n'est pas vérifiée et l'entée est directement copiée dans un buffer dont la taille est inférieure à la taille de l'entrée. On se retrouve donc en situation de débordement, et l'exploitant peut ainsi accéder à la mémoire.

iv. Les scripts

Principalement web (ex : Perl, PHP, ASP), ils s'exécutent sur un serveur et renvoie un résultat au client. Cependant, lorsqu'ils sont dynamiques (i.e. qu'ils utilisent des entrées saisies par un utilisateur), des failles peuvent apparaître si les entrées ne sont pas correctement contrôlées.

L'exemple classique est l'exploitation de fichier à distance, tel que l'affichage du fichier mot de passe du système en remontant l'arborescence depuis le répertoire web.

[3] Un shellcode est une chaîne de caractères qui représente un code binaire exécutable capable de lancer un shell ('/bin/sh' sous Unix ou command.com sous DOS et Microsoft Windows par exemple). Un shellcode peut être utilisé par un cracker voulant avoir accès à la ligne de commande.

v. Les injections SQL

Tout comme les attaques de scripts, les injections SQL profitent de paramètres d'entrée non vérifiés. Comme leur nom l'indique, le but des injections SQL est d'injecter du code SQL dans une requête de base de données. Ainsi, il est possible de récupérer des informations se trouvant dans la base (exemple : des mots de passe) ou encore de détruire des données.

vi. Man in the middle

Moins connue, mais tout aussi efficace, cette attaque permet de détourner le trafic entre deux stations. Imaginons un client C communiquant avec un serveur S. Un pirate peut détourner le trafic du client en faisant passer les requêtes de C vers S par sa machine P, puis transmettre les requêtes de P vers S. Et inversement pour les réponses de S vers C.

Totalement transparente pour le client, la machine P joue le rôle de proxy. Il accédera ainsi à toutes les communications et pourra en obtenir les informations sans que l'utilisateur s'en rende compte.

4) Le Déni de service

Evoqué précédemment, le déni de service est une attaque visant à rendre indisponible un service. Ceci peut s'effectuer de plusieurs manières : par le biais d'une surcharge réseau, rendant ainsi la machine totalement injoignable ; ou bien de manière applicative en crashant l'application à distance.

L'utilisation d'un buffer overflow peut permettre de planter l'application à distance. Grâce à quelques instructions malicieuses et suite à une erreur de programmation, une personne mal intentionnée peut rendre indisponible un service (serveur web, serveur de messagerie, … etc) voire un système complet.

Voici quelques attaques réseaux connues permettant de rendre indisponible un service :

- **SYN Flooding :** exploite la connexion en 3 phases de TCP (Three Way Handshake : SYN / SYN-ACK / ACK). Le principe est de laisser un grand nombre de connexions TCP en attente. Le pirate envoie de nombreuses demandes de connexion (SYN), reçoit les SYN-ACK mais ne répond jamais avec ACK. Les connexions en cours occupent des ressources mémoire, ce qui va entraîner une saturation et l'effondrement du système.

- **UDP Flooding** : le trafic UDP est prioritaire sur TCP. Le but est donc d'envoyer un grand nombre de paquets UDP, ce qui va occuper toute la bande passante et ainsi rendre indisponible toutes les connexions TCP.

  Exemple : faire une requête *chargen* (port 19 / service de génération de caractères) à une machine en spoofant l'adresse et le port source, pour rediriger vers *echo* (port 7 / service qui répète la chaîne de caractères reçue) d'une autre machine.

- **Packet Fragment** : utilise une mauvaise gestion de la défragmentation au niveau ICMP.

  Exemple : ping of death. La quantité de données est supérieure à la taille maximum d'un paquet IP.

  Remarque : pour rappel, nous avons vu que les techniques d'attaque se basant sur la fragmentation des paquets peuvent aussi être utilisées pour outrepasser un filtre IP.

- **Smurfling** : le pirate fait des requêtes ICMP ECHO à des adresses de broadcast en spoofant l'adresse source (en indiquant l'adresse de la machine cible). Cette machine cible va recevoir un nombre énorme de réponses, car toutes les machines vont lui répondre, et ainsi utiliser toute sa bande passante.

- **Déni de service distribué** : le but est ici de reproduire une attaque normale à grande échelle. Pour ce faire, le pirate va tenter de se rendre maître d'un nombre important de machines. Grâce à des failles (buffer overflows, failles RPC[4], … etc) il va pouvoir prendre le contrôle de machines à distance et ainsi pouvoir les commander à sa guise.

  Une fois ceci effectué, il ne reste plus qu'à donner l'ordre d'attaquer à toutes les machines en même temps, de manière à ce que l'attaque soit reproduite à des milliers d'exemplaires. Ainsi, une simple attaque comme un SYN Flooding pourra rendre une machine ou un réseau totalement inaccessible.

5) Actuellement

La sécurité contre les attaques distantes se renforce, notamment par le biais d'équipements réseaux plus puissants (comme des firewalls plus intelligents), mais les attaques locales restent toutefois encore fort efficaces : l'ARP Spoofing, le vol de session, … restent souvent possibles.

L'informatique évolue, les applications sont de plus en plus complexes et les délais laissés aux programmeurs et administrateurs sont souvent très (trop) courts. Les risques de failles applicatives sont, de ce fait, très grands et peuvent s'avérer dangereux pour des applications largement répandues.

Les attaques distribuées seront toujours redoutables si la plupart des machines personnelles ne sont pas protégées. Ce qui nous amène à notre seconde partie : comment détecter et empêcher ces attaques ?

III. Détection d'attaques : les IDS

Afin de détecter les attaques que peut subir un système, il est nécessaire d'avoir un logiciel spécialisé dont le rôle serait de surveiller les données qui transitent sur ce système, et qui serait capable de réagir si des données semblent suspectes. Plus communément appelé IDS (Intrusion Detection Systems), les systèmes de détection d'intrusions conviennent parfaitement pour réaliser cette tâche.

A l'origine, les premiers systèmes de détection d'intrusions ont été initiés par l'armée américaine, puis par des entreprises. Plus tard, des projets open-source ont été lancés et certains furent couronnés de succès, comme par exemple Snort et Prelude que nous détaillerons par après. Parmi les solutions commerciales, on retrouve les produits des entreprises spécialisées en sécurité informatique telles que *Internet Security Systems*, *Symantec*, *Cisco Systems*, …

1) Les différents types d'IDS

Comme nous l'avons vu, les attaques utilisées par les pirates sont très variées. Certaines utilisent des failles réseaux et d'autres des failles de programmation. Nous pouvons donc facilement comprendre que la détection d'intrusions doit se faire à plusieurs niveaux.

[4] RPC (**R**emote **P**rocedure **C**all) est un protocole permettant de faire des appels de procédures sur un ordinateur distant à l'aide d'un serveur d'application. Ce protocole est utilisé dans le modèle client-serveur et permet de gérer les différents messages entre ces entités.

Ainsi, il existe différents types d'IDS dont nous détaillons ci-dessous les caractéristiques principales.

i. Les systèmes de détection d'intrusions (IDS)

Définition : ensemble de composants logiciels et matériels dont la fonction principale est de détecter et analyser toute tentative d'effraction (volontaire ou non).

Fonctions : *détection* des techniques de sondage (balayages de ports, fingerprinting), des tentatives de compromission de systèmes, d'activités suspectes internes, des activités virales ou encore audit des fichiers de journaux (logs).

Remarque : utopiquement, il s'agit d'un système capable de détecter tout type d'attaque.

Certains termes sont souvent employés quand on parle d'IDS :

- Faux positif : une alerte provenant d'un IDS mais qui ne correspond pas à une attaque réelle.

- Faux négatif : une intrusion réelle qui n'a pas été détectée par l'IDS

ii. Les systèmes de détection d'intrusions « réseaux » (NIDS)

Objectif : analyser de manière passive les flux en transit sur le réseau et détecter les intrusions en temps réel.

Un NIDS écoute donc tout le trafic réseau, puis l'analyse et génère des alertes si des paquets semblent dangereux.

Les NIDS étant les IDS plus intéressants et les plus utiles du fait de l'omniprésence des réseaux dans notre vie quotidienne, ce document se concentrera essentiellement sur ce type d'IDS.

iii. Les systèmes de détection d'intrusions de type hôte (HIDS)

Un HIDS se base sur une unique machine, n'analysant cette fois plus le trafic réseau mais l'activité se passant sur cette machine. Il analyse en temps réel les flux relatifs à une machine ainsi que les journaux.

Un HIDS a besoin d'un système sain pour vérifier l'intégrité des donnés. Si le système a été compromis par un pirate, le HIDS ne sera plus efficace. Pour parer à ces attaques, il existe des KIDS (Kernel Intrusion Detection System) et KIPS (Kernel Intrusion Prevention System) qui sont fortement liés au noyau. Ces types d'IDS sont décrits un peu plus loin.

iv. Les systèmes de détection d'intrusions « hybrides »

Généralement utilisés dans un environnement décentralisé, ils permettent de réunir les informations de diverses sondes placées sur le réseau. Leur appellation « hybride » provient du fait qu'ils sont capables de réunir aussi bien des informations provenant d'un système HIDS qu'un NIDS.

L'exemple le plus connu dans le monde Open-Source est Prelude. Ce framework permet de stocker dans une base de données des alertes provenant de différents systèmes relativement variés. Utilisant Snort comme NIDS, et d'autres logiciels tels que Samhain en tant que HIDS, il permet de combiner des outils puissants tous ensemble pour permettre une visualisation centralisée des attaques.

Remarque : nous parlerons de tous ces produits plus tard dans ce document, en évoquant les spécificités et l'installation de chacun.

v. Les systèmes de prévention d'intrusions (IPS)

Définition : ensemble de composants logiciels et matériels dont la fonction principale est **d'empêcher** toute activité suspecte détectée au sein d'un système.

Contrairement aux IDS simples, les IPS sont des outils aux fonctions « actives », qui en plus de détecter une intrusion, tentent de la bloquer. Cependant, les IPS ne sont pas la solution parfaite comme on pourrait le penser.

Plusieurs stratégies de prévention d'intrusions existent :

- host-based memory and process protection ➔ surveille l'exécution des processus et les tue s'ils ont l'air dangereux (buffer overflow). Cette technologie est utilisée dans les KIPS (Kernel Intrusion Prevention System) que nous décrivons un peu plus loin.

- session interception / session sniping ➔ termine une session TCP avec la commande TCP Reset : « *RST* ». Ceci est utilisé dans les NIPS (Network Intrusion Prevention System).

- gateway intrusion detection ➔ si un système NIPS est placé en tant que routeur, il bloque le trafic ; sinon il envoie des messages à d'autres routeurs pour modifier leur liste d'accès.

Un IPS possède de nombreux inconvénients. Le premier est qu'il bloque toute activité qui lui semble suspecte. Or, il est impossible d'assurer une fiabilité à 100% dans l'identification des attaques. Un IPS peut donc malencontreusement bloquer du trafic inoffensif ! Par exemple, un IPS peut détecter une tentative de déni de service alors qu'il s'agit simplement d'une période chargée en trafic. Les faux positifs sont donc très dangereux pour les IPS.

Le deuxième inconvénient est qu'un pirate peut utiliser sa fonctionnalité de blocage pour mettre hors service un système. Prenons l'exemple d'un individu mal intentionné qui attaque un système protégé par un IPS, tout en spoofant son adresse IP. Si l'adresse IP spoofée est celle d'un noeud important du réseau (routeur, service Web, ...), les conséquences seront catastrophiques. Pour palier ce problème, de nombreux IPS disposent des « white lists », c'est-à-dire des listes d'adresses réseaux qu'il ne faut en aucun cas bloquer.

Et enfin, le troisième inconvénient et non le moindre : un IPS est peu discret. En effet, à chaque blocage d'attaque, il montre sa présence. Cela peut paraître anodin, mais si un pirate remarque la présence d'un IPS, il tentera de trouver une faille dans celui-ci afin de réintégrer son attaque... mais cette fois en passant inaperçu.

Voilà pourquoi les IDS passifs sont souvent préférés aux IPS. Cependant, il est intéressant de noter que plusieurs IDS (Ex : Snort, RealSecure, Dragon, ...) ont été dotés d'une fonctionnalité de réaction automatique à certains types d'attaques.

vi. Les systèmes de prévention d'intrusions « kernel » (KIDS/KIPS)

Nous l'évoquions précédemment dans le cadre du HIDS, l'utilisation d'un détecteur d'intrusions au niveau noyau peut s'avérer parfois nécessaire pour sécuriser une station.

Prenons l'exemple d'un serveur web, sur lequel il serait dangereux qu'un accès en lecture/écriture dans d'autres répertoires que celui consultable via http, soit autorisé. En effet, cela pourrait nuire à l'intégrité du système. Grâce à un KIPS, tout accès suspect peut être bloqué directement par le noyau, empêchant ainsi toute modification dangereuse pour le système.

Le KIPS peut reconnaître des motifs caractéristiques du débordement de mémoire, et peut ainsi interdire l'exécution du code. Le KIPS peut également interdire l'OS d'exécuter un appel système qui ouvrirait un shell de commandes.

Puisqu'un KIPS analyse les appels systèmes, il ralentit l'exécution. C'est pourquoi ce sont des solutions rarement utilisées sur des serveurs souvent sollicités.

Exemple de KIPS : SecureIIS, qui est une surcouche du serveur IIS de Microsoft.

vii. Les firewalls

Les firewalls ne sont pas des IDS à proprement parler mais ils permettent également de stopper des attaques. Nous ne pouvions donc pas les ignorer.

Les firewalls sont basés sur des règles statiques afin de contrôler l'accès des flux. Ils travaillent en général au niveau des couches basses du modèle OSI (jusqu'au niveau 4), ce qui est insuffisant pour stopper une intrusion. Par exemple, lors de l'exploitation d'une faille d'un serveur Web, le flux HTTP sera autorisé par le firewall puisqu'il n'est pas capable de vérifier ce que contiennent les paquets.

Il existe trois types de firewalls :

- Les systèmes à filtrage de paquets sans état : analyse les paquets les uns après les autres, de manière totalement indépendante.

- Les systèmes à maintien d'état (stateful) : vérifient que les paquets appartiennent à une session régulière. Ce type de firewall possède une table d'états où est stocké un suivi de chaque connexion établie, ce qui permet au firewall de prendre des décisions adaptées à la situation.

  Ces firewalls peuvent cependant être outrepassés en faisant croire que les paquets appartiennent à une session déjà établie.

- Les firewalls de type proxy : Le firewall s'intercale dans la session et analyse l'information afin de vérifier que les échanges protocolaires sont conformes aux normes.

viii. Les technologies complémentaires

Les scanners de vulnérabilités : systèmes dont la fonction est d'énumérer les vulnérabilités présentes sur un système. Ces programmes utilisent une base de vulnérabilités connues (exemple : Nessus).

Les systèmes de leurre : le but est de ralentir la progression d'un attaquant, en générant des fausses réponses telle que renvoyer une fausse bannière du serveur Web utilisé.

Les systèmes de leurre et d'étude (Honeypots) : le pirate est également leurré, mais en plus, toutes ses actions sont enregistrées. Elles seront ensuite étudiées afin de connaître les mécanismes d'intrusion utilisés par le hacker. Il sera ainsi plus facile d'offrir des protections par la suite.

Les systèmes de corrélation et de gestion des intrusions (SIM – Security Information Manager) : centralisent et corrèlent les informations de sécurité provenant de plusieurs sources (IDS, firewalls, routeurs, applications, …). Les alertes sont ainsi plus faciles à analyser.

Les systèmes distribués à tolérance d'intrusion : l'information sensible est répartie à plusieurs endroits géographiques mais des copies de fragments sont archivées sur différents sites pour assurer la disponibilité de l'information. Cependant, si un pirate arrive à s'introduire sur le système, il n'aura qu'une petite partie de l'information et celle-ci lui sera inutile.

2) Les méthodes de détection

Pour bien gérer un système de détection d'intrusions, il est important de comprendre comment celui-ci fonctionne. Une question simple se pose alors : comment une intrusion est-elle détectée par un tel système ? Quel critère différencie un flux contenant une attaque d'un flux normal ?

Ces questions nous ont amenés à étudier le fonctionnement interne d'un IDS. De là, nous en avons déduit deux techniques mises en place dans la détection d'attaques. La première consiste à détecter des signatures d'attaques connues dans les paquets circulant sur le réseau. La seconde, consiste quant à elle, à détecter une activité suspecte dans le comportement de l'utilisateur.

Ces deux techniques, aussi différentes soient-elles, peuvent être combinées au sein d'un même système afin d'accroître la sécurité.

i. L'approche par scénario (misuse detection)

Cette technique s'appuie sur la connaissance des techniques utilisées par les attaquants pour déduire des scénarios typiques. Elle ne tient pas compte des actions passées de l'utilisateur et utilise des signatures d'attaques (= ensemble de caractéristiques permettant d'identifier une activité intrusive : une chaîne alphanumérique, une taille de paquet inhabituelle, une trame formatée de manière suspecte, …).

Recherche de motifs (pattern matching)

La méthode la plus connue et la plus à facile à comprendre. Elle se base sur la recherche de motifs (chaînes de caractères ou suite d'octets) au sein du flux de données. L'IDS comporte une base de signatures où chaque signature contient les protocole et port utilisés par l'attaque ainsi que le motif qui permettra de reconnaître les paquets suspects.

Le principal inconvénient de cette méthode est que seules les attaques reconnues par les signatures seront détectées. Il est donc nécessaire de mettre à jour régulièrement le base de signatures.

Un autre inconvénient est que les motifs sont en général fixes. Or une attaque n'est pas toujours identique à 100%. Le moindre octet différent par rapport à la signature provoquera la non détection de l'attaque.

Pour les IDS utilisant cette méthode, il est nécessaire d'adapter la base de signatures en fonction du système à protéger. Cela permet non seulement de diminuer les ressources nécessaires et donc augmenter les performances ; mais également réduire considérablement

le nombre de fausses alertes et donc faciliter le travail des administrateurs réseaux qui analyseront les fichiers d'alertes.

Cette technique est également utilisée dans les anti-virus.

Recherche de motifs dynamiques

Le principe de cette méthode est le même que précédemment mais les signatures des attaques évoluent dynamiquement. L'IDS est de ce fait doté de fonctionnalités d'adaptation et d'apprentissage.

Analyse de protocoles

Cette méthode se base sur une vérification de la conformité (par rapport aux RFC) des flux, ainsi que sur l'observation des champs et paramètres suspects dans les paquets. Cependant, les éditeurs de logiciels et les constructeurs respectent rarement à la lettre les RFC et cette méthode n'est pas toujours très performante.

L'analyse protocolaire est souvent implémentée par un ensemble de préprocesseurs, où chaque préprocesseur est chargé d'analyser un protocole particulier (FTP, HTTP, ICMP, ...). Du fait de la présence de tous ces préprocesseurs, les performances dans un tel système s'en voient fortement dégradées.

L'intérêt fort de l'analyse protocolaire est qu'elle permet de détecter des attaques inconnues, contrairement au pattern matching qui doit connaître l'attaque pour pouvoir la détecter.

Analyse heuristique et détection d'anomalies

Le but de cette méthode est, par une analyse intelligente, de détecter une activité suspecte ou toute autre anomalie.

Par exemple : une analyse heuristique permet de générer une alarme quand le nombre de sessions à destination d'un port donné dépasse un seuil dans un intervalle de temps prédéfini.

ii. L'approche comportementale (Anomaly Detection)

Cette technique consiste à détecter une intrusion en fonction du comportement passé de l'utilisateur. Pour cela, il faut préalablement dresser un profil utilisateur à partir de ses habitudes et déclencher une alerte lorsque des événements hors profil se produisent.

Cette technique peut être appliquée non seulement à des utilisateurs mais aussi à des applications et services. Plusieurs métriques sont possibles : la charge CPU, le volume de données échangées, le temps de connexion sur des ressources, la répartition statistique des protocoles et applications utilisés, les heures de connexion, …

Cependant elle possède quelques inconvénients :

- peu fiable : tout changement dans les habitudes de l'utilisateur provoque une alerte.
- nécessite une période de non fonctionnement pour mettre en œuvre les mécanismes d'auto-apprentissage : si un pirate attaque pendant ce moment, ses actions seront assimilées à un profil utilisateur, et donc passeront inaperçues lorsque le système de détection sera complètement mis en place.
- l'établissement du profil doit être souple afin qu'il n'y ait pas trop de fausses alertes : le pirate peut discrètement intervenir pour modifier le profil de l'utilisateur afin d'obtenir

après plusieurs jours ou semaines, un profil qui lui permettra de mettre en place son attaque sans qu'elle ne soit détectée.

Plusieurs approches peuvent être utilisées pour la méthode de détection comportementale :

Approche probabiliste

Des probabilités sont établies permettant de représenter une utilisation courante d'une application ou d'un protocole. Toute activité ne respectant pas le modèle probabiliste provoquera la génération d'une alerte.

Exemple : Avec le protocole HTTP, il y a une probabilité de 0.9 qu'une commande GET soit faite après une connexion sur le port 80. Il y a ensuite une probabilité de 0.8 que la réponse à cette commande GET soit « HTTP/1.1 200 OK ».

Approche statistique

Le but est de quantifier les paramètres liés à l'utilisateur : taux d'occupation de la mémoire, utilisation des processeurs, valeur de la charge réseau, nombre d'accès à l'Intranet par jour, vitesse de frappe au clavier, sites les plus visités, …

Cette méthode est très difficile à mettre en place. Elle n'est actuellement présente que dans le domaine de la recherche, où les chercheurs utilisent des réseaux neuronaux et le data mining pour tenter d'avoir des résultats convaincants.

Autres méthodes

D'autres méthodes existent mais ne sont pas encore répandues. Parmi celles-ci, nous pouvons noter :

- L'utilisation de l'immunologie, c'est-à-dire construire un modèle de comportement normal des services.
- La présentation d'une activité habituelle sous forme de graphe.

iii. Les méthodes répandues

En général, les IDS mélangent les différentes techniques de détection par scénario en proposant du pattern matching, de l'analyse protocolaire et de la détection d'anomalies.

De nombreuses techniques et algorithmes sont utilisés dans la détection d'intrusions :

Pattern Matching ➔ algorithmes de recherche de motifs (ex : Boyer-Moore) ,
➔ algorithmes de comptage
➔ algorithmes génétiques

Analyse Protocolaire ➔ conformité aux RFC

Détection d'anomalies ➔ méthodes heuristiques

Analyse statistique ➔ modèles statistiques

Analyse probabiliste ➔ réseaux bayésiens

Autre analyse comportementale ➔ réseaux de neurones, systèmes experts + data mining, immunologie, graphes, ...

Il est bien sûr impossible de détailler chacun des algorithmes mis en oeuvre dans les IDS. Nous avons cependant dédié le chapitre 6 aux algorithmes de pattern matching.

3) Principes généraux et installation technique

Jusqu'à présent, nous avons vu les types d'IDS existants et les méthodes de détection qu'ils utilisent. Nous allons maintenant détailler une étape importante dans la mise en place d'un IDS : l'installation technique.

Lors de la mise en place d'un système de détection d'intrusions au sein d'un réseau, il est important de le déployer correctement d'une part, mais aussi de comprendre son fonctionnement interne pour pouvoir le configurer efficacement. Toute erreur lors de l'installation d'un IDS pourra le rendre inefficace ou inutilisable.

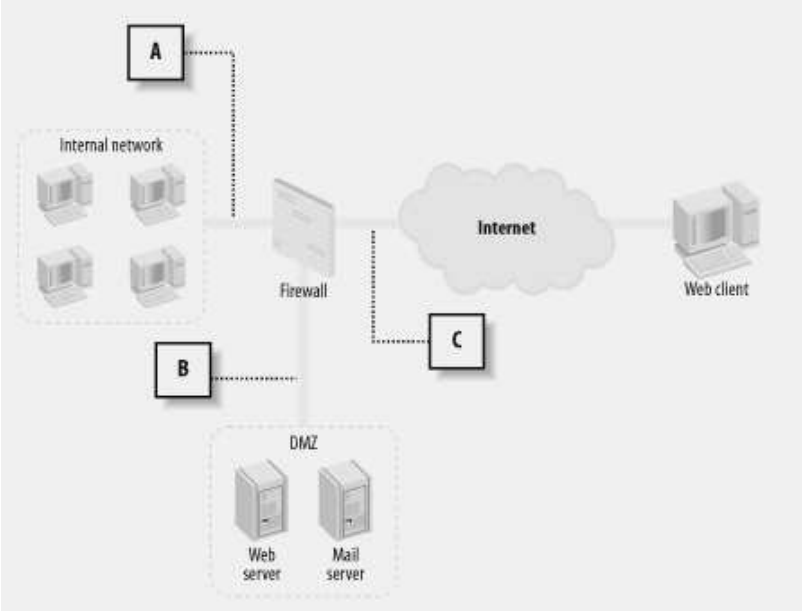
i. Déploiement d'un NIDS

Il faut tout d'abord prendre conscience qu'un NIDS n'est pas suffisant pour assurer la sécurité. En plus d'installer un NIDS, il ne faudra pas oublier les actions habituelles :

- les systèmes et les applications doivent être mises à jour régulièrement (patches de sécurité)
- les systèmes utilisant Internet doivent être dans un réseau isolé (DMZ[5])
- chaque utilisateur doit être averti de l'importance de la sécurité de ses mots de passe
- les fonctionnalités des services qui ne sont pas utilisées doivent être désactivées.

Lors du déploiement d'un IDS, il faut le configurer correctement : par exemple, si le réseau est sous Windows, les règles destinées à Unix ne sont pas nécessaires. Il faut donc faire une configuration en fonction de l'OS, des applications et du matériel utilisés.

L'emplacement du senseur[6] est très important :



[5] Une **zone démilitarisée** (DMZ) est un sous-réseau isolé par un pare-feu. Ce sous-réseau contient des machines se situant entre un réseau interne (LAN) et un réseau externe (typiquement, Internet).
[6] Un **senseur** est une sonde, placée sur le réseau, dont le rôle est d'attraper le trafic circulant sur celui-ci.

➔ A l'emplacement B, seul le trafic entre les systèmes de la DMZ et Internet est analysé. Le trafic entre le réseau interne et Internet n'est pas analysé. Pour cela, il faudra également placer un senseur au point A.

➔ A l'emplacement C, le trafic entre Internet et le réseau interne ou la DMZ est analysé. Par contre, le trafic entre le réseau interne et la DMZ est invisible.

Il est impensable de vouloir analyser tout le trafic d'un réseau. Il faut donc donner la priorité aux systèmes à risque : ceux qui offrent des services accessibles par Internet (HTTP, FTP, ...).

De plus, il est souvent préférable de placer le senseur après le firewall du côté interne. Ainsi, seul les flux acceptés par le firewall sont analysés, ce qui réduit fortement la charge de la sonde IDS.

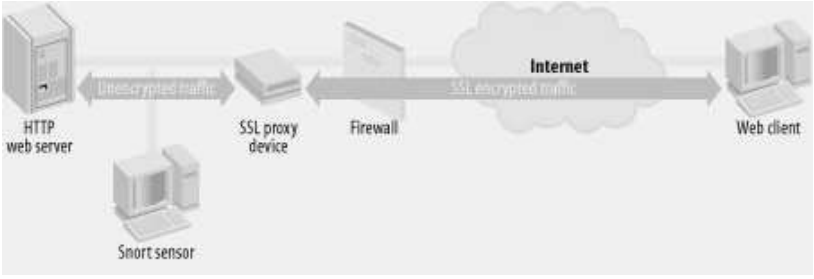
Lors de l'installation d'un NIDS, le choix matériel a également une grande importance. Puisqu'une sonde NIDS doit être capable d'analyser le trafic réseau quelque soit le destinataire, elle devra recevoir elle-même tous les paquets. Pour cela, le NIDS devra jouer le rôle d'un sniffer[7], mais le matériel réseau pose parfois problème :

- un switch simple (commutateur) : en utilisant un tel équipement, la conversation avec l'IDS est impossible du fait de la nature d'un switch : il commute les paquets directement au destinataire. Il est dès lors impossible d'installer une sonde qui analysera le trafic global.

- un hub (concentrateur) → la conversation avec l'IDS est possible car les hubs répètent les paquets en les émettant à toutes les machines connectées. Cependant, les hubs sont peu fiables et sont donc à éviter.

- il existe des switches professionnels qui copie le trafic et l'envoie sur un port spécifié (où sera placé le NIDS) : SPAN port (Switch Port Analyzer). **Attention** : pour ce port, il faudra utiliser une connexion rapide (ex : Gigabits) capable d'analyser entièrement le trafic provenant ou à destination des différents sous réseaux.

Un autre problème doit être pris en considération : lorsque les flux sont cryptés (ex : par SSL, c'est le cas pour les VPN[8]), il est impossible pour l'IDS de décrypter ces flux. Il faut dans ce cas utiliser un proxy SSL comme illustré ci-dessous.



Nous savons donc maintenant que l'emplacement des sondes et le matériel réseau utilisés sont très importants. Cependant, un autre point ne doit pas être oublié : la sécurisation du senseur et des logs d'alerte. En effet, si la sonde elle-même ou si les alertes qu'elles génèrent ne sont pas sécurisées, un pirate pourrait très bien rendre l'IDS complètement inefficace.

[7] Un **sniffer** est une machine écoutant toutes les données circulant sur le réseau.
[8] Le **Réseau privé virtuel** (*VPN* ou *Virtual Private Network*, en anglais), est une extension des réseaux locaux qui procure une norme de sécurité en télécommunications.

Pour sécuriser les sondes et les fichiers d'alertes, il est par exemple possible de mettre en place un réseau de management très contrôlé, avec son propre firewall. Ce système sera primordial pour la sécurité du réseau et plusieurs mesures devront être prises pour assurer son fonctionnement :

- Le système d'exploitation du senseur devra tenu à jour
- Un système d'authentification robuste (ex : PKI) pourra être mis en place pour renforcer la sécurité.
- Tous les mots de passe devront être changés régulièrement
- Il est également possible d'utiliser deux interfaces réseaux pour le senseur : la première pour générer les alertes et contrôler le trafic ; et la deuxième, complètement invisible, en tant que point de monitoring. Ce genre d'interface est communément appelée *stealth interface*.

ii. Problèmes techniques

Le premier problème est de configurer correctement l'IDS afin qu'il n'inonde pas les rapports d'alertes avec des faux positifs. La présence de faux positifs semble inoffensive. Or, s'ils sont trop nombreux, les rapports d'alertes seront longs à analyser. Par conséquent, les administrateurs passeront beaucoup de temps à distinguer un faux positif d'une véritable intrusion. Et de plus, en voyant toutes ces fausses alertes, ils auront tendance à minimiser le risque d'attaque.

Bien sûr, il faut également veiller à ce que l'affinement de la configuration ne génère pas des faux négatifs car toute intrusion non détectée peut avoir des conséquences dramatiques.

Le deuxième problème provient des débits actuels sur les réseaux : ces débits augmentent de plus en plus, et les IDS ont de plus en plus de paquets à traiter et à analyser. En plus d'avoir un équipement réseau performant, l'IDS doit utiliser des algorithmes adaptés et optimisés.

Afin de bien comprendre comment améliorer les performances d'un NIDS, il faut comprendre les différentes étapes. Chaque paquet de données traité par l'IDS va subir une suite de traitements :

- capture de la trame par l'interface en mode promiscuité (promiscous mode)
- analyse de la trame et filtrage éventuel en bas niveau
- détection de la présence de fragments ou non et passage éventuel à un moteur de reconstruction
- transfert de la trame vers le système d'exploitation
- filtrage éventuel
- applications de divers préprocesseurs en fonction du type de requête afin de contrer des techniques d'évasion d'attaques (voir plus loin)
- passage vers le moteur d'analyse (protocole, pattern matching, statistique, …)

Pour améliorer les performances de l'IDS, il peut donc être judicieux de répartir les charges. Par exemple, il est envisageable de séparer les flux à analyser en fonction du protocole de niveau 4 : une sonde pour l'analyse des flux Web, une autre pour l'analyse des flux FTP et une analyse des requêtes SQL.

Un autre problème est la corrélation des informations provenant de plusieurs types de sondes. Voici les actions à réaliser pour regrouper ces informations :

- agrégation : rassembler les informations des différentes sondes
- fusion : fusionner en supprimant les doublons (même attaque détectée par plusieurs sondes)
- corrélation : définir un motif commun, c'est-à-dire interpréter une suite d'événements et les résumer

Une corrélation intéressante serait de ne garder que les alertes qui concernent une faille probable du système. Pour cela il faut utiliser un scanner de vulnérabilités par exemple, ou ne pas afficher les alertes concernant IIS si on possède Apache, ce qui entraînera moins de faux positifs.

Néanmoins, l'utilisation d'un scanner de vulnérabilités n'est pas parfaite car il est difficile d'échanger des informations entre un scanner de vulnérabilités et un IDS. Cependant, depuis peu, des consoles de corrélation entre IDS et scanners de vulnérabilités sont proposées (ex : *Nevo* de Tenable Network Security).

Enfin, un dernier problème est qu'il n'y a aucune interopérabilité entre les différents IDS du marché, mis à part la possibilité d'exporter les informations dans des formats standard. Pourtant, des normes ont été établies, comme nous allons le voir.

iii. Complémentarité des IDS

Nous avons vu qu'il existe plusieurs types d'IDS, dont leur rôle est complètement différent. Plus exactement, leurs rôles sont complémentaires. En effet, un NIDS ne fera qu'analyser le trafic réseau. Mais complété par un HIDS, des intrusions non détectées sur le réseau pourront l'être lorsqu'elles atteindront la machine cible.

En général, un seul NIDS par réseau est suffisant. Mais il est possible de placer des sondes à différents endroits du réseau afin de répartir la charge.

L'idéal pour les HIDS serait d'en déployer sur toutes les machines du parc informatique mais cela n'est rarement fait pour des raisons de coût et d'exploitation. Un compromis souvent choisi est d'installer des agents HIDS sur toutes les machines de la DMZ, ainsi que sur les serveurs importants.

Les KIDS/KIPS ralentissant énormément l'exécution des programmes, ils sont souvent délaissés ou configurés de façon à n'utiliser que les fonctionnalités de base. Pourtant, ils sont les seuls à pouvoir détecter de manière efficace les tentatives de buffer overflows. Ils sont donc très conseillés sur les machines sensibles.

4) Normalisation

Il y a quelques années, un comité du DARPA a défini 4 briques fonctionnelles pour décrire l'architecture globale d'un IDS :

- Générateur d'événements (boîte E) : envoie des événements à la boîte A
- Analyseur d'événements (boîte A) : produit des alertes
- Base de données événementielle (boîte D)
- Système de réponse (boîte R) : réponse en temps réel face aux attaques

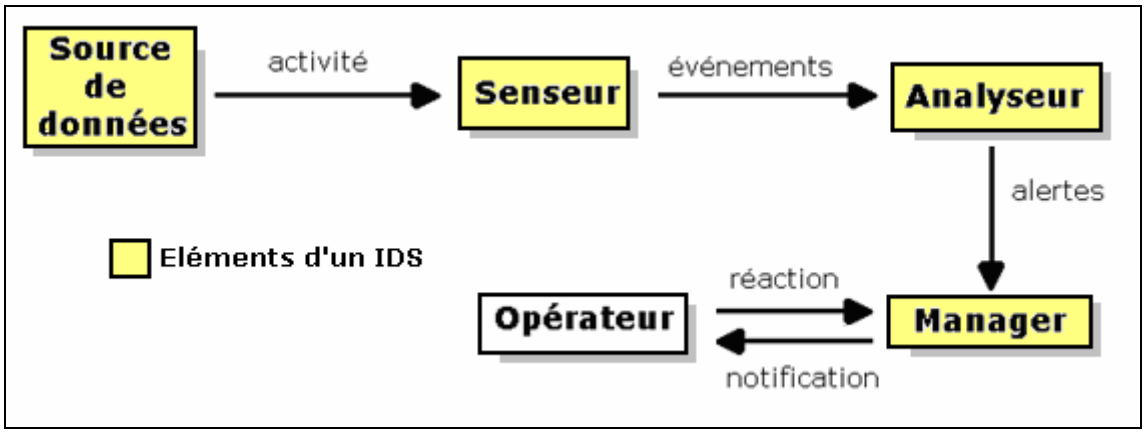
Ce comité a aussi défini un langage de description des intrusions (CISL – Common Intrusion Specification Language) qui utilise des expressions verbales (ex : « ouvrir session », « effacer objet », …). Cependant, ce langage n'a jamais été utilisé mais il a inspiré d'autres comités.

L'IDWG (Intrusion Detection Working Group) a effectué la plupart des travaux dans le domaine de la standardisation des IDS :

➔ norme IDMEF (Intrusion Detection Message Exchange Format) : définit le format des messages échangés dans un IDS.

➔ protocole IDXP (Intrusion Detection eXchange Protocol) : procédures de transport entre les entités de l'IDS.

Selon l'IDWG, un IDS est ensemble de plusieurs senseurs, analyseurs et managers :



C'est un schéma théorique, rarement implémenté de cette façon dans les IDS. La norme IDMEF préconise une représentation en XML des messages où dans chaque message, on retrouve l'ID de l'analyseur, le type d'alerte, l'emplacement (le nœud réseau), le jour et l'heure, l'adresse, la classification de l'attaque, …

5) Techniques anti-IDS

Comme tout système informatique, ou presque, il existe des failles dans les IDS, ou plutôt des techniques qui permettent d'outrepasser ces systèmes sans se faire repérer. Si un pirate détecte la présence d'un IDS, il peut le désactiver, ou mieux encore, générer de fausses attaques pendant qu'il commettra son forfait tranquillement.

Il existe trois catégories d'attaques contre les IDS :

- Attaque par déni de service : rendre l'IDS inopérant en le saturant.

- Attaque par insertion : le pirate, pour éviter d'être repéré, injecte des paquets de leurre qui seront ignorés par le système d'exploitation de la cible mais pris en compte par l'IDS : l'IDS ne détecte rien d'anormal, alors que sur le système cible, l'attaque a bien lieu puisque les paquets superflus sont ignorés.

- Attaque par évasion : il s'agit de la technique inverse à l'attaque par insertion. Ici, des données superflues sont ignorées par l'IDS mais prises en compte par le système d'exploitation. Nous détaillons un peu plus loin quelques techniques d'évasion Web.

i. Détecter un IDS

Comme nous l'avons déjà signalé, il est très dangereux que la présence d'un IDS soit remarquée par un pirate. Car dans ce cas, il tentera d'obtenir un maximum d'informations sur l'IDS installé (ex : la version utilisée) pour pouvoir l'outrepasser et attaquer sans se faire remarquer. Voici quelques techniques qui permettent de détecter un IDS :

Usurpation d'adresse MAC : les NIDS mettent l'interface de capture en mode promiscuité (promiscuous mode), il est donc possible de détecter l'IDS en envoyant par exemple un ICMP « *echo request* » à la machine soupçonnée d'être un NIDS avec une adresses MAC inexistante. Si la machine répond alors elle est en mode promiscuous et peut donc être un NIDS.

Mesure des temps de latence : puisque l'interface est en mode promiscuous, les temps de réponse sont plus longs. Voici une méthode pour exploiter ces temps de latence :

- le pirate génère une série de pings vers l'adresse à tester, puis il mesure et note les temps de réponses.
- le pirate sature ensuite le réseau en broadcast[9] dans le but de ralentir l'IDS, qui recevra tous les paquets. Enfin, le pirate réémet la même série de pings en mesurant les nouveaux temps de réponse. S'ils sont bien plus élevés que les premiers temps obtenus, il est fort possible que la machine soit en mode promiscous.

Exploiter les mécanismes de réponses actives : Les IPS réagissent à certaines attaques (fermer session, bloquer port, …) mais en faisant cela, ils laissent souvent des empreintes (header des paquets) permettant d'identifier le type d'IPS.

Observation des requêtes DNS : Les IDS génèrent souvent des requêtes DNS lors des alertes. En observant le DNS primaire lors de fausses attaques, on peut détecter qu'il y a un IDS.

ii. Déni de services contre un IDS

Le but est de désactiver l'IDS en saturant ses ressources (ex : SYN Flood ou paquets fragmentés incomplets). L'IDS sera dès lors incapable d'exécuter sa fonctionnalité de détection, et le pirate pourra réaliser son attaque.

iii. Techniques d'insertion

Comme nous l'avons vu, ces techniques consiste à injecter des données supplémentaires de telle sorte que :

- l'IDS les estiment inoffensives
- la cible ne les décode pas

Voici quelques méthodes permettant d'utiliser des techniques d'insertion :

✓ en utilisant la fragmentation IP qui est gérée de manière différente selon l'OS lors de cas anormaux (ex : recouvrement de paquets) : soit les fragments anciens sont favorisés, soit les nouveaux le sont. Par exemple, il est donc possible que les IDS favorisent les anciens paquets alors que le système d'exploitation utilisé favorise les nouveaux.

[9] Le **broadcast** est un terme anglais (en français on utilise le terme **diffusion**) définissant une diffusion de données à un ensemble d'ordinateurs connectés à un réseau informatique.

Pour utiliser le recouvrement de fragments (*fragmentation overlap*), il faut modifier artificiellement les champs « longueur » et « décalage » (offset) des fragments IP.

- ✓ en utilisant l'écrasement de fragments (*fragmentation overwrite*) : même principe que précédemment mais des fragments entiers sont remplacés, et non seulement des parties de paquets.

- ✓ en utilisant le timeout de fragmentation : les systèmes conservent en général les fragments pendant 60 secondes pour le réassemblage ; mais les IDS les gardent souvent moins longtemps. On peut donc espacer les fragments dans le temps pour ne pas se faire repérer par l'IDS tout en réalisant une attaque complète sur le système d'exploitation.

- ✓ découpage de sessions TCP (*session splicing*) : la requête TCP est divisée en paquets, tout en modifiant le numéro de séquence pour créer des recouvrements : même principe qu'avec la fragmentation IP + possibilité de timeout (Apache Linux : 5min dans tampon, IIS : 10 min). L'un des outils pour réaliser ce genre d'attaques se nomme *fragroute*.

- ✓ insérer un faux paquet avec checksum erroné : certains IDS ne verront pas l'attaque car peu d'IDS vérifient le checksum. Le système, par contre, rejettera le paquet erroné.

  Attention : de nos jours, les routeurs rejettent souvent les paquets erronés ; il faut donc utiliser un autre champ que le checksum, comme par exemple le TTL.

iv. Techniques d'évasion

Nous présentons maintenant quelques techniques d'évasion. Pour rappel, ces techniques ont pour but d'insérer des données qui seront ignorées par l'IDS mais qui ne gêneront nullement l'attaque.

- Evasions HTTP : le principe est de modifier la syntaxe des URL mais sans changer la sémantique. La première personne à avoir présenté ce genre d'attaques avait pour pseudonyme Rain Forrest Puppy qui est l'auteur du très célèbre outil *Whisker*, un scanner de vulnérabilités Web.

  Voici quelques exemples d'évasion HTTP qui permettaient à une époque d'outrepasser les signatures de nombreux NIDS, tout en émettant des requêtes HTTP valides. Bien sûr, les NIDS prennent maintenant en compte ces techniques.

  ➔ faire une requête HEAD au lieu de GET.

  ➔ encoder les URL en hexadécimal, cette forme d'encodage étant tout à fait valide selon la RFC du protocole HTTP.

  exemple : www%2Emonsite%2Ecom%2Fcgi%2Fscript

  ➔ utiliser des doubles slashes au lieu de slashes simples

  exemple : www.monsite.com//cgi//script

  ➔ traverser des répertoires fantômes

  exemple : www.monsite.com/repFantome/../cgi/script

  ➔ utiliser des auto-références

  exemple : www.monsite.com/cgi/././.script

➔ simuler la fin d'une requête prématurée avec le caractère de fin de chaîne %00 ou bien avec les caractères de fin de requête HTTP : %0D%0A

➔ utiliser des URL longues : certains IDS n'analysaient qu'une partie de l'URL

➔ utiliser la syntaxe MS DOS / Windows : remplacer les caractères / par \

exemple : www.monsite.com\cgi\script

➔ découper la requête HTTP sur plusieurs paquets

➔ remplacer les espaces par des tabulations

- Shellcodes polymorphiques : parmi les attaques décrites précédemment, nous avons vu les tentatives de buffer overflow. Celles-ci peuvent être décomposées en trois grandes parties :

  - des instructions pour remplir le buffer. Assez souvent, ce sont des instructions assembleurs NOP, dont le code est 0x90 sur architecture IA32.

  - le shellcode, c'est-à-dire le code qui sera exécuté sur la machine et qui permettra de donner accès à un shell de commandes.

  - une adresse de retour de procédure (qui pointe souvent vers les NOP) qui permettra lors du dépassement de buffer d'exécuter le shellcode.

De nombreux IDS sont capables de détecter les tentatives de buffer overflow. La première méthode est de surveiller la présence importante d'instructions NOP. La deuxième est de détecter la chaîne « bin/sh » qui est souvent présente dans les shellcodes pour Unix. La troisième méthode est d'avoir des signatures complètes de shellcodes répandus sur Internet.

Cependant, il est facilement possible de camoufler une tentative de buffer overflow. Voici quelques techniques simples mais efficaces :

- mettre une autre instruction que des NOP pour remplir le buffer (ex : DAA, AAA, XOR, INC, …)

- écrire les instructions par des équivalences, comme par exemple réécrire une instruction « MOV EAX, 0 » par « XOR EAX,EAX » ou bien encore « SUB EAX, EAX » qui auront toutes les trois pour effet de mettre le registre EAX à 0.

- réaliser un cryptage (XOR) du shellcode et ainsi le rendre polymorphique. Dans ce cas, le décodeur et la clé doivent être présents dans le shellcode pour pouvoir le décrypter lors de son exécution chez la victime.

Nous pouvons remarquer qu'il devient très difficile pour un IDS de détecter un shellcode en utilisant des signatures. Le seul moyen efficace est de détecter le buffer overflow lors de son exécution et l'empêcher au dernier moment de lancer un shellcode. Pour cela, seuls les KIPS (Kernel Intrusion Prevention System) se montrent adaptés puisqu'ils surveillent les appels systèmes.

6) Critères de tests d'un IDS

Lors de la mise en place d'un IDS, il est nécessaire de prendre en considération plusieurs critères qui permettront de choisir au mieux l'IDS.

Tester un IDS avec des scanners de vulnérabilité est une mesure nécessaire pour évaluer un IDS, mais est loin d'être suffisante. D'autres critères doivent être pris en compte :

- Méthodes et capacités de détection : estimer le taux de faux positifs et la qualité d'information fournie par l'IDS.

- Rapidité : tester l'IDS en condition de charge élevée. Il est important de tester cela de manière réaliste, et non pas en utilisant des générateurs de paquets.

- Ouverture : il faut que l'IDS permette de modifier les signatures afin d'éviter certains faux positifs, mais aussi d'ajouter de nouvelles signatures spécifiques à l'environnement.

- Résistance aux techniques d'évasion : utiliser des outils tels que Whisker, Nikto, Babelweb, Fragroute ou Mendax pour observer le comportement de l'IDS.

- Architecture logicielle : pour les grandes entreprises, il est intéressant de pouvoir séparer les fonctions d'administration.

- Exploitabilité des données : il faut disposer d'outils permettant de retrouver et analyser facilement les événements suspects car le volume généré par les IDS est important.

  Afin de centraliser les données, il peut être intéressant de disposer de consoles de reporting ou de tableaux de bords.

- Ergonomie : on retrouve différents types d'interfaces dans les IDS. Tout d'abord, les interfaces graphiques qui sont adaptées aux particuliers ou aux PME. Ensuite, les interfaces de type Web ou encore les interfaces en ligne de commandes réservées aux spécialistes. Dans tous les cas, l'interface doit offrir de nombreuses fonctionnalités.

D'autres critères, comme la réactivité de l'éditeur (mises à jour des signatures, correctifs, …), ou le prix (solution libre ou non) rentrent en jeu. Pour évaluer un IDS, il est intéressant de pondérer chacun de ces critères selon l'importance qu'on leur attribue, et donner une note à l'IDS pour chaque critère.

IV. Mise en œuvre d'IDS

Maintenant que nous connaissons le but, le fonctionnement mais aussi les faiblesses des IDS, nous pouvons découvrir plusieurs solutions logicielles existantes.

1) NIDS / NIPS : Snort

i. Description

Snort est un NIDS / NIPS provenant du monde Open Source. Avec plus de 2 millions de téléchargements, il s'est imposé comme le système de détection d'intrusions le plus utilisé. Sa version commerciale, plus complète en fonctions de monitoring, lui a donné bonne réputation auprès des entreprises.

Snort est capable d'effectuer une analyse du trafic réseau en temps réel et est doté de différentes technologies de détection d'intrusions telles que l'analyse protocolaire et le pattern matching. *Snort* peut détecter de nombreux types d'attaques : buffer overflows, scans de ports furtifs, attaques CGI, sondes SMB, tentatives de fingerprinting de système d'exploitation, ...

Snort est doté d'un langage de règles permettant de décrire le trafic qui doit être accepté ou collecté. De plus, son moteur de détection utilise une architecture modulaire de plugins.

Notons que *Snort* dispose de trois modes de fonctionnement : sniffer de paquets, logger de paquets et système de détection/prévention d'intrusions. Nous ne nous intéresserons qu'à ce dernier mode.

ii. Installation

1. Télécharger les sources sur www.snort.org. Lors de l'écriture de ce document, la dernière version stable était la 2.4.4.
2. Télécharger et installer les bibliothèques nécessaires pour *Snort* :

   - libpcap (http://www.tcpdump.org) : offre des fonctions de sniffer
   - PCRE (http://www.pcre.org) : permet d'utiliser des expressions régulières de type Perl

   La compilation de ces bibliothèques se fait très aisément : *./configure*, *make*, *make install*.

3. Ouvrir un terminal, décompresser ensuite l'archive de *Snort* et se placer dans le répertoire des sources décompressées.
4. Configurer la compilation de *Snort* afin d'activer plusieurs fonctionnalités : *./configure [options]*.

   Deux options nous ont semblé intéressantes :

   - *--with-mysql=DIR :* activer le support de MySQL. Ainsi Snort enregistrera les alertes dans une base de données accessible par d'autres applications (ex : BASE, décrite plus loin). MySQL n'est bien sûr pas l'unique SGBD supporté. PostgreSQL ou Oracle peuvent également être utilisés avec les options *--with-postgresql* et *--with-oracle*.

   - *--enable-flexresp* : activer les réponses flexibles en cas de tentatives de connexion hostile. Pour activer cette option, la bibliothèque libnet (http://www.packetfactory.net/libnet) est nécessaire.

5. Compiler les sources : *make*
6. Installer Snort : *make install* en mode root
7. Pour que *Snort* puisse fonctionner en mode détection/prévention d'intrusions, il est nécessaire de lui fournir des fichiers de règles. Le site de *Snort* propose deux types de règles : les règles officielles et les règles créées par la communauté. Certaines règles officielles ne sont disponibles que pour les utilisateurs enregistrés, tandis que les règles communautaires sont disponibles à tous et mises à jour régulièrement. Il est cependant important de noter que les règles proposées par la communauté n'ont pas été forcément testées par l'équipe officielle de *Snort*.

   Après téléchargement, l'archive des règles doit être décompressée. Il est conseillé de placer le répertoire *rules* dans le dossier de *Snort*. Nous pouvons remarquer que les règles consistent en de simples fichiers textes, et qu'un fichier un peu spécial est présent : *snort.conf*. Ce dernier va nous permettre de configurer *Snort*.

iii. Configuration

Afin de configurer correctement *Snort* pour qu'il puisse fonctionner en mode détection d'intrusions, il faut modifier le fichier *snort.conf*. L'emplacement par défaut de ce fichier doit normalement être */etc/snort.conf*. Cependant, il sera possible de spécifier un autre emplacement lors de l'exécution de *Snort*, à l'aide de l'option -c.

Le fichier de configuration contient de nombreuses options paramétrables, ainsi que des explications pour pouvoir les modifier correctement. Nous n'allons nous intéresser ici qu'à quelques variables :

- La variable *HOME_NET* permet de spécifier quels réseaux ou quelles interfaces seront surveillés par *Snort*. La valeur *any* signale à *Snort* de surveiller tout le trafic.

- Si le réseau à surveiller possède des serveurs DNS, SMTP, FTP, etc , il est possible de spécifier les adresses IP de ces serveurs via les variables *DNS_SERVERS*, *SMTP_SERVERS*, ... Si le réseau ne possède pas un type spécifique de serveur, il est conseillé de commenter (avec le caractère #) la ligne concernée, afin d'optimiser le traitement de Snort. En effet, il est inutile d'analyser du trafic HTTP si aucun serveur Web n'est disponible.

- Certains ports de services peuvent être configurés via des variables telles que *HTTP_PORTS* ou *ORACLE_PORTS*.

- La variable *RULE_PATH* est très importante. Elle permet de spécifier le répertoire où sont stockés les fichiers de règles de *Snort*.

- Les directives *include* permettent d'inclure des fichiers de règles. Ici encore, il est conseillé de n'inclure que les règles nécessaires en fonction des services disponibles sur le réseau.

iv. Exécution

L'exécution de *Snort* se fait en lançant l'exécutable *snort* en mode root et avec différentes options. Voyons les principaux arguments de *Snort* :

- *-A* : générer des alertes. Activé par défaut avec l'option –c
- *-c <emplacement de snort.conf>* : lancer Snort avec des fichiers de règles.
- *-l <répertoire de log>* : spécifier le répertoire où les logs d'alertes seront stockés (défaut : /var/log/snort)
- *-v* : mode verbose. Permet d'afficher les paquets capturés
- *-T* : mode test. Permet de tester la configuration de Snort

Avant de lancer *Snort* en mode NIDS, il est préférable de tester si le programme arrive à récupérer les paquets qui circulent sur le réseau. Pour cela, nous pouvons par exemple lancer *Snort* en simple mode Sniffer : *snort -v.* Si aucun paquet n'est capturé et affiché, il est probable que *Snort* n'écoute pas sur la bonne interface. L'option *-i* permet de spécifier une autre interface.

Lançons maintenant *Snort* en mode NIDS. Pour cela, nous lui précisons l'emplacement du fichier de configuration avec l'option *–c* : snort -c /opt/snort/rules/snort.conf

Toutes les alertes détectées sont ainsi stockées dans le fichier */var/log/snort/alert*. Pour chaque alerte, Snort donne une priorité, une description, les flags des paquets et éventuellement des adresses sur Internet où se trouvent de plus amples informations sur la tentative d'intrusion.

Exemple :

```
[**] [1:1384:8] MISC UPnP malformed advertisement [**]
[Classification: Misc Attack] [Priority: 2]
03/25-17:34:49.251861          192.168.0.1:1900          ->
239.255.255.250:1900
UDP TTL:1 TOS:0x0 ID:37277 IpLen:20 DgmLen:437
Len: 409
[Xref                                                    =>
http://www.microsoft.com/technet/security/bulletin/MS01-
059.mspx][Xref         =>         http://cve.mitre.org/cgi-
bin/cvename.cgi?name=2001-0877][Xref                     =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2001-
0876][Xref => http://www.securityfocus.com/bid/3723]
```

v. Création de nouvelles règles

Bien que le site officiel de Snort propose des règles prêtes à l'emploi et régulièrement mises à jour, il peut être intéressant de créer ses propres règles afin d'adapter au mieux Snort au réseau qu'il doit surveiller et protéger. Par convention, les nouvelles règles personnelles sont à placer dans le fichier *local.rules.*

Une règle *Snort* est composée de deux parties et possède le format suivant : *Header* (*Options)*.

Le format de la partie *Header* est défini de la manière suivante :

```
action protocole adresse1 port1 direction adresse2 port2
```

Le champ action peut prendre les valeurs suivantes :

- *alert* : générer une alerte + logger le paquet
- *log* : logger le paquet
- *pass* : ignorer le paquet
- *activate* : activer une règle dynamique
- *dynamic* : définir une règle dynamique, qui est passive tant qu'elle n'est pas activée par une autre règle
- *drop* : demander à *iptables*[10] de bloquer le paquet, puis le logger
- *reject* : demander à *iptables* de bloquer le paquet, puis le logger, et envoyer une commande TCP RST (reset) ou une réponse ICMP *Host unreachable*
- *sdrop* : demander à *iptables* de bloquer le paquet. Ce dernier n'est pas loggé.

Le champ protocole spécifie le protocole pour lequel la règle s'applique. Les valeurs possibles sont : tcp, udp, icmp ou ip.

Les champs adresse1 et adresse2 indiquent l'adresse IP source et destination du paquet. Le mot clé *any* permet de spécifier une adresse quelconque. Les adresses doivent être numériques, les adresses symboliques ne sont pas acceptées.

Les champs port1 / port2 spécifient les numéros de port utilisés par la source et la destination. Le mot clé *any* permet de spécifier un port quelconque. Des noms de services peuvent être utilisés : tcp, telnet, ... De même des plages de ports peuvent être spécifiées avec le caractère ':'

[10] **Netfilter** est le module qui fournit sous Linux depuis la version 2.4 les fonctions de pare-feu, de partage de connexions internet (NAT) et d'historisation du trafic réseau. **Iptables** est la commande Linux qui permet à un administrateur de configurer **Netfilter** en mode Utilisateur.

Le champ direction spécifie l'orientation du paquet. Cet opérateur peut prendre deux valeurs :

-> : adresse1 vers adresse2
<> : de adresse1 vers adresse2, ou de adresse2 à adresse1

Notons qu'il n'y a pas d'opérateur <- .

La partie *Options* des règles contient différentes options, séparées par un point-virgule, qui vont permettre de préciser des critères de détection. Pour chaque option, le format est *nomOption : valeur1[, valeur2, ...]*

Voici les options importantes :

- *msg* : spécifier le message qui sera affiché dans le log et dans l'alerte
- *reference* : faire référence à un site expliquant l'attaque détectée
- *classtype* : définir la classe de l'attaque (troyen, shellcode, ...)
- *priority* : définir la sévérité de l'attaque
- *content* : spécifier une chaîne de caractères qui doit être présente dans le paquet pour déclencher l'action de la règle
- *rawbytes* : spécifier une suite d'octets qui doit être présente dans le paquet pour déclencher l'action de la règle
- *uricontent* : identique à *content* mais est adapté au format normalisé des URI (ex : hexadécimal accepté)
- *pcre* : utiliser une expression régulière compatible Perl pour spécifier le contenu du paquet
- *ttl* : spécifier la valeur du TTL du paquet
- *flags* : spécifier la présence d'un flag TCP dans le paquet (ex : SYN, FIN, ...)
- *fragbits* : vérifier la présence de certains bits IP (more fragments, don't fragment ou bit réservé)
- *session* : extraire toutes les informations de la session TCP à laquelle le paquet suspect appartient
- *resp* : activer une réponse flexible (flexresp) afin de bloquer l'attaque. Il est ainsi possible d'envoyer une commande TCP ou ICMP précise. Cette option nécessite l'activation du mode *flexresp* lors de la compilation de Snort.
- *limit* : limiter le nombre d'actions pendant un intervalle de temps pour le même événement.

Exemple de règle :

```
alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
ATTACKS /bin/ls command attempt"; uricontent:"/bin/ls";
nocase; classtype:web-application-attack;)
```

Cette règle permet de générer une alerte quand un paquet provient d'un couple (adresse:port) quelconque, est à destination des serveurs HTTP définis dans *snort.conf*, et contient la chaîne « /bin/ls » dans l'URI. Le message de l'alerte sera « WEB-ATTACKS /bin/ls command attempt ». Cette attaque sera classée dans la classe *web-application-attack* (priorité *medium* par défaut).

Il est bien sûr impossible d'être exhaustif ici pour décrire le format des règles *Snort*. Le manuel utilisateur disponible sur le site officiel indique comment utiliser aux mieux le langage des signatures de *Snort*.

vi. La console BASE

Par défaut, les alertes de *Snort* sont enregistrées dans un simple fichier texte. L'analyse de ce fichier n'est pas aisée, même en utilisant des outils de filtre et de tri. C'est pour cette raison qu'il est vivement conseillé d'utiliser des outils de monitoring. Parmi ceux-ci, le plus en

vogue actuellement est *BASE* (Basic Analysis and Security Engine), un projet open-source basé sur *ACID* (Analysis Console for Intrusion Databases). La console *BASE* est une application Web écrite en PHP qui interface la base de données dans laquelle *Snort* stocke ses alertes.

Pour fonctionner, *BASE* a besoin d'un certain nombres de dépendances :

- Un SGBD installé, par exemple MySQL
- Snort compilé avec le support de ce SGBD
- Un serveur HTTP, par exemple Apache
- L'interpréteur PHP avec les supports pour le SGBD choisi, la bibliothèque GD et les sockets.
- La bibliothèque ADODB : http://adodb.sourceforge.net

Nous ne détaillerons pas ici l'installation de chaque dépendance. La documentation livrée avec les sources de *BASE* (disponibles sur http://secureideas.sourceforge.net) fournit les informations nécessaires.

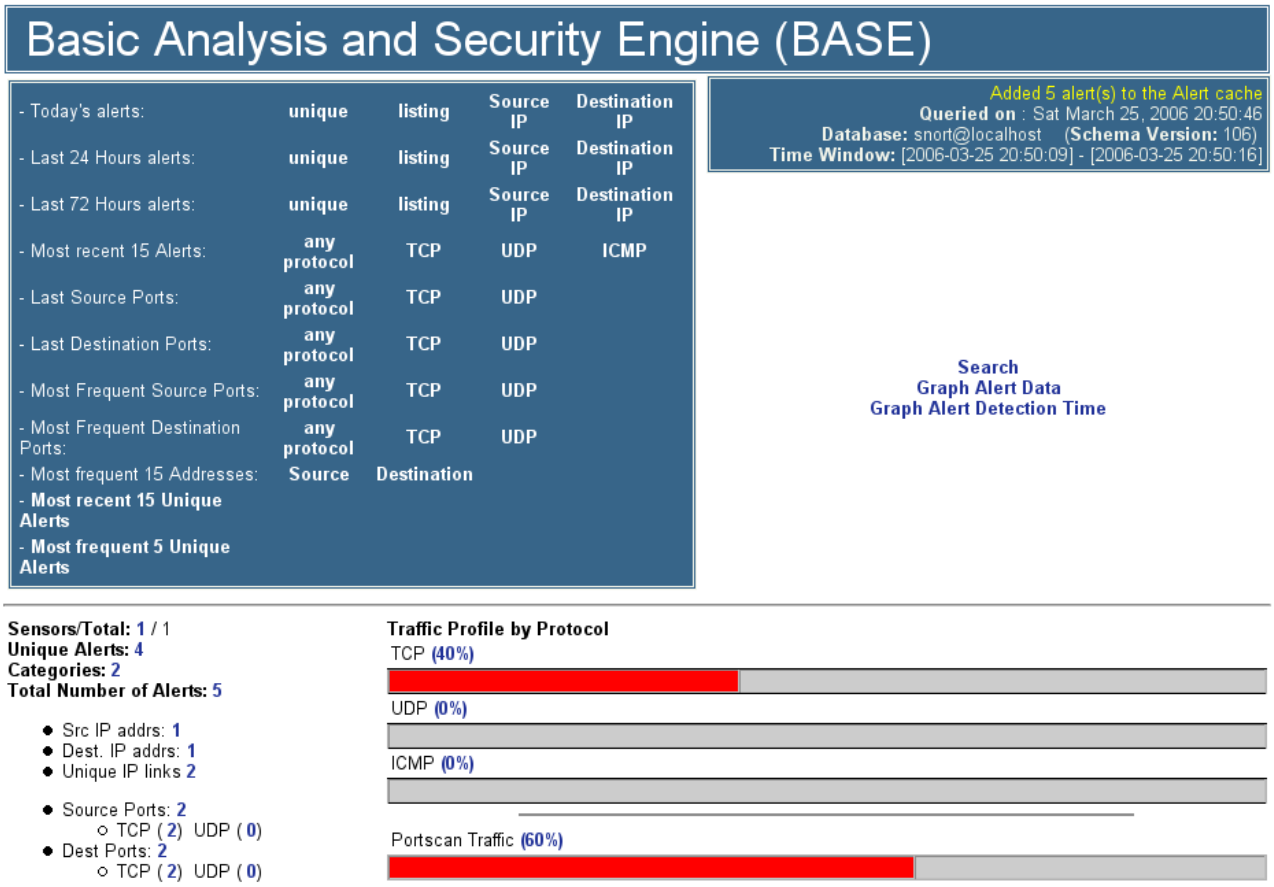
Notons cependant que l'archive des sources de *Snort* dispose également d'un dossier *schemas* contenant le code SQL pour créer la structure de la base de données pour différents SGBD. Le fichier *doc/README.database* donne toutes les indications pour créer le schéma de la base de données.

Afin que *Snort* enregistre les alertes dans la base de données, il ne faut pas oublier de modifier le fichier *snort.conf* et rajouter une ligne *output database* avec les informations pour se connecter à la base de données.

Exemple :

```
output database: log, mysql, user=snortusr password=pwd
dbname=snort host=localhost
```

Après configuration et installation de *BASE* ainsi que de toutes ses dépendances, nous pouvons y accéder avec un navigateur internet. Si tout se passe bien, un écran similaire à l'illustration suivante est obtenu :



2) NIDS : Bro

i. Description

Bro est un NIDS Open Source, développé essentiellement par une équipe du Lawrence Berkeley National Laboratory. *Bro* se base sur un ensemble de règles décrivant des signatures d'attaques ou des activités inhabituelles.

Le comportement après détection de trafic suspect peut être paramétré : simple log, alerte en temps réel à l'administrateur ou exécuter un programme (par exemple pour reconfigurer un routeur).

Initialement, le projet *Bro* a été créé dans un but de recherche pour la détection d'intrusions et l'analyse de trafic réseau. De ce fait, ce NIDS n'est pas destiné aux personnes (physiques ou morales) recherchant une solution prête à l'emploi, mais plutôt à des administrateurs Unix chevronnés, désirant configurer au maximum leur système de détection.

Parmi les fonctionnalités intéressantes, on peut noter :

- Un langage de script propre à *Bro* permettant d'écrire les règles de détection et d'action.
- L'utilisation des expressions régulières pour exprimer les motifs des signatures.
- La possibilité d'exécuter des programmes tiers après détection d'intrusion permet de réaliser de nombreux types d'actions.
- Une compatibilité avec les règles de *Snort*, grâce à un convertisseur nommé *snort2bro.*

- La possibilité d'utiliser GnuPG (Gnu Privacy Gard) pour crypter et signer les rapports d'alertes. Cela permet d'empêcher l'espionnage ou la falsification des logs par un pirate.

- L'envoi d'un rapport périodique des alertes par mail.

L'architecture de *Bro* est définie en 3 couches principales :

- Le module *Packet Capture* : chargé d'écouter le trafic en mode sniffer et de transmettre tous les paquets à la couche supérieure.

- Le module *Event Engine* : classe les flux par protocole, crée une table d'états pour les connexions, contrôle l'intégrité (checksum), réassemble les fragments et analyse les flux. Ce module génère des événements qu'il transmet à la troisième couche.

- Le module *Policy Layer* : utilise les scripts écrits dans le langage de *Bro* pour traiter les événements.

ii. Installation

1. Télécharger les sources sur http://www.bro-ids.org. Lors de l'écriture de ce document, la dernière version stable était la 0.9.
2. Télécharger et installer les bibliothèques nécessaires pour *Bro* :
    - libpcap (http://www.tcpdump.org) : offre des fonctions de sniffer
    - termcap (ftp://ftp.gnu.org/gnu/termcap)

    La compilation de ces bibliothèques se fait très aisément : *./configure*, *make*, *make install.*

3. Ouvrir un terminal, décompresser ensuite l'archive de *Bro* et se placer dans le répertoire des sources décompressées.
4. Générer un makefile adapté au système de destination : *./configure*
5. Compiler les sources : *make*
6. Installer Bro : *make install-brolite* en mode root. Cette commande va non seulement installer Bro mais également créer un exécutable qui va faciliter la configuration de Bro.

iii. Configuration

La configuration générale de *Bro* se fait très simplement en mode interactif à la fin de l'installation. Il suffit dès lors de répondre aux questions qui sont posées, et l'IDS sera configuré automatiquement. Il est possible de reconfigurer *Bro* plus tard en exécutant le script *bro_config*. Notons qu'une configuration manuelle peut être réalisée en éditant le fichier *bro.cfg*.

iv. Exécution

Grâce au script de lancement livré avec *Bro*, le daemon peut être exécuté de manière très simple avec la commande *bro.rc start* (en mode root). Dès le début de l'exécution, les différentes activités sont loggées dans le répertoire *$BROHOME/logs/*. Plusieurs fichiers de log y sont présents :

- alarm.XXX : les attaques détectées
- conn.XXX : un résumé des connexions établies

- ftp.XXX : des informations concernant le trafic FTP suspect
- http.XXX : des informations concernant le trafic HTTP suspect
- info.XXX : des informations générales
- notice.XXX : un suivi des événements lancés

v. Création de nouvelles règles

Tout comme le NIDS *Snort*, il est possible avec *Bro* de créer ses propres règles de détection ou d'adapter les règles existantes en fonction du système utilisé. Pour cela, il faut bien sûr respecter un format très strict mais puissant.

Chaque signature de *Bro* a le format suivant : *signature id { attribute-set }* où *id* représente un identifiant unique (sous forme de chaîne de caractères) de la signature et où *attribute_set* est un ensemble d'attributs qui peut contenir des conditions ou des actions.

Quatre types de conditions existent. Celles-ci vont permettre de définir quand la signature sera reconnue :

- Conditions d'en-tête : elles permettent de spécifier le protocole, les ports et les adresses source et de destination.
- Conditions de contenu : elles sont exprimées sous forme d'expressions régulières pour reconnaître le contenu de certains types de requêtes.
- Conditions de dépendance : elles permettent d'exprimer des dépendances entre les signatures. Par exemple, une signature ne doit être détectée que si une autre signature l'a été.
- Conditions de contexte : elles permettent de rajouter l'exécution de certaines fonctions, écrites avec le langage de script de Bro, qui complémenteront les autres conditions.

Comme nous pouvons le voir, les conditions de détection de signatures sont très évoluées. Cependant, ce n'est pas le cas des actions, c'est-à-dire les opérations à réaliser quand une signature est détectée. Il n'existe pour l'instant qu'une seule action possible : *event*. Celle-ci va permettre de décrire l'éventuelle attaque qui a été détectée.

Enfin, rappelons l'existence de *snort2bro* qui permet de convertir des règles écrites pour *Snort* dans le format de *Bro*. Ainsi, un utilisateur préférant la syntaxe de *Snort* ou utilisant ces deux NIDS, pourra très facilement réutiliser ses règles *Snort* avec Bro.

3) Comparatif Snort & Bro

Nous venons de voir une description des fonctionnalités des deux NIDS les plus utilisés dans le mode Open-Source. Comme nous l'avons remarqué, *Snort* et *Bro* ne partagent pas les mêmes objectifs : l'équipe de *Snort* veut un IDS performant, évolutif et facile à utiliser ; alors que l'équipe *Bro* préfère favoriser l'innovation et la recherche ainsi que profiter de ce projet pour tester des nouvelles méthodes d'analyse.

Voyons cependant un récapitulatif des grandes fonctionnalités qu'offrent ces deux NIDS :

Snort :

+	grande communauté ➔ de nouvelles règles sont très régulièrement proposées par le CERT, SANS ou l'équipe de Snort
+	grande communauté => de nombreux plugins, frontends, consoles de management, ...
+	mise en oeuvre basique rapide
+	projet maintenu par une société : très bonne réactivité et produit adapté aux entreprises
+	de nombreux livres et documentations existants
+	fichiers d'alertes facilement analysables par un humain
+	fichiers d'alertes très complets (header des paquets, lien vers description de l'attaque, ...)
-	fichiers d'alertes difficiles à parser de manière automatisée
-	configuration essentiellement par édition de fichiers texte
-	de nombreuses fonctionnalités sont payantes

Bro :

+	possibilité d'utiliser des règles Snort en les convertissant avec l'utilitaire *snort2bro*
+	forte customisation, qui rend l'IDS très difficile à détecter par un pirate
+	langage de script puissant et adapté à la détection d'intrusion
+	de nombreuses fonctionnalités prévues pour la version 1.0
+	cryptage et signature électronique des rapports
+	rapport périodique résumant les alertes
+	configuration très simple grâce à un script interactif
+	fichiers d'alertes très faciles à parser de manière automatisée
-	projet maintenu par des chercheurs : réactivité moyenne et produit peu adapté aux entreprises
-	fichiers d'alertes difficilement analysables par un humain en format brut
-	peu d'informations dans les rapports d'alertes
-	documentation incomplète empêchant d'utiliser pleinement les nombreuses fonctionnalités
-	aucune GUI actuellement disponible
-	pas de support natif pour enregistrer les alertes dans une base de données

En analysant ces tableaux de fonctionnalités, nous pouvons remarquer que *Bro* est loin d'être un projet amateur car il possède des options très intéressantes. Mais pour devenir un grand NIDS respecté par les entreprises, il devra favoriser le développement de ses méthodes de détection ainsi que simplifier l'accès à l'application par une documentation complétée et des interfaces facilitant la gestion et l'analyse des logs.

Quant à *Snort*, il a fait ses preuves. Mais la version Open-Source est encore réservée à une certaine élite. Bien évidemment, la version commerciale n'a sûrement pas les mêmes faiblesses.

4) NIPS : SnortSam

SnortSam est un plugin Open-Source et multi-plateforme pour *Snort*. Il permet de bloquer automatiquement des adresses IP lorsqu'il détecte une tentative d'intrusion. Le blocage se fait en communiquant avec un firewall matériel (ex : *Cisco Pix*) ou logiciel (ex : *PacketFilter*, *IPtables*, …).

SnortSam est construit autour d'une architecture client / serveur (*Snort* / *SnortSam*) permettant de mettre en place le NIPS de manière distribuée. De plus, pour des raisons de sécurité, toutes les communications réalisées entre *Snort* et l'agent de *SnortSam* sont cryptées à l'aide de l'algorithme *TwoFish*.

Parmi les fonctionnalités intéressantes, on notera la présence d'une *White-List*, c'est-à-dire une liste d'adresses IP qui ne peuvent pas être bloquées. Cela représente une sécurité pour éviter un blocage d'adresses sensibles (routeur, serveur Intranet, …) en cas de spoofing de la part du pirate.

Le plugin *SnortSam* est également doté d'un système de log et de notification par email des événements.

La mise en place d'actions de blocage est très simple. Il suffit de modifier les règles *Snort* pour signaler que la détection de certaines signatures doit provoquer un blocage. Pour cela, le mot clé *fwsam* a été rajouté. Il permet notamment de spécifier une durée de blocage. Cette option de durée peut-être intéressante lors d'un blocage après des tentatives répétées d'authentification avec un mot de passe erroné.

5) NIPS : Snort-Inline

A l'instar de *SnortSam*, *Snort-Inline* est un NIPS basé sur *Snort*. Cependant, il ne s'agit pas d'un plugin mais d'une version modifiée de *Snort*.

Son mode de capture est totalement différent de celui de *Snort* : les paquets ne sont plus capturés grâce à la célèbre bibliothèque *libpcap*, mais via *libipq*, qui permet d'accepter des paquets provenant du firewall *iptables*.

Contrairement à *SnortSam* qui communique avec les firewalls (distants ou non), *Snort-Inline* est un complément au firewall : il doit être installé sur la même machine. Les paquets acceptés seront ensuite transmis sur le réseau.

Snort-Inline est très peu utilisé, surtout depuis l'introduction des actions de blocage directement dans *Snort*. Il est néanmoins très connu en raison de son importance dans le projet *Honeynet* (*www.honeynet.org*).

6) HIDS : OSSEC

i. Description

OSSEC est un HIDS Open Source. Développé initialement dans le but d'analyser quelques fichiers journaux de différents serveurs, il est devenu aujourd'hui bien plus puissant qu'un simple analyseur de logs.

Capable d'analyser différents formats de journalisation tels que ceux d'Apache, syslog, Snort, et intégrant un analyseur d'intégrité système, il est devenu aujourd'hui un logiciel de détection d'intrusions à part entière.

Ses fonctions lui permettent de détecter des anomalies apparues sur le système. Par exemple, de multiples erreurs 404 dans les logs du serveur web Apache, la présence d'un rootkit[11] caché dans un binaire système, ou encore des essais d'envoi de mail par relay smtp.

En réalité, grâce à un système de règles entièrement paramétrable via des fichiers XML, OSSEC est capable de détecter n'importe quelle anomalie. De plus, il est doté d'un système de « réponse active » qui permet à une commande d'être exécutée lors de la détection d'une anomalie.

ii. Installation

L'installation de OSSEC se fait très simplement grâce à un assistant de configuration. Rendez-vous sur http://www.ossec.net pour télécharger l'IDS. Lors de la rédaction de ce document, la dernière version était 0.7p1.

Décompressez l'archive puis lancez l'installeur à l'aide de la commande ./install.sh une fois dans le répertoire décompressé.

Après quelques questions, l'installeur mettra en place OSSEC et il ne restera plus qu'à compléter le fichier de configuration avec les valeurs souhaitées.

iii. Configuration

La configuration se situe dans le fichier ossec.conf du dossier $INSTALL_DIR/etc/. La majorité de ce fichier sera rempli par les paramètres spécifiés lors de l'installation. Ce fichier se divise en plusieurs sections :

- Global : contient les paramètres d'alertes par email.
- Rules : indique les différents fichiers contenant les règles de détection.
- Syscheck : spécifie les répertoires à scanner et les fichiers à ignorer.
- Rootcheck : contient les paramètres de détection des rootkits. Les fichiers contenant les signatures doivent être spécifiés.
- Active-response : active/désactive la réponse active.
- Alerts : spécifie le niveau d'alerte pour avertir par email l'administrateur.
- Localfiles : contient tous les fichiers de logs à vérifier (une entrée par fichier).

iv. Création de nouvelles règles

Grâce à l'externalisation des règles, il est possible de rajouter de nouvelles règles très facilement. Par défaut, les règles concernant un même programme se trouvent toutes dans le même fichier, mais rien n'empêche de créer de nouveaux fichiers.

[11] Un **rootkit** est un programme ou ensemble de programmes permettant à un pirate de maintenir - dans le temps - un accès frauduleux à un système informatique.

Voici un exemple de règle :

```
<group name="syslog,attacks">
...
<rule id="1608" level="14" timeframe="120">
<if_matched_regex>^sshd[\d+]: \.+Corrupted check by bytes
on</if_matched_regex>
<regex>^sshd[\d+]: fatal: Local: crc32 compensation
attack</regex>
<description>SSH CRC-32 Compensation attack</description>
<cve>2001-0144</cve>
<info>http://www.securityfocus.com/bid/2347/info/</info>
</rule>
...
</group>
```

Cette règle est caractérisée par un identifiant unique (chaque règle possède un identifiant propre), un niveau de sécurité allant de 0 à 16 (ici 14) et une fréquence d'apparition (*timeframe*). Cette fréquence d'apparition permet de déclencher une règle uniquement si une règle précédent a déjà été levée. La règle précédente est spécifiée à l'aide de la balise if_matched_regex. Cette dernière option de répétition est bien sur optionnelle.

Plusieurs options de concordance sont possibles. Ici *regex* spécifie qu'il s'agit d'une expression régulière à trouver dans la chaîne. Toutefois, il existe l'option *match* qui spécifie une sous-chaîne à trouver dans la chaîne.

La balise *description* permet de donner une courte description de la règle ; *cve* la date de découverte de la faille et enfin *info* permet de spécifier une information supplémentaire sur la règle, comme par exemple l'url qui référence l'exploit connu.

Plus d'informations sont disponibles dans la documentation sur le site officiel d'OSSEC.

7) HIDS : SamHain

i. Description

De la même façon que OSSEC, Samhain est un système de détection d'intrusions Open Source. Il offre néanmoins des fonctions supplémentaires comme le stockage des alertes d'intrusions dans un système de base de données, le chiffrement des communications client/serveur ainsi qu'une console de visualisation des alertes disponible via un navigateur web.

ii. Installation

L'installation va s'effectuer en plusieurs étapes. Tout d'abord, il faut installer la partie serveur qui va s'occuper de réunir toutes les alertes envoyées par les différents agents. Les sources sont disponibles sur http://la-samhna.de/samhain/.

Exemple d'installation avec le support d'une base de données MySQL :

```
./configure --enable-network=server --enable-xml-log --
with-database=mysql --prefix=/usr/local/yule
make
make install
```

Le serveur (se nommant yule) va ici être installé dans /usr/local/yule. Effectuons à présent l'installation du client :

```
./configure  --prefix=/usr/local/samhain --enable-xml-log -
-enable-network=client \
       --with-data-
file=REQ_FROM_SERVER/var/lib/samhain/samhain_file \
       --with-config-file=REQ_FROM_SERVER/etc/samhainrc \
       --with-logserver=server.yourdomain.com
make
```

La variable *REQ_FROM_SERVER* indique que ce fichier doit être récupérer du serveur. *Logserver* spécifie l'adresse où se trouve le serveur regroupant toutes les alertes, c'est à dire l'adresse à laquelle le client samhain enverra les alertes.

Cette installation n'est toutefois pas terminée, puisque avant d'effectuer la dernière phase, il faut configurer le client pour qu'il puisse s'authentifier au niveau du serveur.

Générons une clé aléatoire sur le serveur :

```
yule -P $SHPW | sed s%HOSTNAME%client.yourdomain.com% >>
/etc/yulerc
```

Indiquons cette clé à notre client :

```
./samhain_setpwd samhain new $SHPW
```

Cette opération a pour but de créer un nouveau binaire se nommant samhain_new, qui nous permettra de nous connecter avec le serveur. Terminons l'installation avec ces lignes :

```
mv samhain.new samhain
make install && make install-boot
samhain -t init
```

iii. Configuration

La configuration par défaut offre un environnement de détection quasi opérationnel. Il suffit de modifier la variable *ExportSeverity* afin de spécifier le niveau de détection considéré comme « alerte ».

Dans le fichier /etc/samhainrc, modifiez ainsi :

```
[Log]
ExportSeverity=warn
```

8) HIDS : RkHunter

i. Description

Rkhunter est un détecteur de rootkits. Il permet de détecter la présence d'un rootkit dissimulé dans un binaire système. Il offre également la possibilité de scanner des fichiers de configuration afin de détecter la présence d'anomalies de configuration (utilisateur avec UID 0 : root), ou encore la présence de ports ouverts suspects (utilisés par certains rootkits).

ii. Installation

L'installation de *rkhunter* se fait automatiquement. Récupérez les sources sur http://www.rootkit.nl/ puis décompressez l'archive. L'installation s'effectue à l'aide de l'assistant installer.sh.

Aucune configuration supplémentaire n'est à effectuer. Il suffit de lancer le binaire pour lancer un scan. Cette ligne de commande permettra d'effectuer un scan non interactif :

```
/usr/local/bin/rkhunter -c -sk
```

Voici un exemple de résultat :

```
Networking
* Check: frequently used backdoors
  Port 2001: Scalper Rootkit                [ OK ]
  Port 2006: CB Rootkit                     [ OK ]
  Port 2128: MRK                            [ OK ]
  Port 14856: Optic Kit (Tux)               [ OK ]
  Port 47107: T0rn Rootkit                  [ OK ]
  Port 60922: zaRwT.KiT                     [ OK ]

* Interfaces
     Scanning for promiscuous interfaces    [ OK ]


Security advisories
* Check: Groups and Accounts
   Searching for /etc/passwd...             [ Found ]
   Checking users with UID '0' (root)...    [ OK ]

* Check: SSH
   Searching for sshd_config...

* Check: Events and Logging
   Search for syslog configuration...       [ OK ]
   Checking for running syslog slave...     [ OK ]
   Checking for logging to remote system... [ OK (no remote
logging) ]

------------------ Scan results --------------------
MD5
MD5 compared: 0
Incorrect MD5 checksums: 0

File scan
Scanned files: 342
Possible infected files: 0

Application scan
Vulnerable applications: 0

Scanning took 62 seconds
```

9) Comparatif HIDS

Le tableau suivant montre les différences entre les trois HIDS que nous venons de présenter. Ce tableau les compare sur les principaux critères exigés d'un HIDS en environnement de production.

Produit / Fonctionnalités	OSSEC	Samhain	Rkhunter
Architecture	C-S / local	C-S / local	local
Rootkits	Oui	Oui	Oui
Fichiers logs	Contenu	Contenu + Taille	X
Fichiers configuration	Perm., Taille, Prop., md5	Perm., Taille, Prop., md5	X
Interface	X	Web	X
Alertes	Log, Mail	Log, Mail, Syslog	Log, Mail

Tirons quelques conclusions de ce tableau comparatif. D'une part, nous remarquons immédiatement la déficience du logiciel Rkhunter. Ceci est normal, Rkhunter est uniquement un analyseur système pour vérifier la présence de rootkits. Mais nous avons jugé important de le placer dans ce comparatif de part ses fonctionnalités d'alerte qui se rapprochent, voire équivalent, celles d'autres HIDS.

Les deux systèmes Ossec et Samhain sont fort proches. Ils possèdent tous les deux la possibilité d'être mis en place de manière locale ou bien via une architecture client/serveur. Chacun d'entre eux analyse le système pour vérifier la non présence de rootkits, inspecte les fichiers de logs pour y détecter des activités anormales, et permet d'avertir l'administrateur via des emails ou en journalisant les alertes.

En dehors de cela, nous remarquerons la maigre supériorité de Samhain sur OSSEC. Toutefois, cette supériorité est à tempérer. En effet, durant nos tests, l'installation de Samhain nous a posé quelques problèmes, et la configuration n'a pas été des plus simples. A l'inverse, le système OSSEC est doté d'un installeur qui prend les choses en main et nous demande uniquement les paramètres de configuration. Ainsi, malgré un nombre de fonctionnalités important, Samhain n'est peut-être pas aussi mature qu'il le laisse paraître.

10) IDS Hybride : Prelude

Comme nous l'avons précédemment constaté, il existe peu d'outils permettant d'unifier l'utilisation conjointe de différents IDS. Prelude est né de cette observation, il se veut être un framework[12] de centralisation et d'unification des événements permettant d'accélérer l'analyse des différentes alertes.

Prelude est un IDS Hybride, c'est-à-dire qu'il combine les événements détectés par toute sorte d'applications de sécurité : NIDS, analyseur de logs (HIDS), anti-virus, etc… Dans le but de réaliser cette tâche, Prelude s'appuie sur le standard IDMEF (Intrusion Detection Message Exchange Format), dont nous avons déjà parlé dans la partie « Normalisation ».

Parmi les applications de sécurité supportées par Prelude, on retrouve le NIDS Snort, le scanner de vulnérabilités Nessus, le HIDS Samhain, et plus de 30 analyseurs de logs.

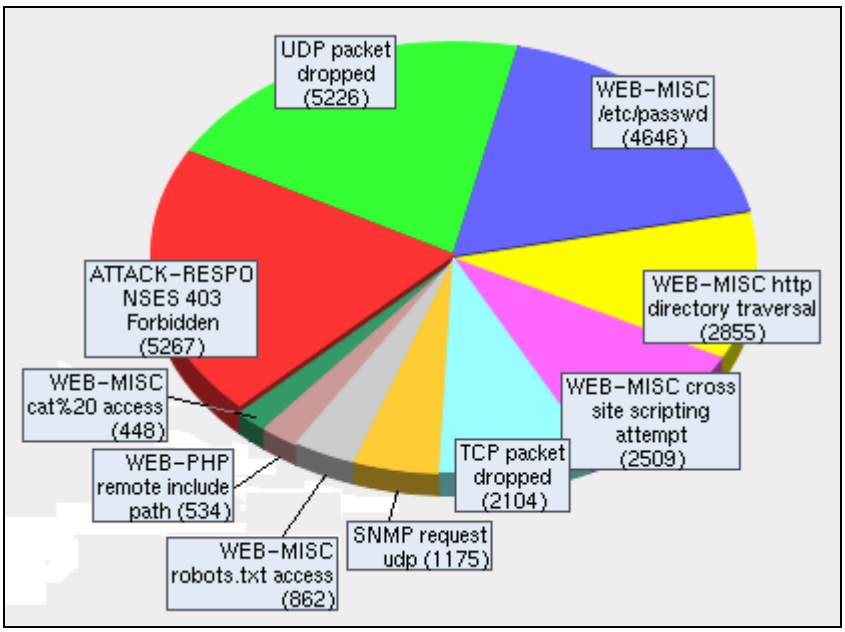
[12] Un **Framework**, ou cadre d'applications, est un ensemble de bibliothèques et d'outils permettant de faciliter l'utilisation groupée d'applications.

Le framework se compose d'une bibliothèque de génération de messages IDMEF, d'un gestionnaire d'événements, d'un analyseur de logs et d'une console de visualisation des alertes.

L'interface de visualisation des alertes est une application web développée en tant que script CGI. Elle permet de mettre en évidence les alertes importantes, comme le montre l'illustration ci-dessous.

Classification	Source	Target	Sensor	Time
WEB-MISC robots.txt access	84.104.217.36:45295	194.246.101.65:80	snort (on awale.prelude-ids.org)	13:01:07
2 x TCP packet dropped (failed)	216.184.192.227:2927	194.246.101.67:139 interface: eth1	prelude-lml/netfilter (on awale.prelude-ids.org)	12:54:56 - 12:54:53
Mail server suspicious access (failed)	62.103.252.123	194.246.101.65 (awale.prelude-ids.org)	prelude-lml/Postfix (on awale.prelude-ids.org)	12:53:01
(14910/15088 alerts not shown... expand) 6 x **(spo_bo) Back Orifice Traffic detected** 2 x **TFTP GET passwd** 8 x **WEB-CGI /cgi-bin/ access** 40 x **WEB-CGI a1stats a1disp3.cgi directory traversal attempt** 1 x **WEB-CGI agora.cgi attempt** 20 x **WEB-CGI AltaVista Intranet Search directory traversal attempt** 20 x **WEB-CGI Amaya templates sendtemp.pl directory traversal attempt** 41 x **WEB-CGI anaconda directory transversal attempt** 20 x **WEB-CGI Armada Style Master Index directory traversal** 20 x **WEB-CGI auktion.cgi directory traversal attempt**	82.226.58.44	194.246.101.65:22 (awale.prelude-ids.org)	prelude-lml/sshd (on awale.prelude-ids.org)	12:47:22 - 12:43:21

De plus, cette console permet de générer des graphiques statistiques rassemblant des informations importantes sur les cibles visées et permettant de voir rapidement quelles sont les alertes les plus fréquentes. L'illustration suivante est un exemple des possibilités de la console de *Prelude*.



11) KIDS : LIDS

Le système de détection d'intrusions *LIDS* est un patch à appliquer au noyau Linux. Celui-ci permet d'augmenter la sécurité du noyau en implémentant un contrôle d'accès à chaque ressource système. Une fois activé, tout accès fichier, toute opération d'administration réseau, tout usage mémoire et toute lecture/écriture sur disque deviennent impossibles même pour l'utilisateur root.

Il est possible de spécifier la liste des fichiers accessibles par chaque programme indépendamment. Ce système offre une sécurité optimale, mais peut s'avérer compliqué à mettre en place et à configurer. Mal configuré, ce système peut paralyser une machine et la rendre totalement hors d'usage.

LIDS supporte les noyaux 2.6 et 2.4 à l'heure actuelle. Il est distribué gratuitement sous licence GPL.

V. Tests de détection d'attaques

Afin de tester les performances de détection des NIDS présentés dans ce rapport, nous avons réalisé quelques simulations d'attaques. Celles-ci sont cependant très basiques, et les résultats présentés ici ne reflètent pas la capacité de détection d'intrusions face aux techniques utilisées par des professionnels.

1) Exploit phpBB

Présentation :

Cette attaque utilise une faille présente dans certaines versions du très célèbre moteur de forum *phpBB*. Pour exploiter cette faille, nous avons utilisé le framework *Metasploit*.

L'exploit consiste à utiliser le paramètre *highlight* de la page *viewtopic.php* dont le contenu n'est pas vérifié correctement, permettant ainsi d'injecter du code arbitraire.

Résultats du test :

Snort	L'attaque est **détectée**. Cependant, l'alerte générée ne concerne pas exactement l'attaque mais l'accès au fichier *viewtopic.php*. Or ce fichier est l'un des principaux de *phpBB* ! La signature de cette alarme va générer beaucoup trop de faux positifs, il est conseillé de l'enlever.
Bro	L'attaque n'est **pas détectée** par défaut, sauf si on convertit la même signature que possède Snort, concernant l'accès au fichier *viewtopic.php*.

Conclusion :

Aucun des deux IDS ne s'est montré capable de détecter correctement cet exploit. En effet, les signatures par défaut de Snort ne détectent pas l'attaque en elle-même mais l'accès au fichier concerné. Néanmoins, il est impensable de générer une alerte pour chaque accès au fichier *viewtopic.php* sur un serveur disposant d'un forum *phpBB*.

2) Scan TCP SYN

Présentation :

Il s'agit d'un scan de ports TCP qui consiste à ne pas établir complètement de connexion. Lorsqu'un port ouvert est détecté, la poignée de main est réinitialisée.

L'utilitaire *nmap* peut réaliser un tel scan grâce à l'argument *–sS*.

Résultats du test :

Snort	Le scan est **détecté**. L'alerte générée signale également les ports qui étaient ouverts lors du scan.
Bro	Le scan est **détecté**.

Conclusion :

Ce genre de scan passait inaperçu il y a quelques années mais il est facilement détectable de nos jours.

3) Scan TCP Connect

Présentation :

Il s'agit d'un scan de ports TCP qui détecte les ports ouverts en établissant une connexion complète à ceux-ci.

L'utilitaire *nmap* peut réaliser un tel scan grâce à l'argument *–sT*.

Résultats du test :

Snort	Le scan est **détecté**. L'alerte générée signale également les ports qui étaient ouverts lors du scan.
Bro	Le scan est **détecté**.

Conclusion :

Cette méthode de scan est la plus facile à mettre en œuvre, mais c'est également la moins discrète.

4) Scan Null

Présentation :

Il s'agit d'un scan de ports TCP qui détecte les ports ouverts en envoyant un paquet TCP avec tous les champs du header à 0. Si le port est fermé, le serveur répond normalement par une commande *RST*. Si le port est ouvert, le serveur ne répond pas.

L'utilitaire *nmap* peut réaliser un tel scan grâce à l'argument *–sN*.

Résultats du test :

Snort	Le scan est **détecté**.
Bro	Le scan n'est **pas détecté**.

Conclusion :

Ici, Snort se montre plus performant que Bro, qui ne voit rien d'anormal lors du scan.

5) Remarques sur les scans

Les résultats des scans présents ci-dessus concernent les scans par défaut. Il est important de noter que tout scan passe inaperçu si chaque test de port est suffisamment espacé. L'option *Paranoid Trottling* de *nmap* permet notamment d'attendre 5 minutes entre chaque test de port. Le scan devient dès lors très lent (plusieurs heures pour un scan complet) mais est difficilement détectable.

6) Exploit : Overflow HTTP d'Oracle 9i (win32)

Présentation :

La version 9i du célèbre SGBD Oracle est victime d'une faille dans son moteur XML, appelé XDB. Nous avons testé un exploit tirant profit cette faille avec le framework Metasploit. L'exploit consiste à provoquer un stack overflow lors de l'identification HTTP au serveur de base de données.

Résultats du test :

Snort	L'attaque n'est **pas détectée**. Un processus peut être lancé en mode SYSTEM sans que la moindre anomalie ne soit remarquée !
Bro	Pas de version Windows de ce NIDS

Conclusion :

Par défaut, Snort n'est pas capable de détecter l'attaque. Cependant, un bon administrateur réseaux pourra créer facilement une règle générant une alerte lors de cette attaque. Pour cela, il devra analyser entièrement l'exploit, disponible sur Internet.

7) Exploit : Overflow FTP d'Oracle 9i (win32)

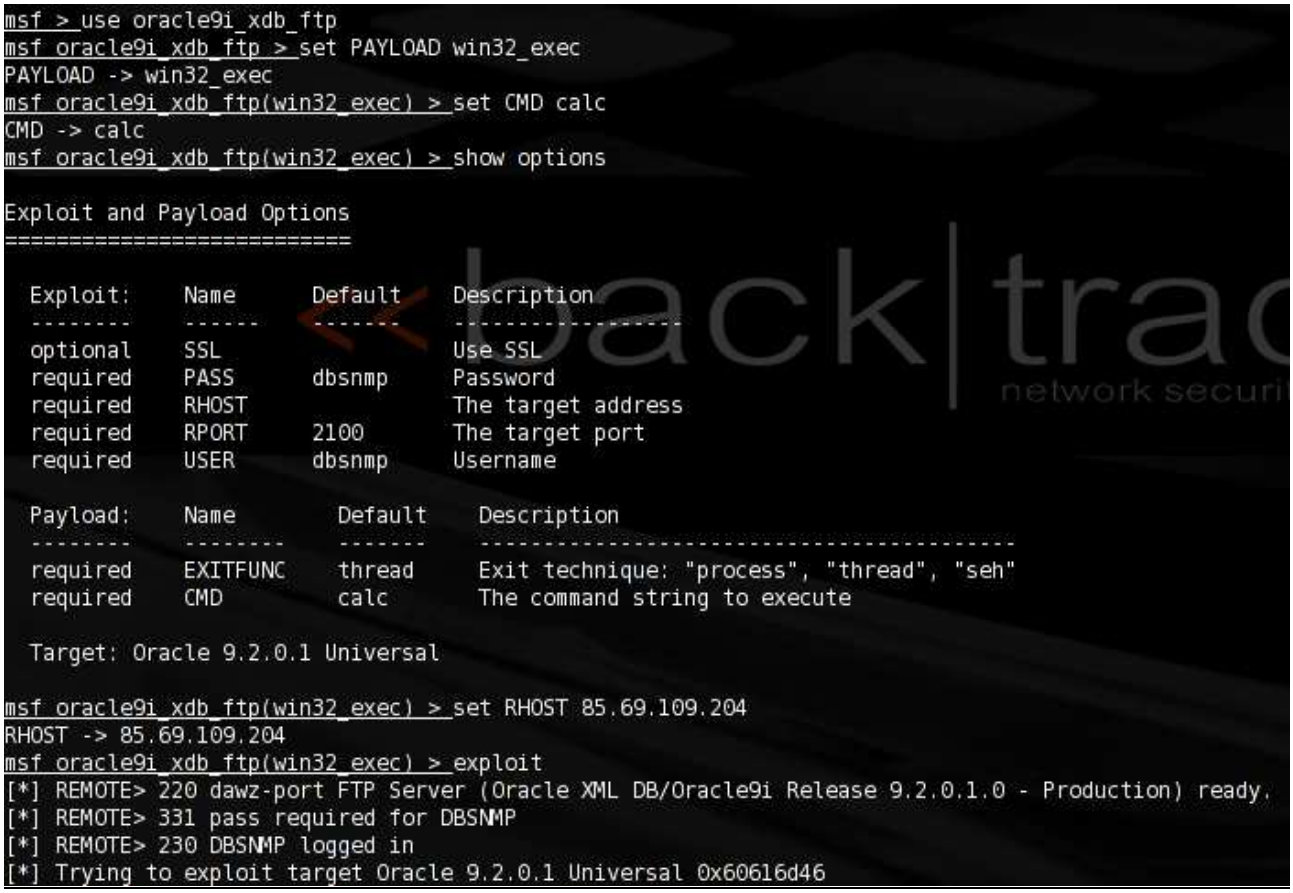
Présentation :

Il s'agit d'une faille identique à la précédente, à l'exception que sa mise en œuvre est réalisée en utilisant le protocole FTP. Ici encore, nous avons utilisé le framework Metasploit pour tester cette attaque.

Résultats du test :

Snort	L'attaque est **détectée**.
Bro	Pas de version Windows de ce NIDS

Screenshot de l'attaque :



Conclusion :

Snort dispose par défaut de la signature pour cette attaque, permettant de la détecter très facilement.

8) Remarques sur les exploits Oracle 9i

Comme nous l'avons vu, une version par défaut de Snort ne détecte pas toujours les attaques contre le moteur XML d'Oracle 9i. Pour se protéger de manière efficace, l'entreprise Oracle préconise tout simplement de désactiver les services HTTP et FTP.

VI. Algorithmique : le Pattern Matching

Bien que la détection d'intrusions soit plutôt liée aux réseaux et aux systèmes d'exploitation, l'algorithmique reste très importante dans ce domaine. Il serait impossible d'être exhaustif pour les algorithmes mis en œuvre dans les IDS. De ce fait, nous avons décidé de nous limiter aux techniques de pattern matching.

1) Introduction

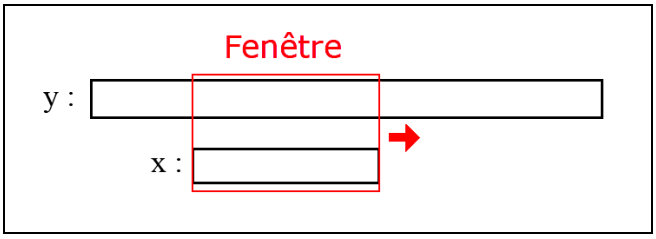
La recherche de motif (*pattern matching*) est un sujet important dans le domaine du traitement de données. Son but est de localiser une occurrence d'un mot (= le *motif*) dans une chaîne de caractères (= le *texte*).

Le pattern matching joue un rôle crucial dans les NIDS qui utilisent des signatures pour détecter les attaques ou intrusions. Un bon NIDS doit pouvoir analyser les paquets qu'il reçoit en temps réel. Avec les vitesses élevées des réseaux actuels, il est évident que les algorithmes de pattern matching ne doivent pas être négligés car il en va de la sécurité des systèmes surveillés.

Plusieurs algorithmes de recherche de motif vont être expliqués dans ce chapitre. Pour la suite, les notations suivantes seront utilisées :

- **x = $x_0x_1...x_{m-1}$** : le motif à rechercher, de longueur **m**
- **y = $y_0y_1...y_{n-1}$**: le texte sur lequel aura lieu la recherche, de longueur **n**
- **x[i]** et **y[i]** : le $i^{ème}$ caractère de x ou y, équivalent à x_i et y_i

La plupart des algorithmes de pattern matching utilisent une fenêtre glissante de taille *m*. Cette fenêtre représente la position de l'éventuelle occurrence du motif. En d'autres termes, elle englobe les caractères du texte qui vont être comparés avec le motif pour la tentative en cours. Cette fenêtre est décalée à chaque non correspondance (*mismatch*) et la procédure de comparaison reprend au début de la fenêtre.



Deux catégories de pattern matching peuvent être distinguées :

- avec motifs fixes et texte variant : c'est le cas des NIDS
- avec texte fixe et motifs variants : correspond à une recherche dans un dictionnaire

Dans tous les cas, l'efficacité d'un algorithme de pattern matching dépend du nombre de comparaisons de caractères nécessaires pour rechercher le motif.

Lors du choix et de l'implémentation d'un algorithme de pattern matching, plusieurs considérations doivent être prises en compte :

a) **Le type de pattern matching** : recherche multi ou simple motif, motifs statiques ou dynamiques, …

b) **Sensibilité à la casse** : dans le cas d'une recherche insensible à la casse, les caractères du motif et du texte seront par exemple convertis en majuscules.

c) **La taille des motifs** : certains algorithmes ont des performances réduites pour les motifs de grande taille.

d) **La taille de l'alphabet** : la plupart des algorithmes fonctionnent facilement si les caractères sont stockés sur un octet, mais dans le cas du codage Unicode, certaines adaptations devront être apportées.

e) **Le risque d'attaques de l'algorithme** : des pirates peuvent éventuellement utiliser les propriétés et le comportement de l'algorithme pour réduire les performances du programme. Il est donc nécessaire, surtout dans le cas des IDS, d'évaluer les conséquences de telles attaques.

f) **La fréquence des recherches et la taille des textes de recherche** : ces deux aspects peuvent influencer les performances, et l'algorithme devra être adapté aux types de recherches réalisés.

2) Algorithme naïf (Brute Force Algorithm)

Cet algorithme consiste à vérifier pour chaque position dans le texte si une occurrence du motif commence là ou non. Il compare donc le motif *x* à chaque facteur du texte *y* de longueur m. Si une occurrence est rencontrée, on le signale ; sinon, on recommence avec le facteur suivant de *y* en décalant la fenêtre d'exactement une position vers la droite.

```
Procédure Recherche_Naïve (x, y):
   i := 0
   j := 0

   tantque i < m et j < n faire
      si y[j] = x[i] alors // Caractère identique :
         i := i +1 // on continue dans la même fenêtre
         j := j+1 //
      sinon
         j := j - i +1 // Décalage de la fenêtre
         i := 0 // Le motif est reparcouru
      finsi
   fintantque

   si i = m alors
      Occurrence de x trouvée à la position j – m
   sinon
      Pas d'occurrence
   finsi
```

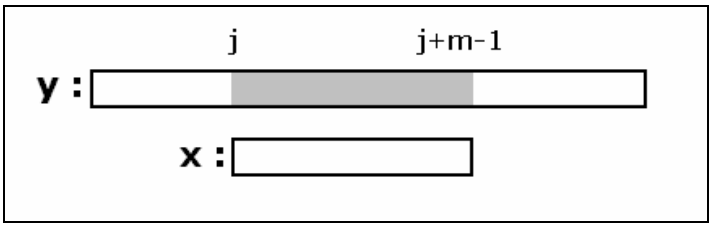
Dans le pire des cas, cet algorithme fait $(n - m + 1)\ m$ comparaisons, ce qui correspond à une complexité temps de O(mn). Ce cas correspond par exemple à la recherche de $a^{m-1}b$ dans a^n puisque à chaque tentative, l'algorithme parcourt tout le motif, pour se rendre compte que le dernier caractère (b) n'est pas identique.

3) L'algorithme de Morris - Pratt

L'algorithme naïf a un gros inconvénient : en cas d'échec, il recommence entièrement la comparaison du motif *x* avec le facteur suivant de *y*, sans exploiter l'information contenue dans la réussite partielle de la tentative. L'algorithme de Morris-Pratt reprend l'algorithme naïf tout en profitant de l'analyse du texte pour collecter des informations.

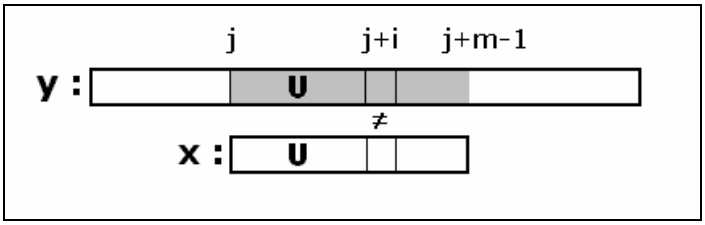
Grâce à un prétraitement sur le motif, il est possible d'obtenir un gain de temps considérable en évitant de répéter des comparaisons.

Considérons une tentative de recherche du motif *x* dans le texte *y*, commençant à la position *j* dans y. Cela correspond donc au cas où la fenêtre est positionnée sur *y[j .. j + m - 1].*



Supposons que la première non correspondance ait lieu lors de la comparaison entre x[i] et y[j+i], avec 0<i<m. Dès lors :

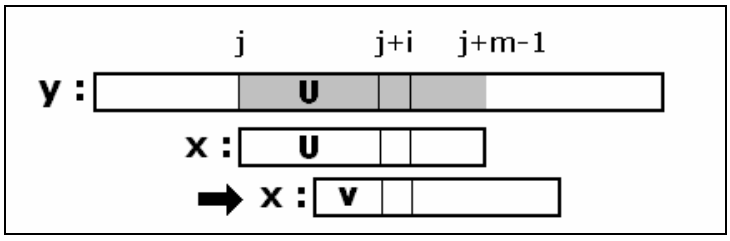
- x[0..i-1] = y[j..j+i-1] : les i-1 caractères précédemment comparés sont identiques. Notons *u* cette sous chaîne.
- x[i] ≠ y[j+i]



Lorsqu'un décalage de la fenêtre de comparaison a lieu, il ne faut pas oublier qu'un préfixe *v* du motif peut correspondre à un suffixe de la portion *u* du texte. Ce préfixe *v* est appelé un bord de *u*.

Définition :

Soit *u* un mot quelconque non vide, un bord de *u* est un mot distinct de *u* qui est à la fois préfixe et suffixe de *u*.



Notation :

Soit mpNext[i] la longueur du plus long bord de x[0..i-1], pour 0 < i ≤ m.

mpNext[0] = -1
mpNext[i] = | Bord maximal (x[0..i-1]) |

Après un décalage, les comparaisons peuvent reprendre entre x[mpNext[i]] et y[j+i] sans risquer d'oublier une occurrence de x dans y, et tout en évitant de faire un retour en arrière dans le texte.

La table mpNext peut être construite en espace et temps O(m). Cette table doit être construite avant la phase de recherche.

```
Procédure PrétraitementMP (x, m, mpNext[] )
  i := 0
  j := mpNext[0] := -1

  tantque (i < m) faire // Parcours du motif

    tantque ((j > -1) et (x[i] ≠ x[j])) faire
      j := mpNext[j];
    fintantque

    j := j + 1
    i := i + 1
    mpNext[i] = j

  fintantque
```

Voyons un exemple de la table des longueurs des bords maximaux pour le motif "ACABACA".

i	0	1	2	3	4	5	6	7
x[i]	A	C	A	B	A	C	A	
mpNext[i]	-1	0	0	1	0	1	2	3

Nous remarquons que mpNext[7] = 3, c'est-à-dire que le bord maximal de x[0..6], et donc du motif entier, est de longueur 3. En effet, ce bord est "ACA" qui est préfixe et suffixe de ACABACA.

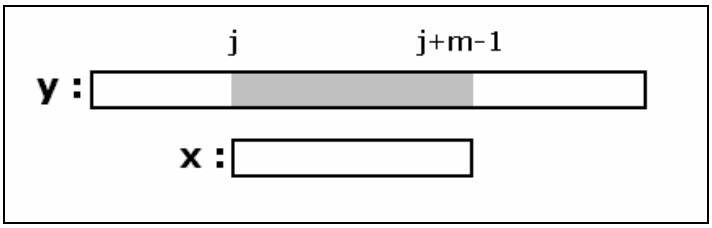
Après la création des longueurs des bords, la phase de recherche peut être faite en temps O(m+n). Au pire des cas, l'algorithme de Morris-Pratt fait 2n-1 comparaisons.

```
Procédure RechercheMP (x, m, y, n)

  PrétraitementMP (x, m, mpNext)
  i := j := 0 // Initialisation des indices

  tantque (j <= n) faire // Parcours du texte

    tantque ( (i > -1) et (x[i] ≠ y[i]) ) faire
      i := mpNext[i] // Décalage en fin de bord
    fintantque

    i := i + 1
    j := j + 1

    si (i ≥ m) alors
      Occurrence de x trouvée à la position j – i
      i := mpNext[i]
    finsi

  fintantque
```

4) L'algorithme de Knuth - Morris - Pratt

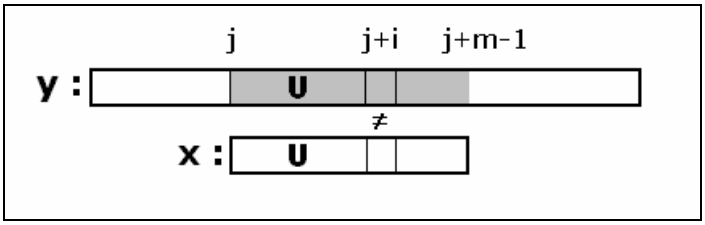
L'algorithme de Knuth-Morris-Pratt est une version optimisée de l'algorithme de Morris-Pratt. En effet, l'algorithme Morris-Pratt peut être optimisé en améliorant la longueur des décalages.

Supposons une recherche du motif *x* à partir de la position *j* dans le texte *y*, c'est-à-dire que la fenêtre est positionnée sur y[j..j+m+1].

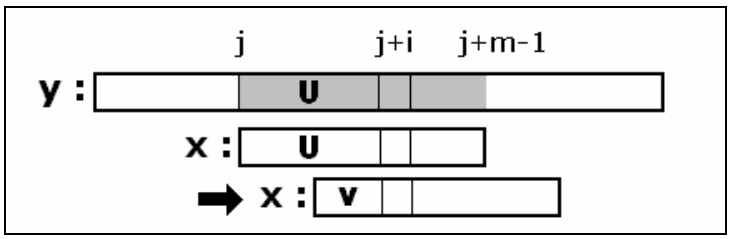


Et supposons que la première différence ait lieu lors de la comparaison de x[i] avec y[j+i], avec 0 < i < m. Dès lors :

- x[0..i-1] = y[j..j+i-1] : les i-1 caractères précédemment comparés sont identiques. Notons *u* cette sous chaîne.
- x[i] ≠ y[j+i]



Lorsqu'un décalage de la fenêtre de comparaison a lieu, il ne faut pas oublier qu'un préfixe *v* du motif peut correspondre à un suffixe de la portion *u* du texte.



De plus, pour éviter une autre non correspondance immédiate, le premier caractère après le préfixe *v* doit être différent de l'ancien x[i] puisque nous savons déjà que y[j+i] ≠ x[i].

Un tel préfixe *v* est appelé *bord disjoint* (tagged border) de *u*.

Notation :

Soit kmpNext[i] la longueur du plus long bord de x[0..i-1], pour 0 < i ≤ m, suivi d'un caractère différent de x[i] ; ou -1 si aucun bord disjoint n'existe.

mpNext[0] = -1
mpNext[i] = | Bord maximal disjoint (x[0..i-1]) |

Après un décalage, les comparaisons peuvent reprendre entre x[kmpNext[i]] et y[j+i] sans rater d'occurrence de *x* dans *y*, et tout en évitant un retour en arrière dans le texte.

La table *kmpNext* peut être calculée en temps et espace O(m). Ce calcul est réalisé avant la phase de recherche.

```
Procédure PrétraitementKMP (x, m, kmpNext)

   i := 0
   j := kmpNext[0] := -1

   tantque (i < m) faire // Parcours du motif

      tantque ( (j > -1) et (x[i] ≠ x[j]) ) faire
         j := kmpNext[j]
```

```
    fintantque

    i := i + 1
    j := j + 1

    si (x[i] = x[j]) alors // Même caractère : bord non
disjoint
      kmpNext[i] := kmpNext[j]
    sinon
      jmpNext[i] := j // Position du bord disjoint
    fsi

  fintantque
```

Après le calcul de la table *kmpNext*, la phase de recherche peut être réalisée en temps O(m+n), avec au plus 2n-1 comparaisons.

```
Procédure KMP (x, m, y, n)

  PrétraitementKMP (x, m, kmpNext)
  i := j := 0

  tantque (j < n) faire // Parcours du texte

    tantque ( (i > -1) et (x[i] != y[j]) ) faire
      i := kmpNext[i]
    fintantque

    i := i + 1
    j := j + 1

    si (i > m) alors
      Occurrence de x trouvée à la position j – i
      i := kmpNext[i]
    finsi
  fintantque
```

Nous remarquons que l'algorithme de Knuth-Morris-Pratt a exactement la même complexité pour le cas le plus défavorable que l'algorithme de Morris-Pratt. Cependant, il y a une différence importante au niveau du délai, c'est-à-dire le nombre de comparaisons faites sur un même caractère dans le pire des cas.

Le délai pour l'algorithme de Morris-Pratt est borné par m, alors qu'il est borné par $\log_\phi(m)$ où ϕ est le nombre d'or : $(1 + 5^{1/2}) / 2$.

5) Algorithme de Boyer - Moore

L'algorithme Boyer-Moore est un algorithme qui utilise également le principe de la fenêtre de décalage. Mais contrairement à l'algorithme naïf et l'algorithme Knuth-Morris-Pratt, les comparaisons sont faites de la droite vers la gauche. Cette modification apparemment anodine est très rentable : en pratique, l'algorithme de Boyer-Moore est le plus rapide des algorithmes connus ! Il est notamment utilisé dans des éditeurs de texte pour les commandes "rechercher" et "remplacer".

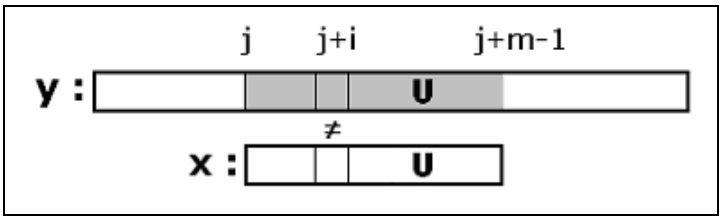
En cas de non correspondance (mismatch), il utilise deux fonctions pour choisir le décalage vers la droite de la fenêtre. Ces deux fonctions sont appelées décalage du bon suffixe et décalage du mauvais caractère.

Supposons qu'une non correspondance ait lieu lors de la comparaison de x[i] avec y[j+i]. Dès lors :

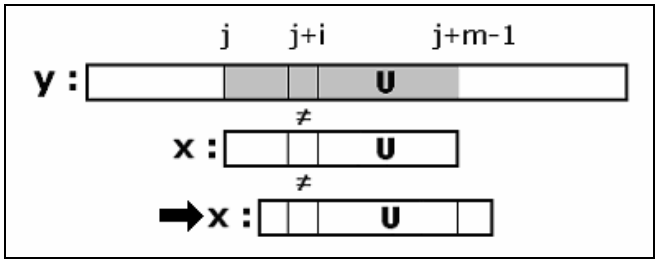
- x[i+1 .. m-1] = y [i+j+1 .. j+m-1], notons *u* cette portion

  ➔ pour rappel : le parcours des comparaisons se fait de gauche à droite, donc de j+m-1 à j.

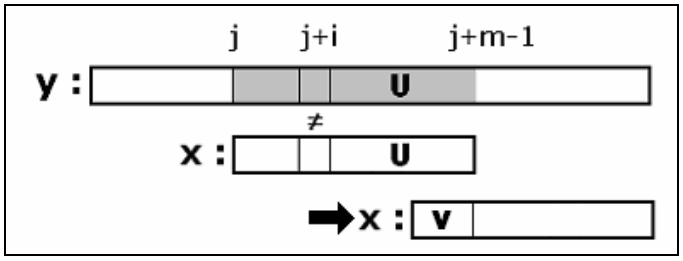
- x[i] ≠ y[j+i]



La fonction de décalage *du bon suffixe* consiste à aligner le segment *u* avec son occurrence la plus à droite dans *x*, qui est précédée par un caractère différent de x[i].

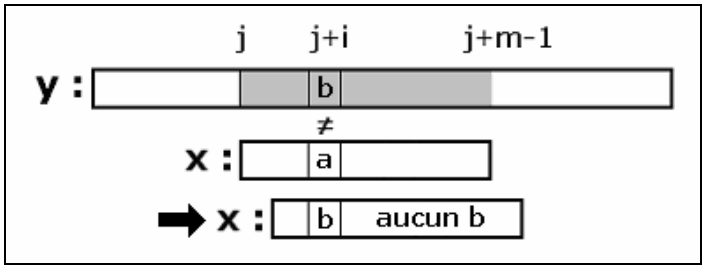


S'il n'y a pas de tel segment, le décalage consiste à aligner le plus long suffixe *v* de y[j+i+1 .. j+m+1] avec un préfixe de x qui correspond.

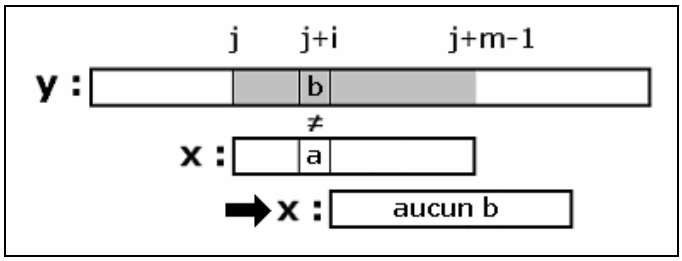


La fonction de décalage *du mauvais caractère* consiste à aligner y[j+i] avec son occurrence la plus à droite dans x[0 .. m-2].

Exemple :



Si aucun caractère égal à y[j+i] n'existe dans le motif *x*, aucune occurrence de *x* dans *y* ne peut inclure y[j+i]. Dans ce cas, le début de la fenêtre est alignée avec le caractère qui suit immédiatement y[j+i], c'est-à-dire y[j+i+1].



L'algorithme Boyer-Moore choisit le décalage maximum retourné par la fonction de *bon suffixe* et la fonction *de mauvais caractère*.

Les résultats de cette fonction sont stockés respectivement dans une table *bmGS* et *bmBC*. Celles-ci peuvent être calculées en temps O(m + σ) avant la phase de recherche, et nécessitent un espace mémoire de O(m + σ), où σ est la taille de l'alphabet.

```
Procédure BoyerMoore (x, m, y, n)

  calculBmGS (x, m, bmGS) // Fonctions non détaillées ici
  calculBmBC (x, m, bmBC)

  j := 0
  tantque (j <= n – m) faire
    i := m – 1
    tantque ((i >= 0) et (x[i] = y[j+i])) faire
      si (i < 0) alors
        MOTIF TROUVE en position j
        j := j + bmGS[0]
      sinon
        j := j + MAX(bmGS[i], bmBC[y[j+i]] – m + 1 + i)
      finsi

      i := i – 1
    fintantque
  fintantque
```

Sur des alphabets très grands, l'algorithme de Boyer-Moore est extrêmement rapide. La recherche de $a^{m-1}b$ dans b^n ne nécessite que O(n / m) comparaisons, ce qui est le minimum absolu pour tout algorithme de pattern matching où seul le motif est prétraité.

L'algorithme au pire des cas est en temps O(mn). Cependant, en pratique, on constate que le nombre total de comparaisons est très souvent inférieur à la longueur du texte. En fait, il a été prouvé que cet algorithme est sous-linéaire en moyenne.

6) Algorithme de Aho - Corasick

Jusqu'à présent, nous avons vu des algorithmes qui permettent de rechercher un et un seul motif dans un texte à la fois. Or, les systèmes de détection d'intrusions doivent parfois vérifier la présence de plusieurs centaines de motifs dans chaque paquet qui arrive. Il est dès lors impensable de lancer séquentiellement autant de recherches que de motifs à chercher.

Pour réaliser une telle opération, il est nécessaire d'utiliser un algorithme qui effectue la recherche parallèlement pour chaque motif. C'est essentiellement le fonctionnement de l'algorithme de *Aho-Corasick*, qui est une généralisation de l'algorithme de *Knuth, Morris et Pratt* au cas de plusieurs motifs à rechercher.

La première étape de l'algorithme est de construire un automate déterministe reconnaissant l'ensemble A*X, où A est l'alphabet et X un ensemble fini de mots à rechercher.

Soit donc X un ensemble fini de mots sur un alphabet A, et soit P l'ensemble des préfixes des mots de X. On construit un automate : $A = (P, \varepsilon, P \cap A^*X)$ reconnaissant A^*X, dont P est l'ensemble d'états, ε est l'état initial et $P \cap A^*X$ l'ensemble d'états terminaux.

La fonction de transition est définie, pour $p \in P$ et $a \in A$, par $p.a = f_X(pa)$. La fonction f_X est définie par : $f_X(u)$ = le plus long suffixe de u qui est dans P.

Cette fonction de transition de l'automate s'exprime à l'aide des bords. En effet :

$$p \,.\, a = \begin{cases} f_X(pa) \text{ si } pa \in P; \\ Bord_X(pa) \text{ sinon;} \end{cases}$$

et de plus,

$$Bord_X(pa) = \begin{cases} Bord_X(p) \,.\, a \text{ si } p \neq \notin \\ \notin \text{ sinon} \end{cases}$$

Nous pouvons ainsi exprimer la fonction de transition comme suit :

```
Fonction TRANSITION(p,a):
  tantque pa ∉ P et p ≠ ε faire
    p := BordX(p)
  fintantque
  si pa ∈ p alors retourner (pa) sinon retourner (ε)
```

La recherche des occurrences des mots de X dans un texte $t = t_1 \ldots t_n$ se fait par l'algorithme suivant, qui incorpore l'évaluation des transitions :

```
Procédure AHO_CORASICK(X, t):
  q := ε  // état initial
  pour i de 1 à n faire // parcours du texte
    tantque qt_i ∉ P et q ≠ ε faire
      q := BordX[q]
    fintantque

    si qt_i ∈ P alors
      q := qt_i  // on avance dans l'automate
    sinon
      q := ε  // motif introuvable : on retourne au début
    finsi

    si q est un état final alors
      MOTIF q TROUVE en position i
    finsi
  finpour
```

Nous pouvons remarquer que l'algorithme de recherche des occurrences des mots de X dans t s'effectue en temps O(n + m) et en place O(m).

7) Snort et le pattern matching

Comme il l'a été évoqué précédemment, le pattern matching joue un rôle important dans les performances de NIDS basés sur des signatures, tel que Snort. Le choix d'un algorithme efficace et adapté aux besoins est donc nécessaire. Mais comme nous allons le voir, cela n'est pas suffisant.

Commençons par un petit historique de Snort. Les versions initiales de ce NIDS utilisaient un algorithme brute force de pattern matching. Celui-ci était bien sûr très lent et a vite montré ses limites.

La première amélioration apportée fut l'implémentation de l'algorithme de *Boyer-Moore* qui apporta de meilleures performances. Bien que très efficace pour de nombreuses applications, cet algorithme n'était pas assez performant dans le cas d'un NIDS doté de milliers de signatures.

En 2002, la version 2.0 de Snort est dotée d'un moteur de recherche multi-motifs très rapide, basé sur l'algorithme de *Aho-Corasick.* Grâce à cela, Snort réussit le test OSEC (http://osec.neohapsis.com) réalisé à 750Mbits/sec. Cependant, la quête aux performances était loin d'être finie.

Pour améliorer les performances, l'équipe de recherche de Snort savait qu'il n'était plus nécessaire de changer d'algorithme. Elle décida ainsi de modifier principalement l'implémentation de l'algorithme *Aho-Corasick*. En effet, le choix de structures de données adaptées n'est pas négligeable pour améliorer la vitesse d'exécution.

La représentation mémoire de la machine à états de l'algorithme *Aho-Corasick* est en général une matrice où les lignes représentent les états, et les colonnes représentent les événements. Les éléments de la matrice fournissent l'état suivant correct en fonction de l'événement d'entrée.

Par le choix de la représentation de cette table d'états, un compromis mémoire – performance est déterminé. De nombreuses applications utilisent des méthodes de représentation très compressée, ce qui a pour conséquence de diminuer les performances de recherche.

Il n'était pas pensable que Snort utilise ces méthodes de compression optimale. Cependant, il était important de minimiser au mieux la taille mémoire de l'automate. Une bonne représentation des signatures apporte non seulement un gain d'espace mémoire, mais peut aussi apporter un énorme gain de performance grâce à une zone mémoire à accès très rapide : le cache.

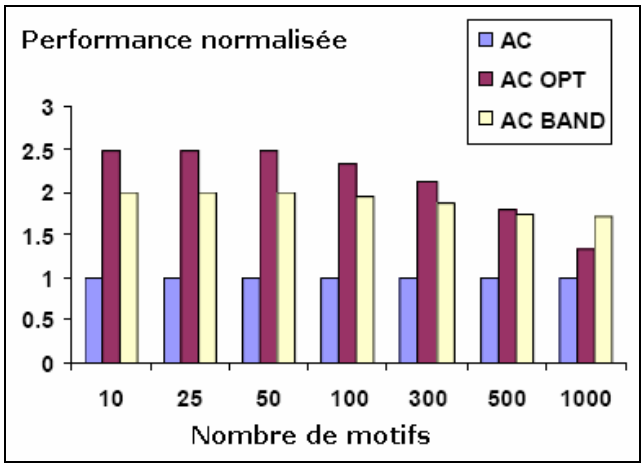
L'optimum serait bien sûr de pouvoir stocker toutes les signatures dans le cache, afin d'y accéder très rapidement. Cependant, cela est difficilement réalisable du fait de la faible capacité des caches et aussi du nombre important (et croissant) des signatures de Snort.

L'équipe de Snort décida de représenter l'automate à états finis de l'algorithme par une matrice creuse (sparse matrix). Dans le domaine de l'algèbre linéaire, il existe des méthodes et des formats de stockage très performants pour de telles matrices. Cependant, il n'est pas suffisant de stocker les données creuses de manière efficace, il faut surtout être capable d'y réaliser un accès aléatoire rapide.

Après avoir comparé plusieurs formats de stockage, les chercheurs de Snort choisirent le format "Banded-Row", constitués de deux vecteurs et de deux index. L'implémentation de l'algorithme de *Aho-Corasick* fut ensuite légèrement modifiée pour s'adapter à la nouvelle structure de données.

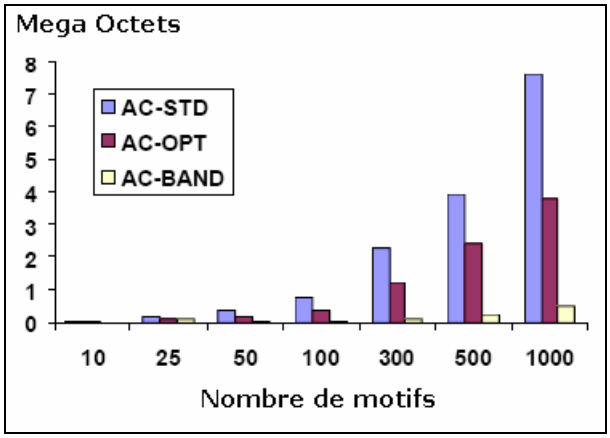
De nombreux benchmarks ont été réalisés afin de s'assurer du gain de performance. Ces benchmarks testaient trois versions : Aho-Corasick standard, Aho-Corasick optimisé avec matrices complètes et Aho-Corasick optimisé avec le format "Banded-Row".

L'illustration suivante montre les performances des trois implémentations, normalisées par rapport à l'algorithme standard.



Nous pouvons remarquer que l'implémentation avec matrices complètes est 2 à 2,5 fois plus rapide que l'implémentation standard. L'implémentation avec le format "Banded-Row" est plus rapide que la version standard mais son intérêt par rapport à la version "matrices complètes" n'apparaît qu'à partir de 1000 motifs. Snort possède plus de 1000 signatures et par conséquent, cette version se montre la plus performante.

L'illustration suivante montre l'espace mémoire nécessaire pour chaque implémentation.



Nous constatons que le stockage "Banded-Row" ne nécessite qu'1/15ème du stockage de la version standard et 1/7ème de la version avec matrices complètes. Là encore, cette implémentation se montre la plus optimisée.

8) Conclusion

Plusieurs algorithmes de pattern-matching ont été présentés, et nous avons vu qu'il n'était pas aisé d'avoir un algorithme efficace dans le cadre d'un NIDS, du fait des contraintes de vitesse des réseaux. Le tableau suivant reprend les caractéristiques majeures des algorithmes analysés précédemment.

Algorithme	***Caractéristiques***
Naïf	- Pas de prétraitement - Fenêtre toujours décalée d'une position vers la droite - Phase de recherche en complexité temps O(mn)

Morris-Pratt	- Prétraitement de complexité temps et espace O(m) - Phase de recherche de complexité temps O(m + n) - Délai borné par m
Knuth-Morris-Pratt	- Prétraitement de complexité temps et espace O(m) - Phase de recherche de complexité temps O(m + n) - Délai borné par $\log_\phi$ (m)
Boyer-Moore	- Comparaisons de droite à gauche - Prétraitement de complexité temps et espace O(m + σ) - Phase de recherche de complexité temps O(mn)
Aho-Corasick	- Recherche multi-motifs - Construction d'un automate déterministe - Phase de recherche de complexité temps O(m + n)

Enfin, nous avons vu que le choix d'un bon algorithme n'était pas suffisant : l'implémentation et les structures de données choisies jouent un rôle très important dans les performances d'un algorithme.

VII. Conclusion

Bien que répandus dans les organisations aujourd'hui, les systèmes de détection d'intrusions ne représentent qu'un maillon d'une politique de sécurité. En effet, même si ceux-ci permettent la détection, parfois l'arrêt, des intrusions, ils restent néanmoins vulnérables eux aussi faces aux attaques externes.

C'est pourquoi, pour une sécurité optimale, ces outils doivent être couplés à d'autres, comme l'indispensable pare-feu. Mais ils doivent aussi être mis à jour, aussi bien le cœur du logiciel comme la base de signatures, qui constitue la base d'une détection efficace. Il faut également coupler les systèmes de détection entre eux : c'est-à-dire ne pas hésiter à placer des NIDS, HIDS et KIDS dans le même réseau. Leurs rôles sont différents, et chacun apporte ses fonctionnalités.

L'efficacité des détections passe aussi par une bonne implémentation des algorithmes de recherche. L'étude que nous avons menée sur ces algorithmes nous a permis d'observer ô combien il est indispensable d'utiliser des structures de données appropriées lors d'un développement. Mais ce n'est pas tout, au-delà de ces structures, doit résider un savoir faire important de la part du (des) concepteur(s).

Toutefois, et nous terminerons par ceci, même si une certaine maturité dans ce domaine commence à se sentir, le plus important reste de savoir de quoi il faut se protéger. Les failles les plus répandues proviennent généralement de l'intérieur de l'entreprise, et non de l'extérieur. Des mots de passe simples, des droits d'accès trop élevés, des services mal configurés, ou encore des failles dans les logiciels restent la bête noire en matière de sécurité.

Bibliographie

Exact String Matching Algorithms
Christian Charras et Thierry Lecroq
http://www-igm.univ-mlv.fr/~lecroq/string/

Eléments d'algorithmique - Recherche de motifs
D. Beauquier, J. Berstel, Ph. Chrétienne
http://www-igm.univ-mlv.fr/~berstel/Elements/Elements.html

Optimizing Pattern Matching for Intrusion Detection
Marc Norton

Increasing Performance in High Speed NIDS, A look at Snort's internals
Neil Desei

Managing Security with Snort and IDS Tools
Kerry J. Cox, Christopher Gerg
O'Reilly – ISBN : 0-596-00661-6

Les IDS – Les systèmes de détection d'intrusions informatiques
Thierry Evangelista
Dunod – ISBN 2 10 007257 9

Les attaques externes
Eric Detoisien

Détection et tolérance d'intrusions
Samuel Dralet

Les tests d'intrusions
Eric Detoisien et Frédéric Raynal

Bro: Un autre IDS Open-Source
Jean-Philippe Luiggi
Misc Magazine n°14

Snort User Manual
The Snort Project
http://www.snort.org

Prelude HandBook
Prelude Hybrid IDS project
http://www.prelude-ids.org/

Bro Reference Manual
Lawrence Berkeley National Laboratory
http://www.bro-ids.org

LIDS Documentation
LIDS Project
http://www.lids.org

Articles Wikipedia
http://www.wikipedia.org

Annexe : Rappels essentiels concernant les protocoles

La plupart des attaques se font au niveau des couches hautes du modèle OSI (Session, Présentation & Application). Cependant, il est important de connaître le fonctionnement des couches basses 3 et 4 (Réseaux et Transport) pour comprendre certaines attaques.

L'en-tête d'un datagramme IP est composé :

- d'une partie fixe de 20 octets
- d'une partie à longueur variable : les options

Il est utile de connaître les champs du protocole IP, qui sont utilisés à des fins de reconnaissance ou d'attaque par déni de services :

- **Version** (4 bits) : version du protocole (IPv4, bientôt IPv6)
- **En-tête** (4 bits) : spécifie la longueur de l'en-tête, en mots de 32bits. Minimum 5 car il y a 20 octets au minimum dans l'en-tête
- **Type de service** (8 bits) : notion de qualité de service
- **Longueur totale** (16 bits) : longueur du datagramme complet (en-têtes + données) : max 65 535 octets
- **Identification** (16 bits) : valeur pour identifier les fragments d'un même datagramme
- **Flags** (3 bits) : pour contrôler la fragmentation IP
  - Bit 0 : réservé ➔ doit être laissé à 0
  - Bit 1 : Accept Fragmentation (AF) ➔ 0 si autorisée
  - Bit 2 : Don't Fragment (DF) ➔ 0 quand il s'agit du dernier fragment
- **Offset de fragmentation** (13 bits) : spécifie le décalage du premier octet par rapport au datagramme complet. Le décalage du premier fragment vaut 0. Le maximum pour un datagramme est de 8192 fragments
- **Durée de vie / TTL** (8 bits) : permet de limiter la durée de vie d'un paquet sur le réseau. Est décrémentée par chaque nœud de transit.
- **Protocole** (8 bits) : indique le protocole de niveau supérieur qui est utilisé (ex : ICMP/IGMP/TCP/UDP)
- **Checksum d'en-tête** (16 bits) : somme de contrôle calculée sur l'en-tête uniquement. Doit être recalculé à chaque modification du TTL ou du ToS
- **Adresse source** (32 bits)
- **Adresse destination** (32 bits)
- **Options** (longueur variable) : rarement utilisé

L'en-tête, permettant d'identifier le protocole de niveau supérieur n'est inclus que dans le premier fragment.

Pour établir une connexion TCP, il y a un protocole de négociation appelé « poignée de main TCP » (handshake). Celle-ci se déroule en 3 phases :

1) La source émet une requête d'établissement de connexion : bit SYN à 1 + numéro de séquence aléatoire (ISN)
2) Si la destination accepte : bits SYN et ACK à 1 + numéro de séquence propre + numéro de séquence de la source augmenté de 1
3) La source acquitte la réponse de la destination en positionnant le bit ACK à 1, en incrémentant son numéro de séquence originel et un numéro d'acquittement égal au numéro de séquence de la destination augmenté de 1

Les champs de TCP, comme ceux de IP, sont souvent utilisés par des pirates à des fins intrusives. Voici un rappel de ces champs :

- **Port source & port destination** (16 bits) : déterminent le service requis
- **Numéro de séquence** (32 bits) : numéro du premier octet de données par rapport au début de la transmissions, sauf si SYN est marqué. Dans ce cas, numéro de séquence = ISN.
- **Accusé de réception** (32 bits) : Si ACK est marqué, ce champ contient le numéro de séquence du prochain octet que le récepteur s'attend à recevoir
- **Header Length** (4 bits) : taille de l'en-tête TCP en nombre de mots de 32 bits
- **Réservé (6 bits)** : usage futur ➔ toujours à 0
- **Flags** (6 bits) : utilisés pour contrôle et gestion des connexions TCP
  - ***URG*** : champ « pointeur de données urgentes » significatif
  - ***ACK*** : champ « accusé de réception » significatif
  - ***PSH*** : fonction Push
  - ***RST*** : réinitialisation de la connexion
  - ***SYN*** : synchronisation des numéros de séquence
  - ***FIN*** : fin de connexion
- **Fenêtre** (16 bits) : nombre d'octets à partir de la position marquée dans l'accusé de réception que le récepteur est capable de recevoir
- **Checksum** (16 bits) : somme de contrôle calculée sur le datagramme
- **Pointeur de données urgentes** (16 bits) : spécifie la position d'une donnée urgente (position par rapport au numéro de séquence)
- **Options (variable)** : doit être un multiple de 8 bits ➔ remplissage (padding) éventuel
  - ***Maximum Segment Size*** (MSS) : taille maximale des segments : envoyée dans la requête de connexion initiale
  - ***Window Scale*** : permet de définir des tailles de fenêtre supérieurs à 64Ko
  - ***Selective Acknowledgement*** (SACK) : acquittements sélectifs
  - ***NOP (No OPeration)*** : séparateur pour aligner le début d'une option sur le début d'un mot de 16 bits

Voici également un rappel des champs du protocole UDP :

- **Port source** (16 bits)
- **Port destinataire** (16 bits)
- **Longueur** (16 bits) : nombre d'octets du datagramme entier (y compris en-tête). Min = 8
- **Checksum** (16 bits)

Le protocole ICMP est souvent utilisé à des fins malicieuses car c'est un protocole utile pour des *messages flash*, et il ne nécessite pas de connexion ni de port.

Les paquets ICMP sont encapsulés dans les paquets IP pour des raisons historiques.

Les champs ICMP sont les suivants :

- **Type** (8 bits) : type de message
- **Code** (8 bits) : précise le message identifié, par rapport au type de message
  - ***Type 0*** (Echo reply) ➔ pas de code
  - ***Type 3*** (Destination Unreachable) ➔ 0 = net unreachable, 1 = host unreachable, 2 = protocole, …
  - ***Type 4*** (Source Quench) : contrôle de flux
  - ***Type 5*** (Redirect) ➔ Code 0 = network, 1 = host
  - ***Type 8*** (Echo request)
  - ***Type 11*** (Time exceeded) : durée de vie écoulée
  - ***Type 13*** (Timestamp) : marqueur temporel
  - ***Type 14*** (Timestamp reply) : réponse au marqueur temporel
  - ***Type 30*** (Traceroute)