Université de Québec en Outaouais (UQO)

Mémoire de Maîtrise

Modèle quantitatif pour la détection d'intrusion.

Une architecture collaborative IDS-HONEYPOT

Présenté par:

Hatem Bouzayani

Sous la direction de :

Pr. Kamel Adi

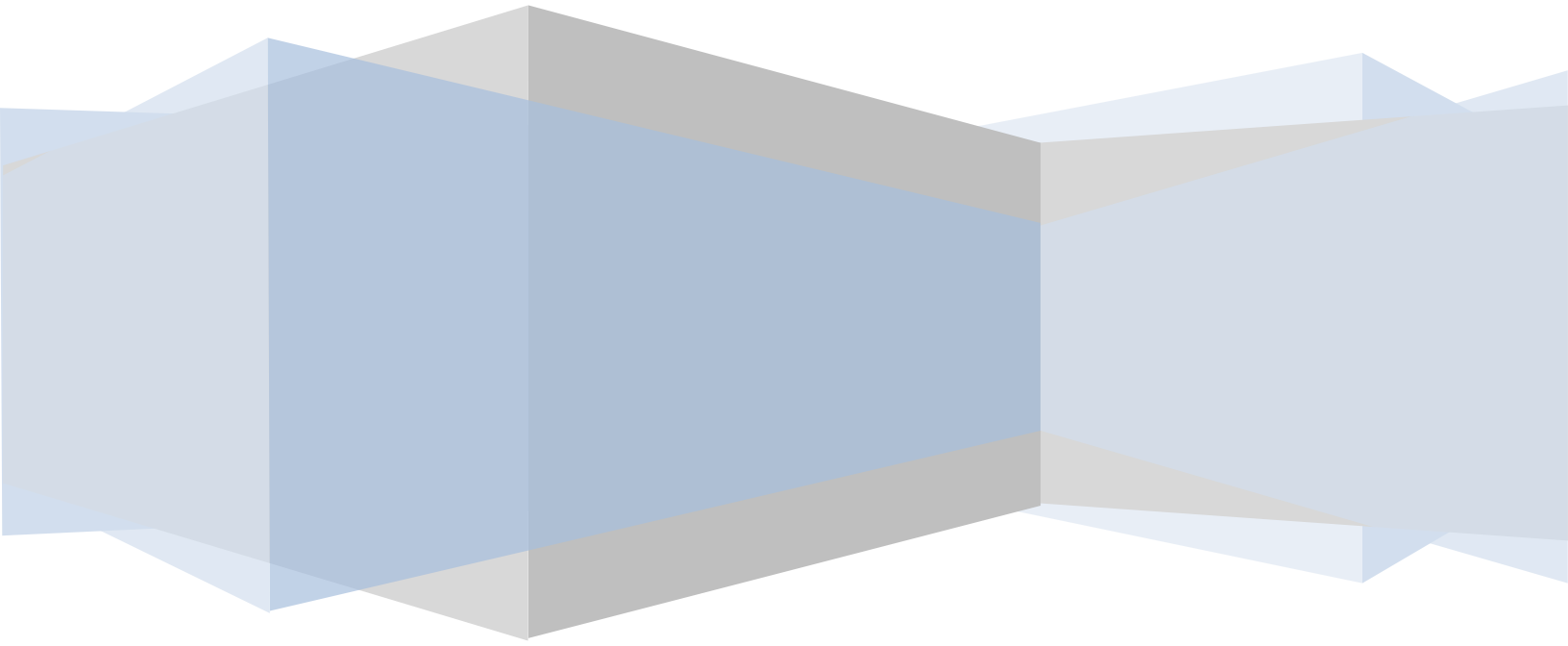


TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES AFFICHAGES

Remerciements

Je tiens à remercier professeur Kamel Adi pour l'encadrement qu'il m'a prodigué dans ma maîtrise en n'oubliant pas son aide et ses orientations.

Je tiens ensuite à remercier Mr Ammar Boulaiche étudiant en doctorat à l'université de Bejaia (Algérie) pour son aide et sa coopération durant l'implémentation et l'installation du système.

Je remercie les membres du jury Prof. Luigi Logrippo et Prof. Mohamed Mejri.

Je remercie les enseignants du DII.

Enfin, et pour toujours, je remercie vivement tous les membres de ma famille qui m'ont toujours supporté et partagé la surcharge de travail avec moi durant ces longues années d'études.

Hatem Bouzayani

INTRODUCTION

De nos jours, les attaques envers les systèmes informatiques sont devenues nombreuses, plus spécifiques, puissantes, intelligentes et causent des dégâts considérables. Malheureusement, la plupart de ces attaques sont nouvelles et inconnues par les systèmes de protection, ce qui demande parfois des interventions complexes et coûteuses pour la récupération et la maintenance après attaque. Ainsi, dans la littérature, plusieurs travaux ont abordé le problème de la mise en place de techniques permettant d'augmenter le pouvoir de prévention et de détection contre les attaques malicieuses. Les systèmes de détection d'intrusions (*IDS : Intrusion Detection System*) sont des outils conçus pour la détection des tentatives d'attaques sur un réseau. Cependant, la détection de telles attaques est souvent basée sur des connaissances préalables sur les attaques elles-mêmes. Malheureusement, la nécessité d'avoir au préalable ces connaissances constitue le principal point de faiblesse qui amène l'IDS à générer de fausses alertes (faux positifs et les faux négatifs).

Les pots de miel (*honeypots*) sont des outils utilisés dans les systèmes informatiques pour étudier et acquérir des connaissances sur le comportement et les techniques utilisées par les pirates pour attaquer les systèmes informatiques. Tout trafic entrant vers un pot de miel est considéré comme suspect et constitue potentiellement une attaque. C'est pour cela que le trafic enregistré au niveau d'un pot de miel constitue une excellente source de données pour étudier les attaques sur les réseaux.

Les *IDS* et les pots de miel ont tous les deux des limites et aucun d'eux ne peut remplacer l'autre, par contre ils peuvent se compléter pour constituer un système plus sécurisé. En effet, dans notre travail de recherche, nous proposons de faire collaborer les deux technologies IDS et pot de miel pour construire une architecture collaborative permettant d'améliorer la capacité de détection d'intrusions. Notre contribution se base sur un modèle quantitatif probabiliste et utilise des techniques de fouille de données (*Data Mining*) permettant d'enrichir les connaissances d'un IDS et ainsi offrir une meilleure efficacité de détection d'intrusion.

PARTIE A : L'ÉTAT DE L'ART

CHAPITRE I

POTS DE MIEL (HONEYPOTS)

Un pot de miel est un outil utilisé fréquemment dans le domaine de la sécurité informatique. Il permet essentiellement d'étudier et d'analyser les attaques sur les systèmes informatiques et le comportement des intrus. Un pot de miel est souvent installé dans un réseau DMZ d'une compagnie pour qu'il soit attaqué ou compromis afin de collecter et d'enregistrer le maximum d'informations sur les actions des attaquants.

Ce chapitre contient une introduction globale sur les pots de miel et détaille le comment et le pourquoi de leurs utilisations dans le domaine de la sécurité informatique. Nous présentons les différentes catégories de pots de miel et détaillons les technologies pots de miel les plus connus.

I.1 Introduction

Les pots de miel (*Honeypots*) des systèmes informatiques configurés dans le but qu'ils soient compromis, sondés ou attaqués par les *pirates* informatiques. Malgré les risques liés à leurs utilisations, les pots de miel constituent une source d'information remarquable sur la nature des attaquants, leurs objectifs d'attaques, leurs méthodes d'attaques et leurs comportements face aux différents systèmes. Les générations successives de ces technologies ont permit d'apporter des améliorations considérables de la valeur de l'information recueillie tout en réduisant les risques de leurs détection par les intrus.

L'idée derrière un pot de miel est simple. Il s'agit de mettre en place un moyen pour contrôler les attaques et les activités des attaquants en leur donnant accès à quelques services, parfois émulés, pour qu'ils puissent interagir avec eux tout en limitant les dégâts causés par ces attaques puisse que l'attaquant n'a pas accès aux serveurs réels en production. Cependant, la

quantité et la qualité de l'information recueillie est directement proportionnelle au degré d'interaction offert par le pot de miel. Ainsi, si les services offerts sont très limités, le pot de miel ne sera pas « très attractif » pour espérer recueillir de bonnes informations et l'attrape mise en place risque d'être facilement détectée et ainsi évitée par les intrus. Toutefois, les versions actuelles offrent plus d'interaction (plus de services), donc plus attractif et offrent une analyse de données quasi automatique intégrant même des technologies avancées de fouilles de données (*Data Mining*).

I.2 Historique

Les premières tentatives pour construire des systèmes pots de miel remontent aux années 90 [1] :

- **1990-1991** : « *The Cuckoo's Egg »* est le premier livre publié sur le sujet, il est écrit par *Clifford Stoll*, ce livre documente la traque et le suivit d'un intrus qui a été capable d'entrer dans un ordinateur du laboratoire LBNL *Lewrence Berkeley National Laboratory*.
- **1997** : la première version de *Deception Toolkits (DTK)* est publiée par *Fred Cohen.* Ce logiciel est connu comme le premier pot de miel *(honeypot)* utilisé dans le domaine de sécurité.
- **1998** : « *CyberCop Sting »* est le premier pot de miel commerciale, il collecte et contrôle l'information sur plusieurs systèmes virtuels.
- **1998 :** publication de *BackOfficer Friendly*, un pot de miel gratuit et simple d'utilisation il ne s'installe que sur la plateforme Windows.
- **1999** : début du *Honeynet Project*. Plusieurs publications dans la série « *Know Your Ennemy* » décrivent les avancées techniques dans ce projet. Le travail dans ce projet continu à ce jour et plusieurs projets similaires ont été définis dans différent pays, pour centraliser la collecte d'informations sur les attaques système.
- **2000-2001** : utilisation des pots de miel pour capturer et étudier les activités des vers « *worms »*. Par ailleurs, plusieurs organisations ont commencé à déployer les pots de miel pour détecter des attaques informatiques et chercher les vulnérabilités « Zero day ».
- **2002** : un *pot* de miel est utilisé pour détecter et capturer une nouvelle attaque sur le système *Solaris.*

I.3 Définition des pots de miel

Dans la littérature, il y a diverses définitions pour les pots de miel [8, 9, 10, 11] selon le contexte d'utilisation. En effet, certains considèrent qu'un pot de miel est un outil pour attirer les attaquants alors que d'autres le considèrent plutôt comme un outil de détection d'intrusions.

Dans le reste de ce document nous adoptons la définition suivante :

«Un pot de miel est une ressource sécurisée qui est mise en place et qui a pour objectif d'attirer des pirates dans le but d'être attaqué ou compromis» [1].

Victor Garcia sur le site *Canadian honeypot Security Research* [2] présente la définition suivante :

«Un pot de miel est une contre-mesure efficace visant à empêcher l'utilisation non autorisée de systèmes d'information critiques dans les réseaux».

I.4 Avantages et inconvénients des pots de miel

L'information collectée par les pots de miel a une valeur très importante car elle est principalement constituée par du trafique suspect (tentatives d'intrusion ou d'attaques du système, etc.). Ces informations sont souvent rassemblées sous des formats lisibles et compréhensibles, ce qui facilite leurs analyses et interprétations manuelle ou automatique. Par exemple, ces données peuvent être utilisées pour des analyses statistiques ou pour la découverte des nouveaux patrons d'attaques par utilisation des techniques de fouilles de données (*Data Mining*).

Les pots de miel sont des outils simples à installer, faciles à configurer et à utiliser. Ils ne demandent pas de matériel ou logiciel spécifique pour les utiliser ou les configurer. Cependant, cette technologie souffre de plusieurs limitations :

Les pots de miel n'attrapent ou ne voient que les attaques dirigées vers le système dans lesquels ils sont installés. Ainsi, les données collectées ne concernent que le trafic local au pot de miel sans considération pour le reste du réseau.

Quelque soit la qualité d'un pot de miel, ce denier peut toujours être détecté par les attaquants en se basant sur sa signature (comportements et caractères spécifiques). Si un pot de miel est détecté par un intrus, alors l'information collectée sera de moindre qualité. Les techniques de

détection de pots de miel sont connues sous le nom de « *Fingerprinting* » et sont surtout efficace contre les pots de miel de recherche (voir section I.5).

En envoyant des informations truquées vers ces pots de miel et selon les réponses spécifiques reçues, ces derniers peuvent alors être facilement identifiés.

Une fois découvert, un pot de miel peut être utilisé pour attaquer le reste du système [6, 7]. Le niveau de risque dépend alors du niveau du pot de miel : plus le niveau d'interaction est bas (voir section I.5), plus le risque est réduit.

I.5 Classification des pots de miel

Nous pouvons distinguer deux types de pots de miel : (i) les pots de miel de production qui sont utilisés par les organisations industrielles afin de pouvoir détecter les attaques venant de l'extérieur et (ii) les pots de miel de recherche qui sont utilisés par les chercheurs voulant étudier les activités des pirates. Un pot de miel peut être construit de différentes manières : il doit répondre aux objectifs que nous lui fixons et sa valeur et sa catégorie peuvent varier selon le degré de complexité et le niveau d'interaction exigé de lui. Par ailleurs, plus le niveau d'interaction d'un pot de miel est faible plus l'impact d'une éventuelle attaque sera faible. À l'inverse, plus le niveau d'interaction d'un pot de miel est fort, plus l'impact d'une éventuelle attaque sera sévère.

Dans le tableau 1, nous pouvons distinguer les différences entre les pots de miel selon les niveaux d'interactions.

Niveau d'interaction	**Installation & configuration**	**Déploiement & maintenance**	**Informations collectées**	**Niveau de risque**
Bas	Simple	Simple	Limitée	Bas
Moyen	impliqué	impliqué	Variables	Moyen
Haut	Difficile	Difficile	Riche	Haut

Tableau 1 : Différents niveaux des pots de miel

Les pots de miel à faible interaction

Ils sont plus faciles à installer, à configurer, à déployer et à maintenir grâce à leur conception assez simple. Ces pots de miel sont connus surtout par leurs pouvoirs de détection de connexions non autorisées. Puisque les fonctionnalités offertes par ces pots de miel sont limitées le niveau de risque d'attaque est donc, lui aussi, limité. Ce type de pot de miel n'est

pas conçu pour découvrir de nouvelles attaques, mais plutôt pour contrôler et analyser l'entourage réseau de l'endroit où ils sont installés. Ces pots de miel sont recommandés pour une utilisation personnelle et aux organisations les utilisant pour la première fois comme une étape d'étude.

Les pots de miel à moyenne interaction

Ces pots de miel offrent plus de fonctionnalités que ceux de la catégorie précédente et leurs niveaux d'interaction sont donc relativement plus élevés. Ils demandent plus d'attention lors de l'installation et de la configuration. Par contre, la configuration doit toujours prendre en considération la sécurité du réseau quand le pot de miel est attaqué. La tâche de configuration devient plus compliquée lorsque le degré de complexité du pot de miel est plus élevé. Dans [12] *Georg Wicherski* a utilisé ce type de pots de miel pour capturer les attaques malicieuses (*malwares).*

Les pots de miel à haute interaction

Ces pots de miel offrent plus d'information sur les attaques, mais consomment plus de temps lors de l'installation, de la configuration et de la maintenance. Ils présentent aussi un niveau de risque très élevé car ils donnent aux attaquants un plus grand contrôle sur le système d'exploitation. Par ailleurs, ces pots de miel sont souvent installés dans une portion de réseau non contrôlée (ex. : absence d'un pare-feu) et il est souvent nécessaire de renforcer la sécurité du reste du réseau afin de minimiser le risque que le pot de miel ne soit utilisé comme point de départ pour une attaque.

I.6 Quelques types de pots de miel

I.6.1 BackOfficer Friendly (BOF)

C'est le pot de miel le plus simple à utiliser, à comprendre et à configurer et il est à bas niveau d'interaction. Il s'agit d'un pot de miel de production qui peut être installé sur une plate-forme Windows ou Unix et il est « *open source* ». Malgré sa simplicité d'utilisation, il est malheureusement trop restrictif dans ses fonctionnalités [1]. En effet, son principal rôle est de découvrir des attaques de type « *scan* » sur des services spéciaux tels que Telnet, FTP, SMTP, POP3 et IMAP2. Une fois une connexion détectée, le pot de miel permet à l'utilisateur pour de bloquer d'éventuelles autres demandes de connexions, puis il génère une alerte contenant

quelques informations telles que la date, le service sur lequel l'attaque s'est faite, l'adresse IP de l'attaquant, etc. [3]. Le fonctionnement de BOF se base sur la création d'écouteurs sur les ports ou par la création de *Sockets* qui seront liés à ces ports. Cependant BOF ne peut écouter sur un port qui est déjà occupé par un autre service. Par ailleurs, BOF ne peut être configuré pour créer un nouveau service en dehors des sept services de base qu'il peut assurer :

- ***Back Orifice*** : service de valeur limité, écoute sur le port UDP 31337.
- ***FTP*** : *File Transfert Protocol*, utilisé pour le transfert des fichiers, écoute sur le port TCP 21, un port majeur utilisé pour les attaquants comme cible.
- ***Telnet*** : *Cleartext Protocol*, utilisé pour l'administration des systèmes à distance, écoute sur le port TCP 25.
- ***SMTP*** : *Simple Mail Transfert Protocol*, utilisé pour envoyer et recevoir des courriels, écoute sur le port TCP 25.
- ***HTTP*** : *Hyper-Text Transfert Protocol*, écoute sur le port TCP 80. Parmi tous les services offerts par BOF, c'est le service le plus ciblé.
- ***POP3*** : *Post-Office Protocol*, protocole de courrier électronique, utilise le port TCP 110.
- ***IMAP*** : *Internet Message Access Protocol*, protocole de courrier électronique, utilise le port TCP 143

Les informations fournies par les alertes de BOF sont limitées, car ce dernier gère uniquement sept services. Par ailleurs, BOF ne peut pas être contrôlé à distance et il n'est pas capable d'envoyer les alertes par courriel vers l'administrateur. De plus, le contrôle de plusieurs BOF demande un grand investissement dû à l'absence de mécanismes de communication et de synchronisation. Par contre, les risques d'attaques sont assez réduits dû au nombre réduit de fonctionnalités offertes.

I.6.2 Specter

Specter est un pot de miel à bas niveau d'interaction, c'est une technologie commerciale crée et supportée par *NetSec* (http://netsec.biz/), une compagnie Suisse qui s'occupe de la sécurité des réseaux. Il ne peut fonctionner que sous la plateforme Windows et il peut gérer plus de ports que BOF et émuler sept services : six pièges (*traps*) fixes et un septième service à

configuration ouverte, ce qui permet à *Specter* d'analyser un nombre important de types d'attaques. *Specter* peut donner l'impression aux attaquants d'être réellement dans un vrai système en production par l'envoi de *feadback* incorrects [1]. Les pièges (*traps*) ne sont pas des services émulés, mais correspondent à des ports préconfigurés pour écouter et permettre la journalisation (le *log*) des connexions. Le septième port est à configurer selon le choix de l'administrateur, il est extrêmement utile, car il peut être configuré pour s'adapter aux nouvelles vulnérabilités. *Specter* peut aussi émuler 13 différents systèmes d'exploitation un système à la fois, ce qui donne à *Specter* plus de flexibilité pour s'adapter à plusieurs environnements. À chaque pot de miel *Specter* est assigné un nom propre et une adresse de domaine. Chaque pot de miel peut avoir sa propre personnalité et par la suite sa propre identité. Les informations offertes par *Specter* sont, par contre, relativement limitées. Par ailleurs, *Specter* offre plusieurs méthodes pour aviser (alerter) l'administrateur et il est possible de configurer chacune de ces méthodes pour permettre aux administrateurs de personnaliser la manière de détecter les attaques et les intrusions et d'envoyer les alertes en temps réel. L'information recueillie par *Specter* peut être analysée avec plusieurs outils, nous citons ici les trois outils les plus importants [1] :

- ***Log Analyzer*** : *Specter* gère une base de données (*Incident Database*) pour archiver les informations sur les attaques. Cette base de données correspond à une collection de fichiers texte, chacun lié à une attaque précise. Le Log Analyzer permet alors aux administrateurs de faire différentes requêtes. Il offre un GUI simple à utiliser et les résultats des requêtes sont bien affichés.
- ***Event Log*** : sous forme d'un GUI avec moins de fonctionnalités. Il est utilisé pour la maintenance des fichiers de journalisation (*logs*).
- ***Syslog*** : utilisé comme un outil du standard Unix pour recueillir les informations d'un système. L'avantage de cet outil c'est de permettre le regroupement d'informations à partir de plusieurs pots de miel. Ces informations peuvent alors être centralisées dans un serveur de journalisation (*log server*) accessible à distance.

I.6.3 Honeyd

Honeyd est développé par ***Niels Provos*** de l'Université Michigan. La première version a été lancée en avril 2002. C'est un pot de miel à faible niveau d'interaction. Il est configuré sous

forme d’un pot de miel de production et utilisé pour la détection des attaques et des activités non autorisées [1]. *Honeyd* à introduit plusieurs nouveaux concepts :

Avec cet outil il est possible d’installer un ou plusieurs pots de miel virtuels à faible interaction avec différentes personnalités (systèmes) et services sur une seule machine, en leur associant des adresses IP qui ne sont pas encore utilisées dans le réseau réel. Ainsi, toute tentative de connexion à une adresse IP non encore utilisée est considérée comme une activité malveillante.

Honeyd peut détecter les connexions sur n'import quel port TCP, les services émulés ne sont pas requis pour la détection, mais seulement pour l'interaction avec les attaquants

Honeyd peut modifier les services émulés et son niveau d'interaction. Il peut ainsi utiliser un nombre important de services avec des niveaux différents d'interaction et d’émulations. Un autre avantage est sa capacité de pouvoir contrôler des millions d'adresses IP et d'en déclarer quelques milliers d'autres en même temps. Le contrôle ne se fait pas une adresse à la fois, mais il se fait par blocs d'adresses IP spécifiées.

Honeyd fonctionne avec le principe d'émulation des services sur les ports où la connexion d'attaque se déroule. Il commence par interagir avec l'attaquant et capte toutes ses activités, l'émulation va continuer jusqu'à la fin de la connexion et attend s'il y'a d'autres attaques similaires autrement il termine l'émulation.

Pour que *Honeyd* puisse avoir l'information sur le réseau en entier, deux méthodes sont utilisées pour permettre la redirection du trafic :

<u>*Blackholing*</u> : cette méthode utilise un simple routeur entre le réseau local et l'internet pour diriger tout le trafic vers *Honeyd* comme le montre la figure 1.

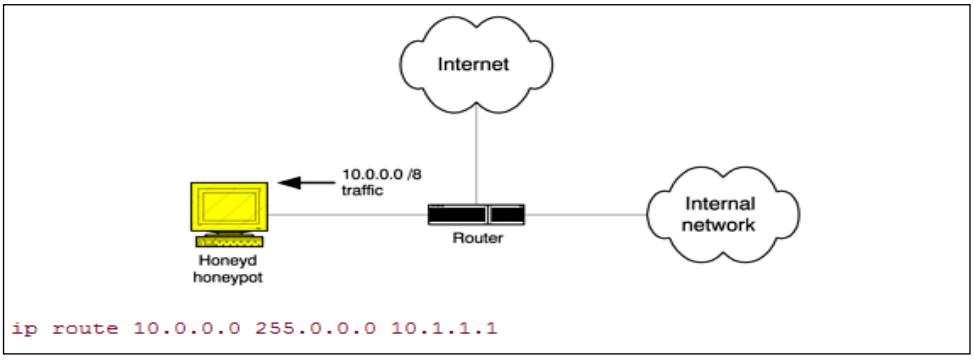


Figure 1 : Redirection du trafic du réseau vers Honeyd [1]

ARP Spoofing : cette méthode est un peu plus complexe. *Honeyd* utilise pour cela le module ARPD. À chaque requête « who is? » du protocole ARP pour trouver l'Adresse MAC correspondant à une adresse IP d'une machine crée par le pot de miel et qui appartient au honeynet (réseau de machines fictives pots de miel crée par *Honeyd*), ARPD répond en donnant l'adresse MAC de la machine sur laquelle est installé le réseau de pots de miel.

Honeyd n'a pas une solution de notification ou d'alerte intégrée. Pour cela il faut utiliser une technologie tierce comme *Swatch* : un outil *OpenSource* qui peut contrôler les *logs* pour des messages spécifiques [1].

I.6.4 ManTrap

ManTrap est le premier pot de miel commercial. Il a été initialement développé par la compagnie *Recource Technologies (recource.com).* Cette compagnie a été, par la suite, rachetée par Symantec (symantec.com) qui manufacture désormais ce produit. C'est un pot de miel professionnel à haute interaction très complet. Il met en œuvre quatre cages (systèmes d'exploitation) logiquement séparés et qui sont installés sur la même machine hôte. Chacun de ces systèmes supporte des applications réelles, et apparaît comme un système indépendant avec sa propre interface réseau. *Mantrap* peut être utilisé comme pot de miel de production, notamment pour la détection et la réaction, et il est aussi rentable comme pot de miel de recherche, mais avec un risque non négligeable.

Parmi les avantages de cette technologie, nous pouvons citer :

- Détection d'activité malveillante sur n'importe quel port de son système grâce au sniffer qu'il emploie
- Capture de toute l'activité des attaquants, incluant le trafic chiffré comme SSH.
- Possède un mécanisme d'administration à distance avec un système d'alerte par courriel.

Par ailleurs, cette technologie souffre de plusieurs inconvénients :

- C'est un pot de miel à haute interaction, donc les attaquants peuvent l'employer pour mener des attaques sur le reste du système.
- Les attaquants peuvent facilement découvrir le « *fingerprint* » et déjouer ainsi le piège.

- Limité aux systèmes d'exploitation Solaris.
- C'est une solution commerciale, donc on ne peut pas le personnaliser librement.

Pour créer les cages, *ManTrap* doit faire quelques modifications sur les composantes du système d'exploitation et le système des fichiers :

Ajustement au niveau du noyau : chaque cage partage les mêmes outils avec le noyau et sauvegarde les informations des utilisateurs et des processus sous format de fichiers logs. Par ailleurs, *ManTrap* peut aussi capturer les informations cryptées, car ces informations sont prises au niveau du noyau.

Système des fichiers : le système des fichiers n'est pas partagé comme dans le cas du noyau, mais chaque cage a sa propre copie de système de fichiers et sa propre structure de fichiers, procédure d'accès, etc. *ManTrap* recueille en continu les informations qui montrent les activités dans les cages et le reste du système même s'il n'y a pas d'attaques. Dès qu'il détecte une attaque, *ManTrap* envoie un courriel à l'administrateur. Après une attaque, *ManTrap* est capable de faire un recouvrement des fichiers pour restaurer le système. Les alertes de *ManTrap* sont détaillées et se font par plusieurs types de courriels formant ensemble un rapport détaillé sur l'attaque. Les informations de log peuvent être consultées de deux manières : par le GUI du *ManTrap* et par les lignes de commandes, comme dans la figure 2 et l'affichage 1.

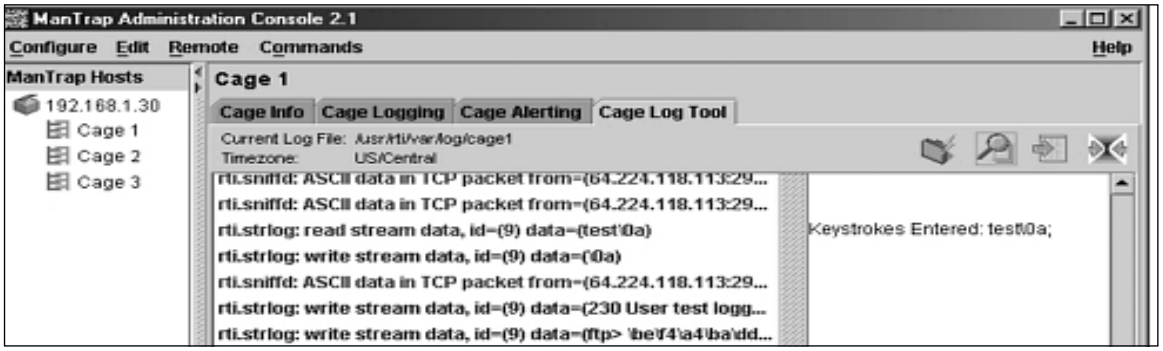


Figure 2 : GUI du *ManTrap* pour consulter le système de log [1]

Avec les lignes de commandes, il est possible d'obtenir plus d'information et les requêtes sont plus précises.

```
mantrap #grep "rti.strlog" /usr/rit/var/log/cage1
2002.01.13:18.31.41:96:rti.strlog: id=(9) data=(w\0a)
2002.01.13:18.31.42:96:rti.strlog: id=(9) data=(ps x\0a)
2002.01.13:18.31.44:96:rti.strlog: id=(9) data=(ps -ef |grep juno\0a)
```

```
   2002.01.13:18.31.50:96:rti.strlog: id=(9) data=(/usr/local/lib/juno\0a)
   2002.01.13:18.31.58:96:rti.strlog: id=(9) data=(/usr/juno\0a)
   2002.01.13:18.32.00:96:rti.strlog: id=(9) data=(cd /usr\0a)
   2002.01.13:18.32.02:96:rti.strlog: id=(9) data=(./juno\0a)
   2002.01.13:18.32.05:96:rti.strlog: id=(9) data=(cd /usr/local/lib\0a)
   2002.01.13:18.32.06:96:rti.strlog: id=(9) data=(./juno\0a)
   2002.01.13:18.32.07:96:rti.strlog: id=(9) data=(cd /usr\0a)
   2002.01.13:18.32.16:96:rti.strlog: id=(9) data=(ftp 10.44.152.111\0a)
   2002.01.13:18.32.18:96:rti.strlog: id=(9) data=(test\0a)
   2002.01.13:18.32.19:96:rti.strlog: id=(9) data=(test\0a)
   2002.01.13:18.32.21:96:rti.strlog: id=(9) data=(get juno\0a)
   2002.01.13:18.32.29:96:rti.strlog: id=(9) data=(\0a)
   2002.01.13:18.32.30:96:rti.strlog: id=(9) data=(bye\0a)
   2002.01.13:18.32.34:96:rti.strlog: id=(9) data=(chmod +x juno\0a)
   2002.01.13:18.32.43:96:rti.strlog: id=(9) data=(./juno 10.112.0.18 1024
&\0a)
   2002.01.13:18.32.58:96:rti.strlog: id=(9) data=(kill -9 12178\0a)
   2002.01.13:18.33.23:96:rti.strlog: id=(9) data=(exit\0a)
```

Affichage 1 : Utilisation de la commande *rti.strlog* pour consulter les informations dans une cage [1]

I.6.5 Honeynets

Un *honeynet* est un réseau de pots de miel combiné avec un ensemble de mécanismes de sécurité comme les pare-feux, les IDS, les serveurs de log, etc. Il donne l'apparence d'un environnement de production complexe avec des failles et des informations pertinentes utilisées comme appât pour attirer et piéger les attaquants. Le *honeynet* permet, ainsi, de surveiller et d'enregistrer ensuite toutes les actions malicieuses des attaquants. Par ailleurs, sa structure en réseau offre l'avantage d'avoir des renseignements sur les communications entre les attaquants et leurs méthodes de collaboration. Ainsi, *honeynet* est considéré comme un outil de premier choix pour apprendre les techniques d'attaques, et les comportements des attaquants, sur un réseau réel. Dans un *honeynet*, il n'y a pas de services émulés, ni de cages (comme dans le cas de *ManTrap*). Il utilise une variété de systèmes pour détecter plusieurs types d'attaques : ceux qui sont connus et ceux qui ne le sont pas (*zero day*). Cependant, *Honeynet* est un outil complexe et difficile à configurer.

Dans l'architecture d'un *honeynet*, trois éléments critiques doivent être présents : le contrôle des données, la saisie des données et la collecte des données. Les deux premiers sont présents dans tous les pots de miel, mais le troisième est surtout nécessaire pour des pots de miel distribués.

Le contrôle de données : se fait par le contrôle des activités des attaquants et la limitation de leurs pouvoirs afin de protéger le système. Cela est assuré par exemple par un pare-feu

dissimulé derrière un routeur, qui utilise plusieurs techniques comme le blocage du trafic après un nombre limité de connexions L'automatisation du contrôle de données reste, cependant, un défi important pour les développeurs et les administrateurs des pots de miel.

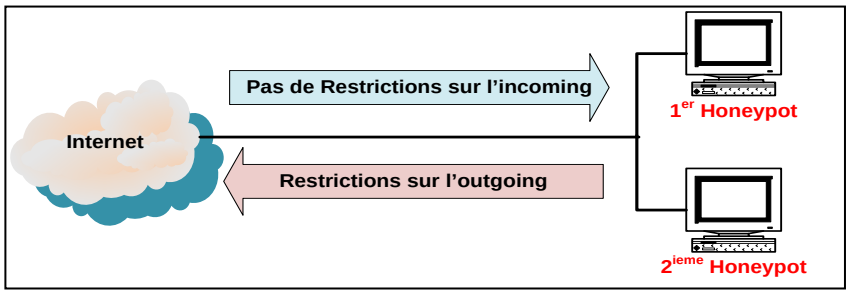


Figure 3 : Contrôle des données

La saisie des données : a pour objectif d'enregistrer les activités des attaquants sur le pot de miel sans que ceux-ci ne puissent détecter qu'ils sont sous surveillance.

Collecte de données : permet de coordonner et de centraliser l'information provenant de plusieurs pots de miel distribués.

Honeynet peut être configuré sous trois architectures différentes : 1ère génération, 2e génération et *Honeywall* :

1ère génération : son principal but est de capturer les activités du *blackhat community*, sans pour autant avoir des capacités évoluées de contrôle des attaques. Le pot de miel est isolé du réseau principal derrière un outil de contrôle d'accès (souvent un pare-feu). Tout le flux qui entre ou sort du pot de miel doit passer par le pare-feu, l'isolation ayant pour but de réduire les risques d'attaque du réseau principal. L'architecture de cette génération est une architecture multi-couches, dont le contrôle et la capture de données sont assurés par plusieurs dispositifs distincts :

Un pare-feu (première couche) limite le nombre de connexions entrantes.

Un routeur (deuxième couche) limite toute connexion vers l'internet et cache le pare-feu du réseau intérieur.

La capture et la collecte de données sont assurées par :

Un pare-feu (première couche) qui garde un journal de toutes les connexions de l'Internet et du *honeynet*.

Un système de détection d'intrusion (deuxième couche) qui garde un log de toutes les activités du réseau *honeynet*. (Il est accessible depuis le réseau de production mais pas depuis le *honeynet*).

Les pots de miel (troisième couche) qui exploitent les journaux du système (*system logs*).

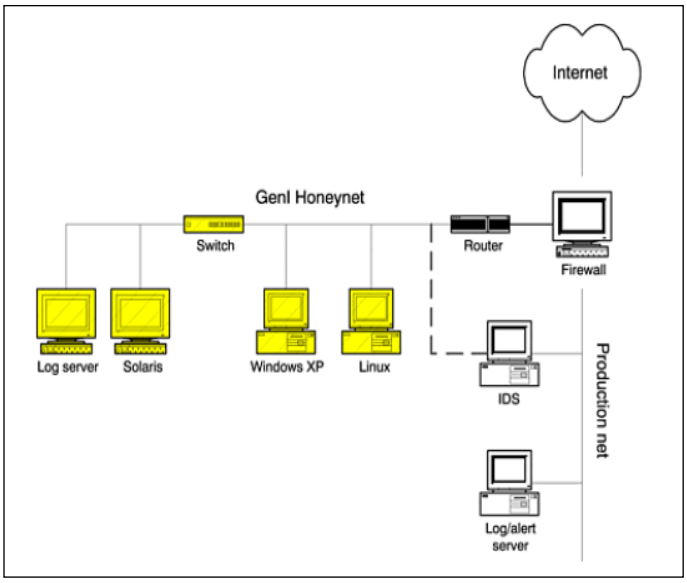


Figure 4 : Diagramme de *Honeynet 1ère génération* **[1]**

2e génération : développée à partir de 2002, elle est plus difficile à détecter. Dans l'architecture de la deuxième génération (voir la figure Figure 5), les opérations de capture, contrôle et analyse des données sont assurées par un seul dispositif appelé "*Honeywall*". Ce dispositif est configuré en mode bridge (pont), ce qui lui permet de fonctionner sur les deux couches basses du modèle OSI (*Open System Interconnection*). Il est donc difficilement détectable par les pirates, car il n'a pas d'adresse IP et il ne décrémente pas le champ TTL (Time To Live). Le but du *Honeywall* c'est le routage et le contrôle du trafic malveillant. Cette génération a permis d'améliorer les capacités des *honeynets*, mais elle reste toujours difficile à employer et à maintenir, à cause du nombre élevé de configurations requises : configuration des *Snort_Inline*, *Sebek*, *IPTables*, etc.

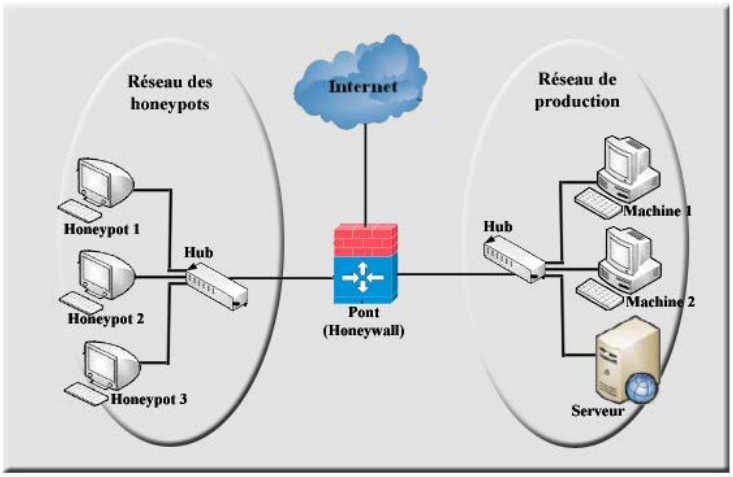


Figure 5 : Diagramme de *Honeynet 2e génération* **[4]**

3e génération : Honeynets ROO CDROM : l'architecture de la troisième génération a été conçue principalement pour régler le problème de la difficulté de déploiement, de gestion et de maintenance des *honeynets* en intégrant de nouveaux outils nécessaires pour la collecte et l'analyse de données dans un seul CD *Bootable* facile à installer et à configurer. La première version de ce CD est appelée *Eeyore*. Cette version a été améliorée ensuite à partir de 2005 en une nouvelle version appelée "Honeynet ROO CDROM". Les outils de contrôle de données dans ce CD sont les mêmes que ceux de la deuxième génération (IPTable, Snort_Inline), tandis que ceux de capture et de collecte de données sont renforcés par des nouveaux outils comme *p0f* et *Argus*. *P0f* est utilisé pour découvrir le système d'exploitation de la machine du pirate et Argus est un outil puissant dans le rapatriement d'informations sur les échanges de données sur le réseau. Argus permet de connaître l'heure de début et de fin d'une connexion, le nombre d'octets et le nombre de paquets transmis dans chaque direction (cas d'une connexion TCP bi-directionnelle). Le CDROM *honeywall* contient aussi un outil en script Perl appelé *Hflow* qui fusionne toutes les données récupérées par les outils de collecte dans une base de données centrale. L'analyse de données de cette génération a été aussi renforcée par une interface web graphique appelée *walleye* (voir Figure 6) qui peut être lancée à distance via une connexion web sécurisée (https) [4]. *Honeywall* CDROM est une application installable gratuitement qui permet de télécharger tous les outils et les ressources pour la création, la maintenance et l'analyse rapide de la troisième génération du *honeynet* [5]. Elle est facile à installer, à maintenir et elle peut être déployée sur plusieurs machines dans un réseau. Elle est basée sur la version Linux *Fedora Core 3*. Il est possible de configurer le système lors de l'installation ou d'utiliser une version pré-configurable.

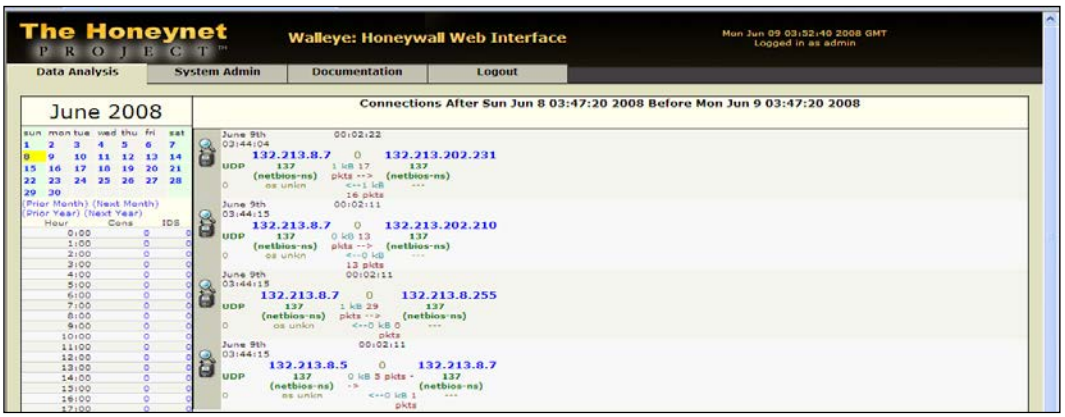


Figure 6 : Interface *Walleye* et l'analyse de données

CHAPITRE II

SYSTÈMES DE DÉTECTION D'INTRUSIONS

Les systèmes de détection des intrusions sont utilisés dans un réseau local pour détecter les attaques externes ou internes. Dans ce chapitre, nous présentons une analyse détaillée de ces systèmes, en mettant en évidence leurs forces et faiblesses pour la protection des systèmes informatiques.

II.1 Introduction

Une intrusion *est l'acte d'entrer dans un lieu sans y avoir droit et sans y être invité* [15]. La détection d'intrusion a pour but de mettre en évidence toute violation ou contournement d'un système automatique en envoyant une alerte au moment où l'intrus débute son attaque. Un système de détection d'intrusions (IDS : *Intrusions Detection System*) est un composant nécessaire et indispensable pour assurer la sécurité d'un réseau informatique. Il permet, de façon automatisée, de collecter l'information sur le comportement des utilisateurs du système et de détecter tout comportement malicieux. En général, nous distinguons deux approches pour la détection d'intrusion : l'approche par scénario où la détection des attaques se fait en comparant les activités des utilisateurs avec des scénario d'attaques déjà connues, et l'approche comportementale qui permet de détecter des comportements d'utilisateurs qui dévrillent de ce qui est considéré comme normal.

II.2 Description d'un système de détection des intrusions (IDS)

Un système de détection des intrusions est une combinaison ou intégration de plusieurs services pour assurer une meilleure gestion et contrôle de sécurité du système. La figure 7

montre un modèle d'un IDS proposé par le groupe IDWG (*Intrusion Detection exchange format Working Group*) [16]. Les éléments de ce modèle sont définis comme suit :

Alerte : est un message formaté et émis par un analyseur lorsqu'il y a des activités intrusives contre une source de données.

Analyseur : c'est un outil matériel ou logiciel qui met en œuvre l'approche choisie pour la détection (comportementale ou par scénarios), il génère des alertes lorsqu'il détecte une intrusion.

Capteur : un logiciel générant des événements en filtrant et en formatant les données brutes provenant d'une source de données.

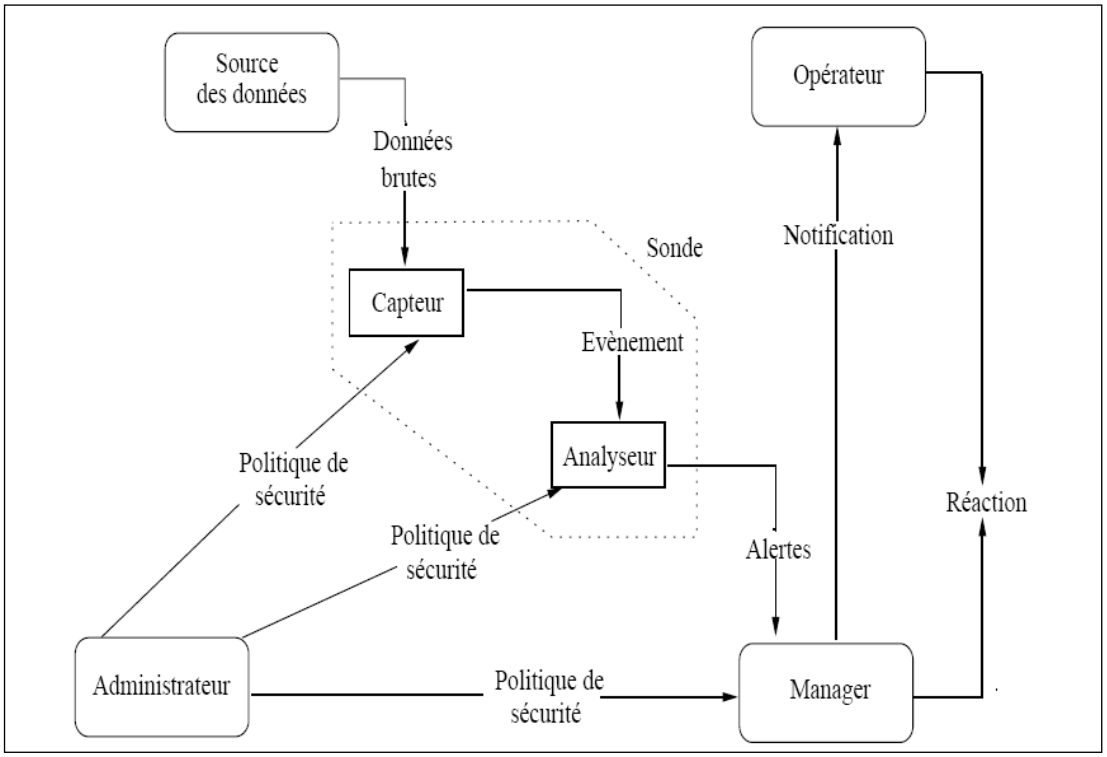


Figure 7 : Modèle général d'un IDS proposé par le groupe IDWG [15]

Événement : un message formaté et envoyé par un capteur. C'est l'unité élémentaire utilisée pour représenter un élément d'un scénario d'attaque.

Manager : composant d'un IDS permettant à l'opérateur de configurer les différents éléments d'une sonde et de gérer les alertes reçues et éventuellement la réaction de l'opérateur.

Notification : la méthode par laquelle le *manager* d'un IDS met au courant l'opérateur de l'occurrence d'une alerte.

Opérateur : une personne chargée de l'utilisation du *manager* associé à l'IDS, elle décide de la réaction à apporter en cas d'alerte, elle est parfois la même entité que l'administrateur.

Politique de sécurité : c'est la spécification des exigences de sécurité à faire respecter dans un réseau d'une organisation afin de garantir l'intégrité, la confidentialité et la disponibilité des ressources sensibles. Elle définit quelles activités sont autorisées et lesquelles sont interdites.

Réaction : sont les mesures passives ou actives prises en réponse à la détection d'une attaque, dans le but de la stopper ou pour corriger ses effets.

Sonde : un ou plusieurs capteurs couplés avec un analyseur.

Source de données : dispositif générant de l'information sur les activités des entités du système à surveiller.

II.3 Les différentes catégories d'IDS

II.3.1 Systèmes de détection des intrusions réseau (NIDS)

Un NIDS (*Network Intrusion Detection System*) est un IDS orienté réseau. Il permet de d'analyser le trafique qui circule au niveau IP (couche réseau) pour détecter d'éventuelles intrusions. Il est composé de sondes (capteurs) qui capturent le trafique acheminées sur le réseau et d'un moteur pour analyser ce trafique.

Le NIDS offre l'avantage de la furtivité et n'ajoute aucune surcharge au réseau en terme de trafique. La figure 8 montre l'architecture d'un réseau contenant un NIDS.

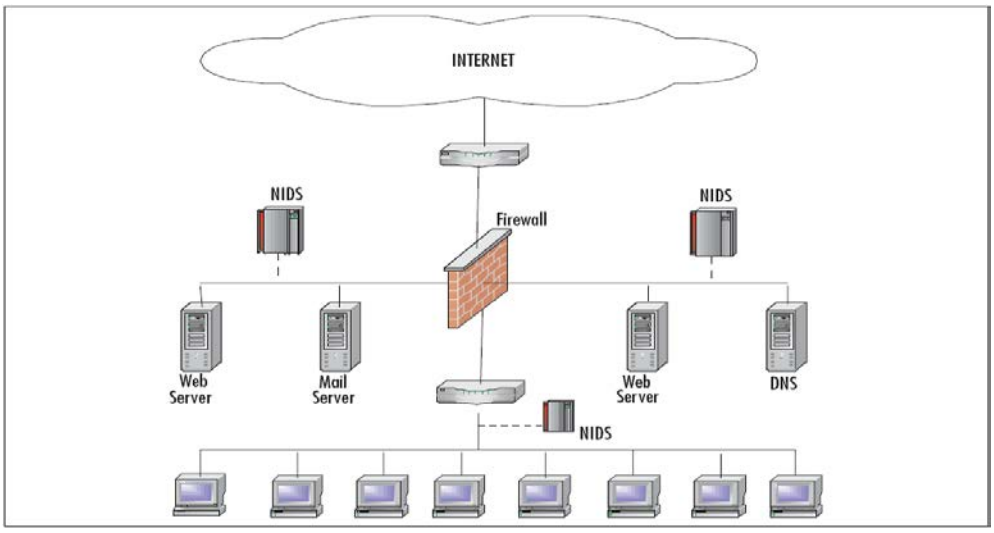


Figure 8 : Modèle d'architecture pour NIDS proposé par le groupe IDWG [15]

II.3.2 Système de détection d'intrusion hôte (HIDS)

Un HIDS (*Host Intrusion Detection System*) est un agent logiciel installé sur la machine à protéger afin d'analyser en temps réel les flux de trafique de cette machine ainsi que les fichiers journaux. Contrairement à un NIDS, un HIDS ne protège donc que le système local. Un HIDS est capable de détecter les changements dans les fichiers et dans le système d'exploitation de la machine hôte. La figure 9 montre l'utilisation de l'HIDS dans un réseau.

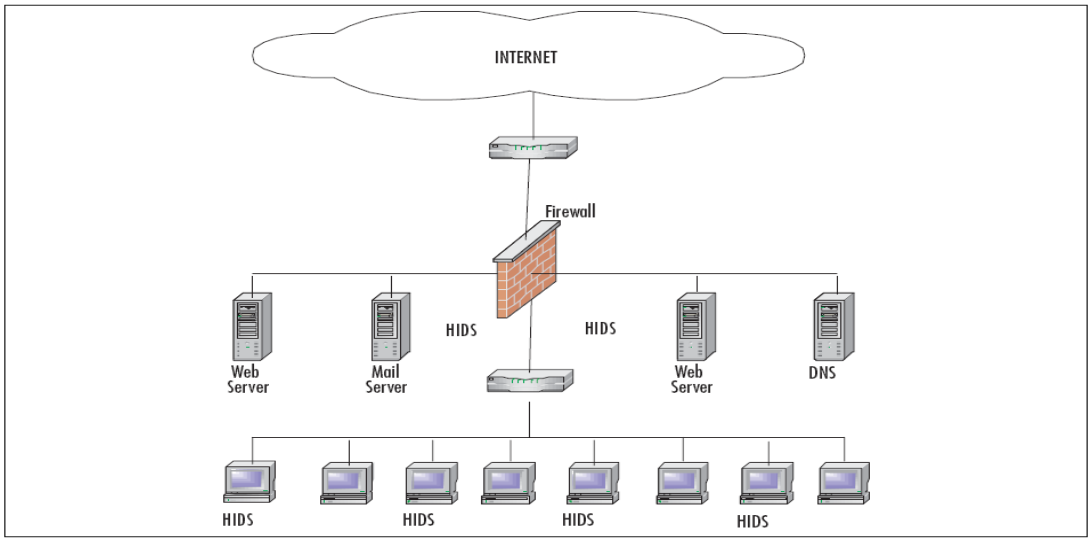


Figure 9 : Réseau HIDS [15]

II.3.3 Système de détection d'intrusion distribué (DIDS)

Un DIDS (*Distributed Intrusion Detection System*) est obtenu en faisant collaborer plusieurs types d'IDS (NIDS, HIDS, senseurs) distribués dans un même réseau local. Chacun de ces

IDS rapporte son analyse vers un même système de corrélation central. Cependant, la gestion des informations recueillies par les différents senseurs ne se traite pas de la même façon car, chaque senseur peut avoir ses propres règles. Ainsi, les alertes renvoyées vers le système de corrélation centrale sont traitées en prenant en considération les spécificités de chaque type d'alerte. La figure 10 représente une vue globale d'un DIDS.

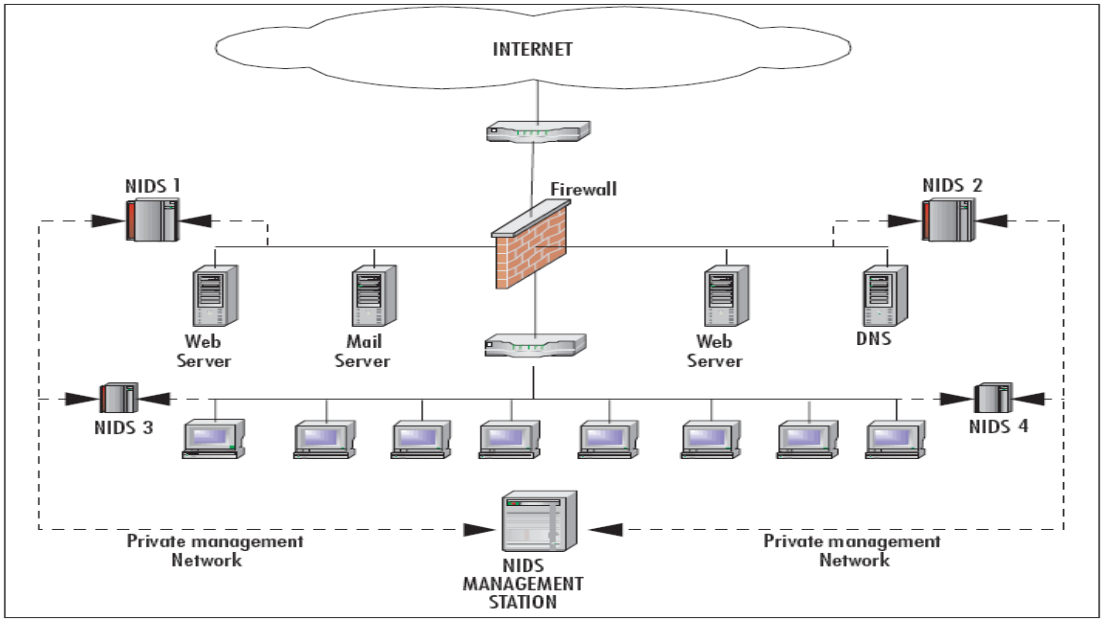


Figure 10 : Schéma global d'un DIDS [15]

II.4 Fonctionnement d'un IDS

II.4.1 Méthodes de détection

Deux stratégies majeures sont largement employées pour distinguer, quel trafic est malicieux ou non. La première stratégie se base sur le comportement passé du système pour distinguer si un trafic est bon ou mauvais (malicieux). Elle permet de détecter toutes les activités inhabituelles comparées à celles qui sont préalablement définies. Anderson dans [18] a utilisé des études statistiques pour classifier et déterminer le comportement d'un utilisateur, mais, généralement la connaissance et la distinction d'un tel comportement peuvent se font par apprentissage. Après cette phase le trafic est catégorisé en bons (de même type que celui rencontré durant la phase d'apprentissage) ou malicieux (dévie par rapport au trafic considéré comme normal).

Cependant, cette stratégie rend l'IDS tributaire de la qualité du trafic analysé durant la phase d'apprentissage et souvent la prédiction n'est pas précise et le système génère alors beaucoup de faux positifs et de faux négatifs. La deuxième technique pour la détection d'intrusion est connue sous le nom « *Rule-based / Signature-based analysis »*. Cette technique se base sur la comparaison du trafic généré par l'activité des utilisateurs dans le réseau avec une base de données de patrons de trafic connus pour être malicieux. Elle consiste à identifier seulement les scénarios d'attaque qui ont un comportement anormal dans une base de scénarios d'attaques prédéfinis. La détection par signatures est l'approche la plus utilisée dans la technologie des *IDSs* commerciaux car elle donne de meilleurs résultats comparativement à l'approche comportementale.

II.4.2 Analyse des attaques

L'analyse du trafic peut différer d'une architecture à une autre, nous pouvons distinguer deux types d'analyses : l'analyse locale centralisée et l'analyse distribuée. L'analyse peut aussi être périodique ou continue. Pour analyser un trafic, nous pouvons distinguer quatre tâches différentes :

Agrégation des données : permet de faire la collecte d'informations et la normalisation du format des données pour faciliter le traitement.

Réduction des données : permet de filtrer les données inutiles, trouver les redondances et éliminer les données erronées.

Corrélation des données : permet d'identifier les relations entre les alertes dans le but de les regrouper (*Clustering*) en fonction de différentes variables (temps, évènement, port destination, protocole, contenu du message, etc.).

Induction des données : essayer de comprendre de nouvelles données, découvrir de nouveaux patrons ou identifier des nouvelles attaques.

	Vrai	***Faux***
Positif	Une alerte est générée et une condition présente doit être révélée	Une alerte et générée et aucune condition présente n'est révélée
Négatif	Une alerte n'est pas générée et aucune condition présente n'est pas révélée	Une alerte n'est pas générée et une condition présente doit est révélée

Tableau 2: Vrai/Faux, Positif/Négatif

II.4.3 Réaction et comportement après une attaque

La qualité de la réaction d'un IDS dépend souvent des règles mises par l'administrateur lors de l'installation et de la configuration du système. Nous pouvons distinguer deux approches pour la réaction d'un IDS :

Approche passive : l'IDS ne fait aucune réaction se contentant d'informer l'administrateur par une alerte sous forme d'un courriel ou un message SMS.

Approche active : l'IDS envoie des alertes en plus d'autres réactions contre cette attaque comme, par exemple, réinitialiser la connexion, bloquer du trafic, supprimer tous les processus du système attaquant, etc.

Nous distinguons deux modes pour une alerte : le mode complet donnant une alerte plus détaillée et un mode réduit donnant quelques informations essentielles (*Full Mode Alert, Fast Mode Alert*), dans les affichages suivants nous pouvons voir les deux modes:

```
[**] [1:1913:8] RPC STATD UDP stat mon_name format string exploit attempt
[**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
11/01-04:27:16.655166 172.16.10.151:807 -> 172.16.10.200:956
UDP TTL:3 TOS:0x0 ID:0 IpLen:20 DgmLen:1104 DF
Len: 1076
[Xref => http://www.securityfocus.com/bid/1480]
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0666]
```

Affichage 2 : Alerte en mode complet. **[15]**

```
11/01-04:27:16.655166 [**] [1:1913:8] RPC STATD UDP stat mon_name format
string exploit attempt [**] [Classification: Attempted Administrator
Privilege Gain] [Priority: 1] {UDP} 172.16.10.151:807 ->
172.16.10.200:956
```

Affichage 3 : Alerte en mode réduit. **[15]**

CHAPITRE III

COLLBORATION POT DE MIEL – IDS

Ce chapitre compare les deux technologies pot de miel et IDS sur plusieurs aspects et met en évidence la synergie qui découle de leur collaboration dans une architecture de sécurité. Quelques travaux qui proposent de combiner des pots de miel et des IDS sont aussi présentés dans ce chapitre.

III.1 Introduction

Un *IDS* détecte généralement les attaques sur les systèmes à travers l'internet ou l'intranet. Comparé à un système de prévention d'intrusions (*IPS* : *Intrusion Prevention System*), il est considéré comme un dispositif passif, son rôle est de générer des d'alarmes dans le cas où l'analyse du trafique réseau en temps réel montre une tentative d'intrusion. Par contre, un IPS possède la capacité de réagir afin de mettre terme à l'intrusion ou de réparer le système après attaque. Les *IDS* peuvent être configurés pour réagir en temps différé. En effet, les descriptions d'alertes peuvent être regroupées dans un fichier log qui est alors évalué manuellement, hors ligne, par un administrateur. Inconvénient majeur, les *IDS* ne détectent que les attaques connues et décrites dans la base de signatures d'attaques. Donc, toute nouvelle attaque va rester en dehors du spectre de détection de l'IDS. L'efficacité et la puissance d'un *IDS* se mesurent, donc, par la qualité et la grosseur de sa base d'information, constituée principalement de signatures d'attaques. Malheureusement, l'évolution rapide des techniques d'attaques constitue un talon d'Achille pour les IDS en termes de mise à jour des bases d'information de l'IDS. Ce problème explique bien le taux élevé de faux négatifs dont souffrent les IDS.

Un pot de miel (*honeypot*) est un système configuré pour être ciblé, compromis ou attaqué, dont le but de capturer le maximum d'informations sur les activités des attaquants, leurs outils, leurs méthodes d'attaque et leurs comportements. De nos jours, les pots de miel constituent l'outil par excellence pour la cueillette d'informations sur les attaquants et leurs

méthodologies d'attaques. La majorité des travaux qui portent sur la collaboration des *IDS* avec les *pots de miel* se font dans le but d'améliorer la détection d'attaques par les IDS.

III.2 Pourquoi faire collaborer les pots de miel et les IDS?

La base d'information d'un *IDS* est gérée par un administrateur qui est responsable pour sa mis à jour. Ainsi, la force et la faiblesse d'un IDS sont liées au facteur humain. En effet, le processus de maintenance demande un niveau de compétence élevé de l'administrateur et des mises à jour intensives de la base d'information afin de contrecarrer les nouvelles attaques. Ces deux exigences, ne sont malheureusement pas toujours satisfaites. Par ailleurs, le temps de réponse d'un IDS pour une même attaque dépend de la manière dont les règles de détection sont écrites. Ainsi, plusieurs IDS configurés par des administrateurs différents vont réagir différemment à la même attaque.

Un pot de miel joue le rôle d'une « caméra de surveillance réseau » et permet ainsi de collecter et de fournir des informations d'une grande valeur. Ces information sont alors utilisées pour générer de manière uniforme des signatures d'attaques et d'alimenter, en temps réel, les bases d'information des IDS. Ainsi, la technologie pot de miel permet de collecter des informations sur les activités des *hackers*. Information qui est analysée, par la suite, pour créer des modèles d'attaques permettant d'enrichir la base d'information des *IDS* [19].

III.3 Nécessité de transformation du format **.pcap* au format signatures

Une capture d'un pot de miel ressemble à celle recueillie par un outil de capture du trafic comme *Wireshark* ou *Ethereal*. Ce trafic est sauvegardé sous format de fichiers **.pcap* qui n'est qu'une collection (séquence) des paquets triés par ordre chronologiques comme le montre la figure 11. Chaque paquet est formé par les éléments suivants :

No : précise l'ordre du paquet par rapport au reste du trafic.

Time : représente le temps écoulé depuis le début de la capture.

Source : l'adresse IP source.

Destination : l'adresse IP destination.

Protocole : le nom du protocole utilisé.

Info : informations détaillées sur le contenu du paquet.

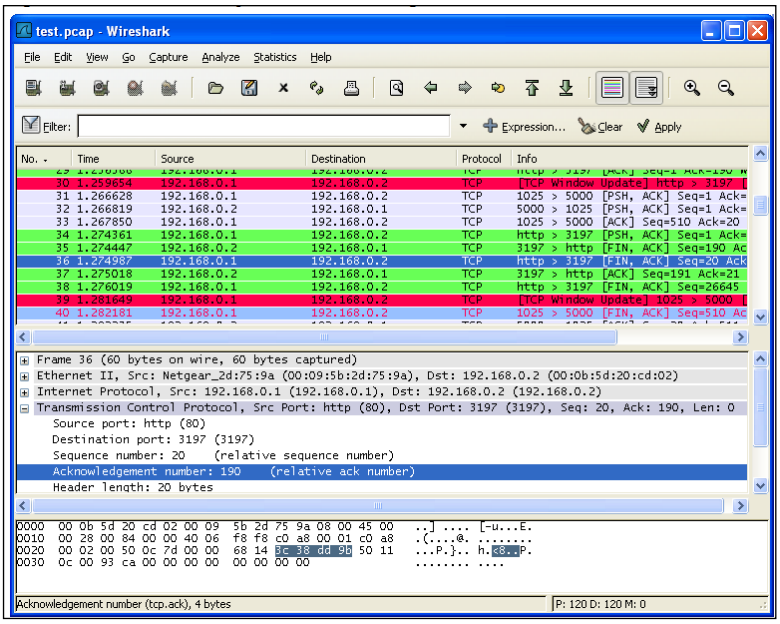


Figure 11 : Paquets recueilli par un Pot de miel (*Wireshark*)

Les paquets réseau généré par les différentes attaques ainsi que les paquets composant le trafique de service (lié au fonctionnement du réseau) sont capturés en désordre ce qui rend difficile l'extraction et le regroupement des paquets d'un même message (action élémentaire) et par la suite les messages (suite d'actions) composant une attaque. Les *IDS* ne peuvent pas prendre l'information telle quelle des fichiers logs fournis par le *honeypot*. Il faut pour cela passer par une procédure d'abstraction pour générer les règles ou les signatures d'attaques qui conviennent pour chaque type d'*IDS*. Dans le cas de notre travail, la mise en forme du trafique capturé au niveau du pot de miel est décrite en détaille dans la partie contribution.

III.4 Les signatures d'attaques

Nous définissons une attaque par la suite d'actions effectuées par un attaquant sur un système d'information. Les actions exécutées sur un système informatique peuvent faire partie d'une attaque ou constituer un trafic légitime. En général, les attaques se distinguent par un comportement anormal, c'est-à-dire, qu'elles divergent de ce qui est considéré par le système comme étant normal. Les signatures d'attaques permettent d'identifier de manière non ambigüe les attaques connues par le système. Par ailleurs, une attaque peut exploiter une ou plusieurs des vulnérabilités connues. Ces vulnérabilités sont répertoriées par des organismes tels que : CVE (*Common Vulnerabilities and Exposures*), *BugTraq*, *Nessus*, *Xforce*, etc. Part

exemple, dans la signature *Snort* de la Figure 12, nous pouvons voir deux références aux vulnérabilités : ***Nessus : 11104*** et ***Cve : CVE-2001-1020.***

Différentes attaques peuvent exploiter une vulnérabilité de différentes manières en fonction de l'objectif de l'attaquant. Ainsi, différents *IDS* peuvent regrouper les attaques sous différentes catégories, dans la figure 12, *Snort* catégorise cette signature sous le groupe ***web-application-attack*** [20]. Ici, le ***sid: 1999*** représente l'identifiant unique de la signature

```
alert tcp $EXTERNAL_NET any -> $INTERNAL_NET 80
  (msg:"WEB-PHP edit_image.php access";
   flow:established,to_server;
   uricontent:"/edit_image.php";
   reference:nessus,11104;
   reference:cve,CVE-2001-1020;
   classtype:web-application-activity;
   sid:1999;
   rev:1;)
```

Figure 12 : Signature d'attaque créée par *Snort* [20]

Par ailleurs, plusieurs approches sont utilisées pour construire une attaque et la même attaque peut avoir plusieurs formes et être composée de différentes actions élémentaires. Les IDS doivent alors tenir compte des variantes d'attaques (même attaque construite différemment ou avec une légère variation) et se mettre à jour rapidement pour contrer ces attaques. Une des techniques utilisée pour résoudre ce problème est la corrélation d'attaques qui permet de regrouper les attaques sous plusieurs catégories en fonction de plusieurs dimensions [20, 21].

III.5 Travaux connexes sur la collaboration Pots de miel / *IDS*

Dans la littérature, plusieurs travaux ont abordés la problématique de la collaboration entre la technologie des pots de miel et la technologie de détection d'intrusion. Les objectifs poursuivis par ces travaux peuvent être résumé comme suit :

- Implémenter et créer un système plus sécurisé.
- Augmenter la valeur ajoutée des pots de miel et des IDS.
- Utiliser les informations collectées par les pots de miel pour créer les scénarios d'attaques.
- Résoudre le problème de limitation des sources d'informations des IDS.

- Augmenter les connaissances des IDS.
- Réduire l'intervention humaine pour maintenir ou configurer les règles de détection d'attaques des IDS.
- Augmenter la précision des IDS : réduire le taux de *faux négatifs* et de *faux positifs*.

Les principaux travaux proposés dans ce domaine sont résumés dans ce qui suit :

A Study of Intrusion Signature Based on Honeypot [23] : ce travail se base sur la construction d'un arbre 'AND-OR' d'attaque pour modéliser une expression qui définit les attaques. La racine de l'arbre est le but de l'intrusion et le chemin de la racine vers une feuille montre la suite des composantes d'une attaque. Ce travail utilise l'algorithme *LCS (Longuest Common Substring)* pour choisir et distinguer entre les différents éléments d'une intrusion, pour créer à la fin une signature d'attaque.

A Novel Architecture for Real-time Automated Intrusion Detection Fingerprinting using Honeypot [27] : cette approche utilise un traitement automatique des données recueillies par un pot de miel et la découverte de nouvelles attaques en se basant sur la connaissance des anciennes attaques. Elle ne se limite pas à certaines couches de communication mais elle traite tous les protocoles de manière égale. La mis à jour de la base d'information d'un IDS se faire d'une manière automatique et en temps réel lors de la découverte d'une nouvelle attaque. En se basant sur les anciennes attaques, le pouvoir de découverte des nouvelles attaques reste, cependant, très limité car les nouvelles attaques n'ont souvent aucune corrélation avec les attaques déjà connues.

The EarlyBird System for Real-time Detection of Unknown Worms [25] : ce système commence par le filtrage des paquets et détermine le niveau d'alerte de chacun d'eux se basant sur des études statistiques. Les signatures d'attaques sont aussi générées en se basant sur des calculs statistiques qui se font par rapport aux connaissances précédentes. Ce système est utilisé pour étudier la réponse d'un IDS par rapports aux anomalies détectées et analyser pourquoi les fausses alertes surviennent. Mais ce travail se limite uniquement à la détection des codes malicieux (*Worms*).

Honeycomb – Creation Intrusion Detection Signatures Using Honeypots [26] : ce système utilise *honeyd* pour recueillir les informations sur les attaques, il utilise l'algorithme LCS pour générer automatiquement des signatures d'attaque.

A Method to obtain Signatures from Honeypot Data [24] : ce système est spécialisé dans la génération de signatures *Snort* et il traite, de ce fait, les protocoles IP, TCP, UDP et ICMP. Comme les données de trafic des pots de miel représentent des activités anormales par définition, les motifs (*patterns*) d'activité extraits de ces données peuvent être utilisés comme des signatures d'attaque. Dans cette approche, seuls les paquets critiques capturés au niveau du pot de miel sont choisis pour générer des signatures d'attaques. Le choix des paquets critiques se base sur l'analyse de la charge utile des paquets.

III.6 Conclusion

Comme mentionné dans ce qui précède un pot de miel seul ou un IDS seul ne peut garantir un système sécurisé. La plus part des travaux qui abordent la collaboration entre les pots de miel et les *IDS* se concentrent dans l'analyse du trafic pour générer des signatures d'attaques et par la suite enrichir les IDS avec cette base d'informations. Ces travaux se basent sur la distinction qualitative entre les signatures d'attaque. Par ailleurs, le nombre de nouvelles attaques augmente de jour en jour. La mise à jour d'un système de détection d'intrusion devient alors un grand défi car il faut alimenter les *IDS* par l'information exacte et en temps réel sur les nouvelles attaques. Donc, la création manuelle de signatures d'attaques ne répond plus aux besoins. Pour cela, il faut investir dans des systèmes automatisés permettant de détecter automatiquement les nouvelles attaques en temps réel et de renforcer automatiquement les systèmes de protection comme les IDS.

PARTIE B : CONTRIBUTION

CHAPITRE IV

ARCHITECTURE ET FONCTIONNEMENT DU MODÈLE QUANTITATIF POUR LA DÉTECTION DES INTRUSIONS

Nous proposons une nouvelle architecture collaborative entre un pot de miel et un système de détection d'intrusion. L'architecture proposée s'appuie sur un système de pot de miel afin de pouvoir collecter le trafic malicieux nécessaire pour construire notre modèle quantitatif. Ce dernier est construit en appliquant des algorithmes de classification et des techniques statistiques. Nous présentons aussi une nouvelle approche pour la génération automatique des scénarios d'attaque à partir des données collectées par un pot de miel. Cette approche est basée sur le principe de distinction des différents éléments d'une attaque et la génération des séries d'actions élémentaires à partir d'un ensemble de paquets TCP/IP capturés en temps réel.

IV.1 Introduction

Jusqu'à présent la technologie des pots de miel (*honeypot)* est considérée comme l'outil le plus efficace pour collecter des informations détaillées sur les attaques des pirates. Un serveur pot de miel est toujours configuré de façon à attirer l'intention des pirates tout en n'étant pas en production. De ce fait, n'importe quel trafic entrant dans un pot de miel est considéré comme un trafic suspect et souvent malicieux. D'où l'idée qu'un pot de miel peut être utilisé pour améliorer nos connaissances sur les attaques des systèmes informatiques. Cependant, pour sécuriser un système informatique, un pot de miel seul n'est pas suffisant et doit nécessairement se renforcer avec d'autres technologies complémentaires comme celle offerte

par les systèmes de détection d'intrusion. Nous proposons donc une architecture qui combine deux technologies : un pot de miel et un système de détection des intrusions comme solution pour sécuriser de manière plus efficace les systèmes informatiques. Notre solution permet d'enrichir la base de connaissance d'un IDS en construisant un modèle probabiliste d'attaques qui est alimenté par le trafic enregistré au niveau du *honeypot.* Le modèle probabiliste est un modèle quantitatif de description de scénarios d'attaques qui complémente le modèle qualitatif utilisé dans les technologies IDS actuelles. En effet, pour détecter des intrusions, un IDS compare le trafic réseau avec une base de signatures d'attaques préétablie, à chaque fois qu'une correspondance est trouvée, l'IDS lance une alerte. Cependant, la construction de la base de signature d'attaques reste encore, de nos jours, un principal défi car il repose principalement sur des analyses manuelle sur du trafic réseau. Ainsi, l'automatisation du processus d'analyse et de génération des scénarios d'attaque est devenue un objectif primordial et plusieurs chercheurs, ont investis ce domaine extrêmement prometteur. Dans [26], *Christian Kreibich* et *Jon Crowcroft* présentent l'outil *honeycomb* permettant de générer des signatures d'attaque pour les systèmes de détection d'intrusion, le principe de cet outil se base sur la concaténation des messages envoyés par l'attaquant pour construire une signature unique. Cependant, ce travail est malheureusement limité à quelques systèmes de détection d'intrusion comme *Bro* [34] et *Snort*[15]. Par ailleurs, plusieurs travaux proposent des modèles de classification du trafic capturé au niveau du réseau. Dans [35], *Cohen* a proposé un système appelé *RIPPER* dans lequel le trafic réseau est analysé en appliquant des motifs et des règles d'association. *Lee* dans [36] a appliqué le travail de *Cohen* pour construire un IDS. De plus, plusieurs travaux ont exploré des approches par algorithmes génétiques [37, 38, 39, 40, 41] alors que d'autres travaux sont basés sur la logique floue [42, 43]. Malheureusement, aucune de ces techniques ne résout complètement le problème. Par exemple, plusieurs de ces techniques se contentent uniquement de classifier le trafic afin de simplifier la tâche de l'analyseur humain pour qu'il ne concentre son analyse que sur le trafic suspect (apprentissage semi-automatique).

L'approche que nous proposons pour la génération automatique des scénarios d'attaque, apporte des idées qui sont, à notre connaissance, complètement nouvelles et totalement différente de toutes les solutions proposées jusqu'à présent, cette approche se base sur l'analyse de tous les paquets TCP/IP capturés par un pot de miel lors d'une attaque réseau, et d'en extraire une série d'actions élémentaires qui représentent toutes les étapes exécutées par

l'attaquant contre une victime. Une action élémentaire dans notre approche représente un message : une requête ou une réponse envoyée ou reçue par l'attaquant. Ce message est généralement véhiculé sur le réseau par un ou plusieurs paquets TCP/IP selon sa taille et selon l'état courant du réseau traversé. Nous avons développé un module appelé *HoneyLens* qui permet, entre autres, de reconstruire des messages représentant des actions élémentaires à partir des paquets qui circulent sur le réseau. Par la suite, les messages sont regroupés pour reconstruire les attaques come une suite bien définie de messages. Ces dernières sont structurés sous forme arborescente et sauvegardées dans un fichier XML. Ceci permet d'y appliquer plus facilement des services de fouille de données (*Data Mining*). La figure 13 illustre une vue globale de l'architecture proposée.

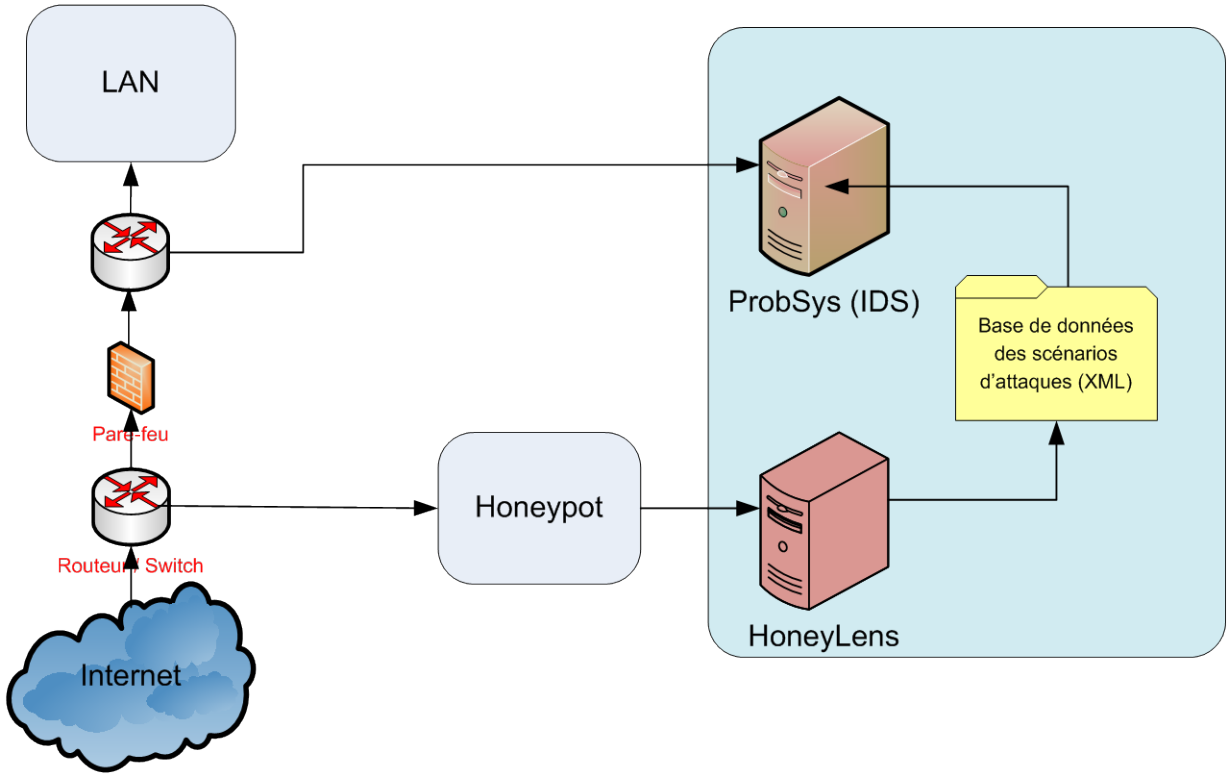


Figure 13 : Architecture globale de notre système

IV.2 Objectifs de recherche

Notre approche a pour objectif de résoudre quelques problèmes relatifs à l'automatisation de l'analyse de trafic et à la réduction du taux de fausses alertes dans les IDS, nous proposons donc d'automatiser les tâches suivantes :

- l'analyse du trafic : les défis qui accompagnent cette tâche sont la gestion d'une très grande quantité d'information et la classification du trafic.

- automatiser le processus de mise à jour de la base d'information des IDS
- Utiliser un modèle statistique prédictif pour réduire le taux de fausses alertes généré par un IDS.

IV.3 Architecture et composants de notre système

L'architecture proposée, comme le montre la figure 14, est constituée de trois parties principales. La première partie est la zone du *honeypot* qui s'installe à l'entrée du réseau locale. La deuxième constitue le cœur de notre système et est formé par les modules *HoneyLens* et le *ProbSys*. Ces deux modules permettent de protéger de manière complètement automatique et autonome le réseau local.

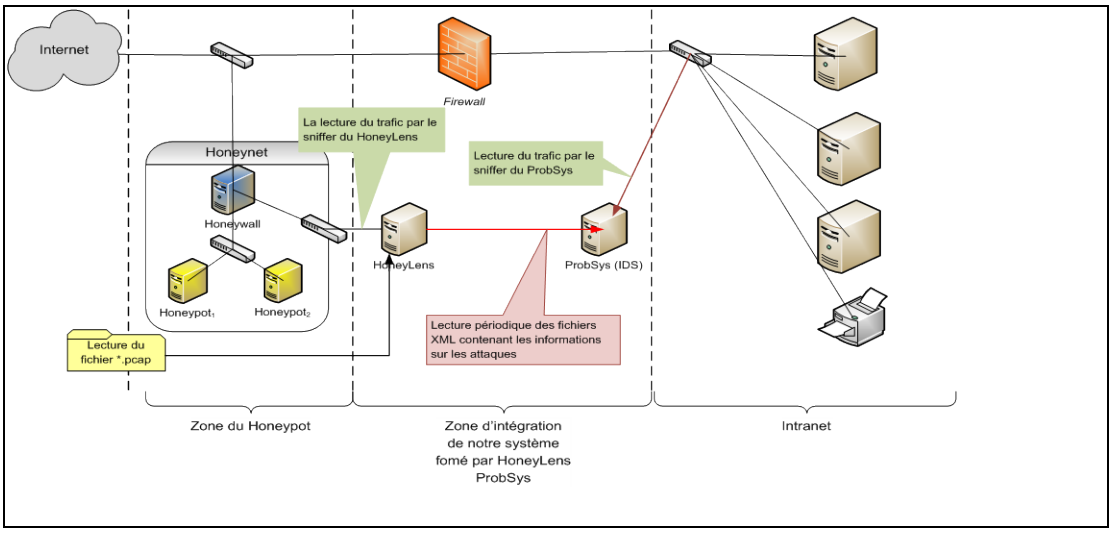


Figure 14 : L'architecture détaillée de notre système

Le module HoneyLens : ce module permet de capturer et d'analyser le trafic entrant ou sortant depuis et vers le pot de miel. L'analyse de ce trafic permet alors de construire un modèle d'attaque quantitatif sauvegardé dans un fichier en format XML. Le fichier est importé périodiquement par le *ProbSys* pour alimenter sa base d'informations.

Le module ProbSys : ce module constitue un système de détection d'intrusion, il permet d'analyser le trafic entrant sur le réseau local et de détecter d'éventuelles intrusions en se basant sur les données fournies par *HoneyLens*. Cet IDS se base sur un modèle d'attaque probabiliste pour prédire les prochaines actions élémentaires formant l'attaque en cours.

IV.4 Analyse du trafic TCP/IP et génération des scénarios d'attaques

Notre système permet l'analyse des paquets TCP/IP capturés au niveau *honeypot*, afin d'extraire une série d'actions élémentaires qui représentent toutes les étapes exécutées par l'attaquant. Dans cette approche, nous définissons ***une action élémentaire*** (aussi appelée message) comme étant une requête ou une réponse TCP envoyé ou reçu sur le réseau. L'action peut être véhiculée dans le réseau par un ou plusieurs paquets TCP/IP selon sa taille et selon l'état courant du réseau traversé.

Cependant, afin de pouvoir extraire une action élémentaire à partir d'un ou plusieurs paquets réseau, il faut bien comprendre le mécanisme et le fonctionnement des protocoles de communication réseaux. En général, nous pouvons distinguer deux classes des paquets qui traversent les réseaux :

- **La classe des paquets de gestion et de contrôle des réseaux :**

Cette classe regroupe tous les paquets qui ne portent aucune donnée à transmettre (pas de charge utile) comme, par exemple, les paquets de résolution d'adresses MAC, les paquets ICMP, les paquets d'ouverture et de fermeture de connexion TCP (les paquets SYN, SYN/ACK, FIN, etc.), les paquets d'acquittements de réception, etc. Pour cette classe, chaque paquet porte une information assurant une telle tâche de contrôle ou de gestion dans le réseau, la taille de ce type de paquets est généralement réduite et indécomposable, donc on peut facilement représenter chaque paquet de cette classe par une action élémentaire.

- **La classe des paquets de transfert de données :**

Cette classe regroupe tous les paquets qui portent des données utilisateurs à transmettre d'une application source vers une application destination, les paquets de cette classe sont tous basés, dans le cas des réseaux TCP/IP, sur les deux protocoles de transport : ***TCP*** (dans le cas d'un transfert fiable) et ***UDP*** (dans le cas d'un transfert non fiable). Pour cette classe, chaque paquet porte une partie (un segment) de la donnée à transmettre. Le nombre de paquets utilisés pour transmettre une telle donnée dépend principalement de la taille totale de cette dernière ainsi que de l'architecture et de l'état courant du réseau traversé par ces paquets. La représentation de chaque paquet par une action élémentaire n'est pas adéquate à cause des problèmes résultants de la dynamicité des protocoles réseau qu'on peut résumer dans les points suivants :

1. La taille des segments de données portés dans les paquets de cette classe est dynamique, elle dépend de la segmentation faite par le protocole TCP et par la fragmentation du système de routage. Le nombre de paquets générés pour la même donnée va être dynamique et varie d'une connexion à une autre. Donc, si on prend chaque paquet comme action élémentaire, pour le transfert de la même donnée, nous risquons d'obtenir des scénarios différents pour la même attaque exécutée à des sessions différentes.

2. Dans le cas d'un transfert basé sur le protocole TCP, le nombre de paquets envoyés successivement sans attendre la réception d'un acquittement est dynamique, il dépend de la taille de la fenêtre de contrôle de flux qui est elle-même dynamique et dépend de la taille du tampon du récepteur. Supposons que la taille du tampon de récepteur peut recevoir 3 paquets alors la valeur de la fenêtre est de 3*(taille d'un paquet en octets); dans ce cas, l'émetteur va envoyer 3 paquets et attendre la réception de l'acquittement, sinon (à l'expiration du temporisateur) il suppose que les paquets sont perdus et il les retransmet de nouveau. Cependant, si on prend chaque paquet comme action élémentaire, pour le transfert de la même donnée, nous allons avoir un scénario différent à chaque changement de la valeur de la fenêtre et à chaque perte d'un ou plusieurs paquets, c'est-à-dire à chaque retransmission d'un paquet perdu.

3. Le contrôle de congestion dépend de l'état courant du réseau (taux d'occupation du réseau), pour assurer cette fonctionnalité le protocole TCP utilise l'algorithme «*Slow-start and collision avoidance*», c'est-à-dire lorsqu'il détecte une congestion (une perte d'un ou plusieurs paquets) il recommence la retransmission d'un seul paquet et à chaque réception d'un acquittement il double le nombre de paquets retransmis jusqu'à l'arrivée à la valeur de la fenêtre, donc on ne peut pas prendre chaque paquet comme action élémentaire.

4. Le Réseau TCP/IP renvoi des paquets ICMP lorsqu'il y a des problèmes de transmission, ces paquets d'erreurs (ICMP) vont influer directement le scénario qu'on va générer. Il faut donc en tenir compte.

5. Lorsqu'un paquet vient de traverser un réseau de plus petite taille, le protocole IP va fragmenter ce paquet pour adapter sa taille à la taille de ce réseau, cette action

entraine une génération d'un nombre dynamique de paquets, ce qui rend la représentation des actions élémentaires par un paquet inadéquate.

Un autre problème assez gênant pour la génération des actions élémentaires réside particulièrement dans le nombre assez énorme de protocoles de la couche application qui peuvent être la cible des différentes attaques, ce qui oblige notre générateur de scénarios à prendre en considération les détails de tous ces protocoles, que ce soit ceux qui sont standards comme HTTP, FTP, etc. ou ceux qui ne le sont pas. Donc, le système doit analyser le contenu des paquets afin de générer les scénarios d'attaques appropriés. Autrement, nous risquons de générer les mêmes scénarios pour une famille d'attaques destinées vers le même service d'application (vers le même numéro de port). On peut donc aisément imaginer la difficulté de la tâche de programmer un générateur de scénarios maitrisant tous les protocoles existants.

IV.4.1 Principe de génération des éléments d'attaque : actions élémentaires et scénarios d'attaque

Dans ce travail nous considérons définitions suivantes :

scénario d'attaque : est formé par les actions exécutés durant une session de connexion entre deux adresses IP, une IP externe et une IP interne (source et destination). Un scénario d'attaque peut aussi être composé de plusieurs sous-scénarios d'attaque.

sous-scénario d'attaque : est formé par une séquence d'actions élémentaires. C'est aussi une session de connexion entre une adresse IP externe à partir **d'un ou plusieurs ports sources choisis aléatoirement par l'attaquant** et une IP destination avec un **port cible bien choisi**, le numéro de port de la machine victime montre la nature du service attaqué.

Action élémentaire : est formée par un ou plusieurs paquets formant un message et qui utilisent le même protocole pour attaquer le même port. On note que, souvent, tous les scénarios d'attaque effectués par le même attaquant sont liés, puisque pour atteindre un objectif d'attaque, l'attaquant doit, souvent, exécuter plusieurs scénarios d'attaques. Par exemple, dans le cas où l'attaquant veut exploiter une vulnérabilité, il commence par une énumération « scan » du réseau pour savoir les identités des machines et le type des systèmes d'exploitation (premier scénario d'attaque : S1). Ensuite, il choisit une machine cible et fait un autre scan local pour identifier les serveurs qui sont actifs (deuxième scénario : S2). Enfin, il choisit un certain service pour lancer son attaque (troisième scénario : S3). Dans cet exemple,

l'attaquant a effectué trois scénarios d'attaque différents (S1, S2 et S3), ces scénarios sont nécessaires pour qu'il complète son objectif. Presque la majorité des attaques (surtout celles qui viennent du même réseau) commencent par les attaques de scan. De cette manière, si on détecte un scénario, on peut prédire le suivant avec une certaine probabilité.

IV.4.1.1 Principe de base de notre approche

Dans notre approche, un scénario d'attaque est constitué principalement d'une suite de sous-scénarios, ces derniers sont constitués d'une séquence d'actions élémentaires. Pour générer ces actions élémentaires, notre système fonctionne comme le montre la figure 15.

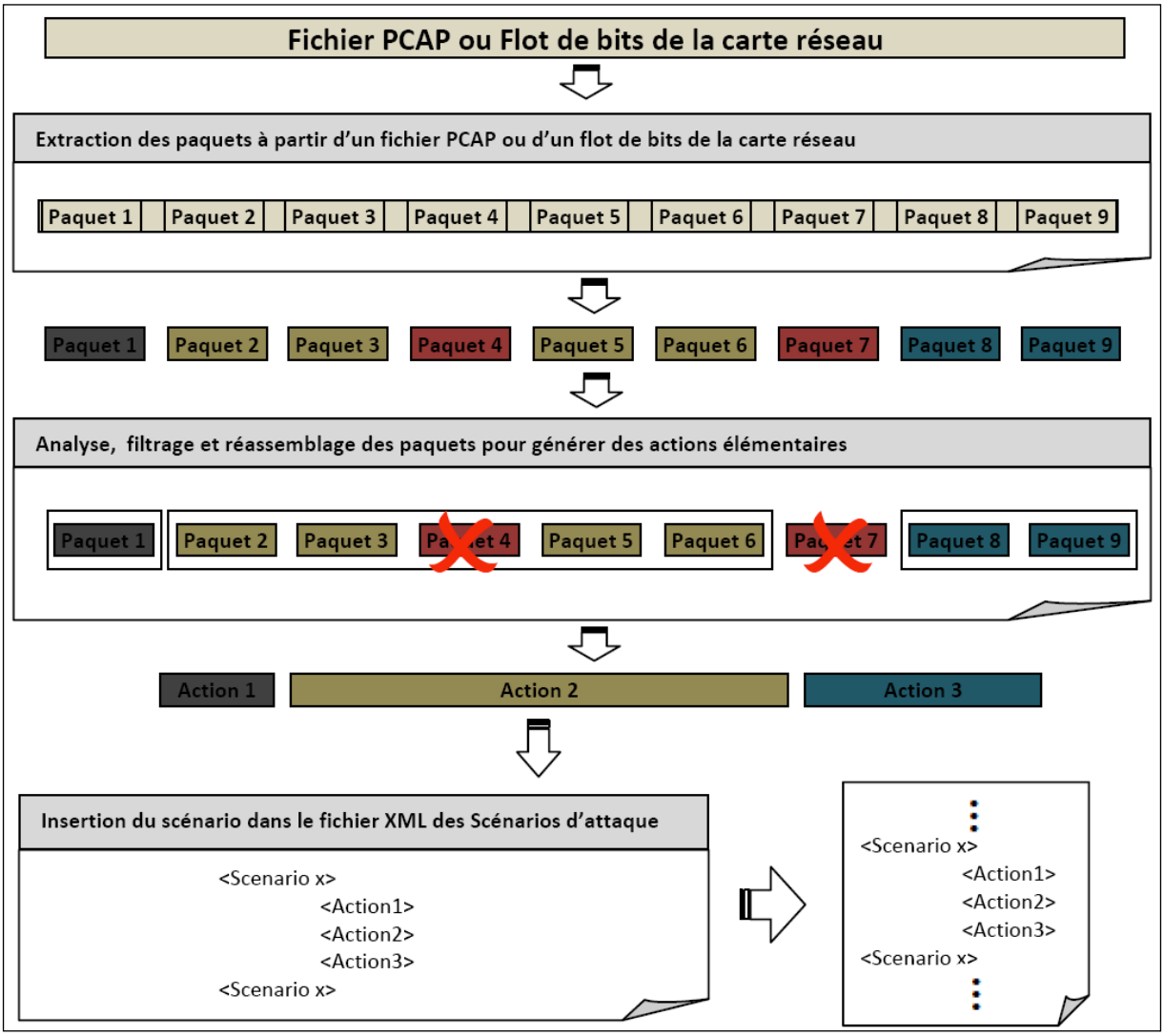


Figure 15 : Principe de fonctionnement de notre approche

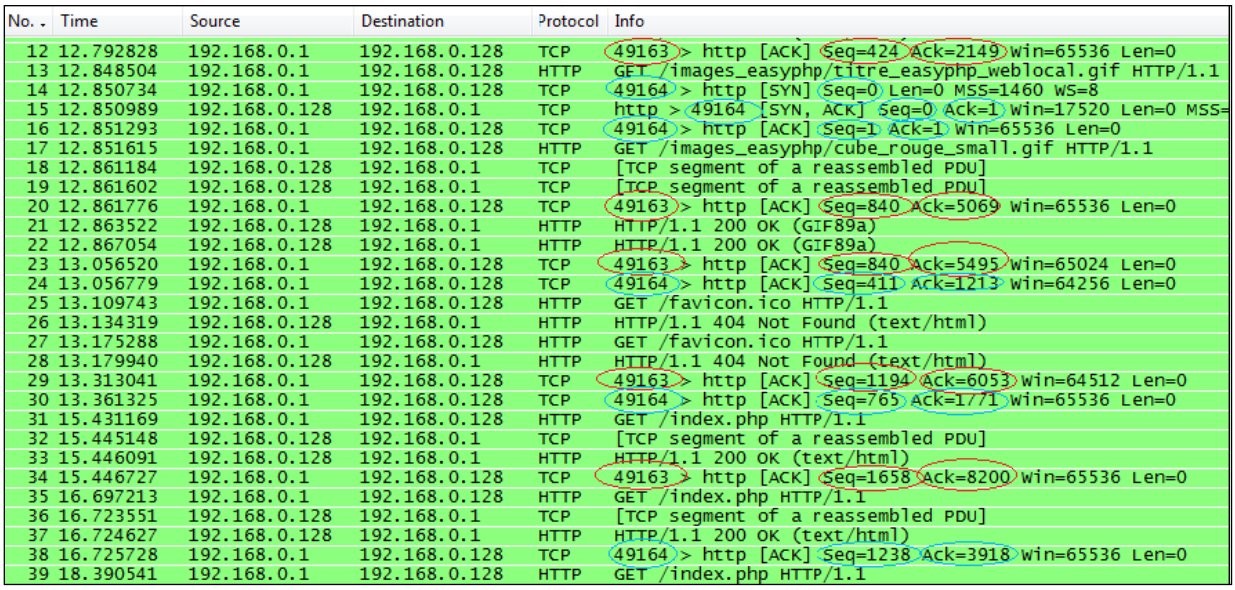
Notre système capture les informations du trafic par la lecture d'un fichier PCAP (une capture de trafic) ou un flot de bits de la carte réseau qui relie nos *pots de miel* avec le monde externe

(Internet). Par la suite, nous traitons ces paquets pour en extraire ceux qui sont utiles à la construction d'une action élémentaire et ceux qui ne le sont pas comme les paquets de service en opérant un ***filtrage***. Ces derniers vont être automatiquement éliminés alors que les autres qui sont censés représenter les éléments d'attaque vont être regroupés en un ou plusieurs paquets et chacun de ces groupes représente une action élémentaire. Cette étape est réalisée par une opération de ***réassemblage*** de paquets. Enfin, les actions élémentaires résultantes qui représentent un scénario d'attaque ou un sous-scénario vont être sauvegardées dans des fichiers XML qui facilitent la lecture et la décomposition de chaque catégorie pour former une base d'informations pour le système probabiliste préventif. Supposons qu'un scénario d'attaque est détecté n fois, le *HoneyLens* ne va pas ajouter à chaque fois ces informations dans ces fichiers XML, mais il va simplement faire le mis à jour de la valeur de la fréquence (nombre de fois que ce scénario est vu), cette idée a deux avantages importantes : avoir des tailles réduites de ces fichiers et de représenter explicitement les fréquences d'apparition des scénarios (sous-scénarios) pour réaliser des calculs probabilistes par le module *ProbSys*.

IV.4.1.2 Techniques d'analyse, de filtrage et de réassemblage des paquets

IV.4.1.2.1 Distinction entre plusieurs attaques interférées

Aujourd'hui, la plupart des adresses IP sont réservées, surtout celles de la classe B et C. Dans le cas des réseaux privés, plusieurs machines partagent la même adresse IP (par défaut l'adresse IP externe de la passerelle des réseaux), c'est le cas de la plupart des réseaux des universités, des fournisseurs d'accès, etc. Si deux ou plusieurs attaquants lancent une ou plusieurs attaques vers la même machine cible à partir de deux ou plusieurs machines du même réseau privé, tous les paquets de ces attaques vont porter la même adresse IP source qui sera celle de la passerelle du réseau privé. La même chose peut arriver si le même attaquant lance plusieurs attaques vers la même machine cible en même temps (en parallèle). Dans ce type de situations, nous allons avoir une interférence des attaques. On ne doit pas représenter toutes ces attaques par un seul scénario d'attaque même celles qui portent les mêmes adresses IP source et destination, mais il faut les séparer dans des scénarios différents en se basant sur le numéro de séquence et celui du fragment. Pour montrer ce problème, voyons la figure 16.



No.	Time	Source	Destination	Protocol	Info
12	12.792828	192.168.0.1	192.168.0.128	TCP	49163 > http [ACK] Seq=424 Ack=2149 Win=65536 Len=0
13	12.848504	192.168.0.1	192.168.0.128	HTTP	GET /images_easyphp/titre_easyphp_weblocal.gif HTTP/1.1
14	12.850734	192.168.0.1	192.168.0.128	TCP	49164 > http [SYN] Seq=0 Len=0 MSS=1460 WS=8
15	12.850989	192.168.0.128	192.168.0.1	TCP	http > 49164 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=
16	12.851293	192.168.0.1	192.168.0.128	TCP	49164 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
17	12.851615	192.168.0.1	192.168.0.128	HTTP	GET /images_easyphp/cube_rouge_small.gif HTTP/1.1
18	12.861184	192.168.0.128	192.168.0.1	TCP	[TCP segment of a reassembled PDU]
19	12.861602	192.168.0.128	192.168.0.1	TCP	[TCP segment of a reassembled PDU]
20	12.861776	192.168.0.1	192.168.0.128	TCP	49163 > http [ACK] Seq=840 Ack=5069 Win=65536 Len=0
21	12.863522	192.168.0.128	192.168.0.1	HTTP	HTTP/1.1 200 OK (GIF89a)
22	12.867054	192.168.0.128	192.168.0.1	HTTP	HTTP/1.1 200 OK (GIF89a)
23	13.056520	192.168.0.1	192.168.0.128	TCP	49163 > http [ACK] Seq=840 Ack=5495 Win=65024 Len=0
24	13.056779	192.168.0.1	192.168.0.128	TCP	49164 > http [ACK] Seq=411 Ack=1213 Win=64256 Len=0
25	13.109743	192.168.0.1	192.168.0.128	HTTP	GET /favicon.ico HTTP/1.1
26	13.134319	192.168.0.128	192.168.0.1	HTTP	HTTP/1.1 404 Not Found (text/html)
27	13.175288	192.168.0.1	192.168.0.128	HTTP	GET /favicon.ico HTTP/1.1
28	13.179940	192.168.0.128	192.168.0.1	HTTP	HTTP/1.1 404 Not Found (text/html)
29	13.313041	192.168.0.1	192.168.0.128	TCP	49163 > http [ACK] Seq=1194 Ack=6053 Win=64512 Len=0
30	13.361325	192.168.0.1	192.168.0.128	TCP	49164 > http [ACK] Seq=765 Ack=1771 Win=65536 Len=0
31	15.431169	192.168.0.1	192.168.0.128	HTTP	GET /index.php HTTP/1.1
32	15.445148	192.168.0.128	192.168.0.1	TCP	[TCP segment of a reassembled PDU]
33	15.446091	192.168.0.128	192.168.0.1	HTTP	HTTP/1.1 200 OK (text/html)
34	15.446727	192.168.0.1	192.168.0.128	TCP	49163 > http [ACK] Seq=1658 Ack=8200 Win=65536 Len=0
35	16.697213	192.168.0.1	192.168.0.128	HTTP	GET /index.php HTTP/1.1
36	16.723551	192.168.0.128	192.168.0.1	TCP	[TCP segment of a reassembled PDU]
37	16.724627	192.168.0.128	192.168.0.1	HTTP	HTTP/1.1 200 OK (text/html)
38	16.725728	192.168.0.1	192.168.0.128	TCP	49164 > http [ACK] Seq=1238 Ack=3918 Win=65536 Len=0
39	18.390541	192.168.0.1	192.168.0.128	HTTP	GET /index.php HTTP/1.1

Figure 16 : Interférence entre deux attaques de même adresses source et destination

Cette figure représente une suite de paquets pour deux attaques différentes provenant de la même machine source malgré que tous les paquets portent les mêmes adresses IP source et destination, il y a une interférence pour deux attaques différentes. Pour distinguer entre ces deux attaques, il suffit de voir les ports source et destination, aussi bien les numéros de séquence et d'acquittement de ces paquets. Dans cette figure, le premier paquet (port source = 49163) est un paquet de la première attaque, alors que le troisième paquet (port source = 49164) est un paquet de la deuxième attaque. Les ports sources de ces deux paquets sont différents, donc ils proviennent de deux applications différentes de la même machine source.

IV.4.1.2.2 Réassemblage des données segmentées

Dans le cas des paquets de transfert de données, notre générateur de scénarios *HoneyLens* doit réassembler toutes les charges utiles des paquets constituants, le message envoyé pour construire une action élémentaire, ce processus est d'ailleurs encore assez difficile, car il n'est pas évident de déterminer à quel moment une séquence de segments devra constituer un message. En effet, attendre la fin de connexion pour construire le message n'est pas réalisable, étant donné que certaines connexions peuvent nécessiter une énorme quantité de données (comme dans le cas des transferts de fichiers). Il faudra donc nous baser sur une décision arbitraire pour réassembler ces segments, afin d'en construire des actions élémentaires indécomposables. La solution que nous avons adoptée pour contourner ce problème se base principalement sur le principe des architectures client/serveur, où pour chaque requête client,

il y'a une réponse du serveur. En se basant sur ce principe, *HoneyLens* va attendre la réception de la réponse du serveur - pour déterminer le dernier segment constituant le message de requête envoyé par le client (l'attaquant) - et la nouvelle requête de ce dernier pour déterminer le dernier segment constituant le message de réponse envoyé par le serveur.

IV.4.1.2.3 Réassemblage des paquets fragmentés

Au contraire des segments des protocoles TCP, le réassemblage des fragments du protocole IP est relativement simple, il suffit de tester la valeur du bit MF (*More Fragment*) et celle du champ « *Fragment Offset* » pour déterminer les limites et l'ordre des fragments constituants un message.

IV.4.1.2.4 Filtrage des paquets inutiles

Lors du transfert d'un message, le récepteur confirme la réception d'un ensemble de parties de ce dernier par l'envoi d'un certain nombre de paquets d'acquittements qui sont utilisés pour assurer le contrôle de flux, le nombre de ces paquets est dynamique, il se change d'une connexion à une autre selon l'état courant du tampon du récepteur. Dans notre approche, ce type de paquets sera éliminé (filtré) puisqu'il ne porte aucune information supplémentaire pour une action élémentaire. La même chose pour les paquets ICMP renvoyés par les routeurs lorsqu'il y a des problèmes de routages, ainsi que les paquets retransmis à cause des erreurs de transmission (perte, altération, etc.).

IV.4.2 Structure des actions élémentaires

IV.4.2.1 Les composants d'une action élémentaire

L'action élémentaire est constituée - comme le montre la figure 17 - d'un ensemble de champs portant des informations nécessaires pour l'identifier, elle ne doit pas contenir des informations dynamiques qui se changent d'une attaque à une autre, comme les adresses IP source et destination, ports source, etc.

Code	Définition	Acteur	Protocole	Port	Signature

Figure 17 : Structure d'une action élémentaire**.**

Ces différents champs sont décrits comme suit:

- ***Code :*** est unique en identifiant l'action élémentaire.

- ***Définition (définition de l'action) :*** ce champ est utilisé pour décrire l'action élémentaire.
- ***Protocole (Protocole de la dernière couche de gestion de réseau de l'action) :*** le protocole de la dernière couche réseau utilisée dans l'action parmi les trois couches : Liaison, Réseau et Transport. Donc, les protocoles concernés par ce champ sont : ARP, RARP, IP, ICMP, TCP et UDP.
- ***Acteur (l'acteur de l'action) :*** soit qu'il est un Attaquant ou une Victime.
- ***Port (Port de destination de l'action) :*** c'est ce port qui va déterminer le service de la couche application attaquée par l'intrus. En effet, on n'a pas besoin de définir le port source de l'action puisque ce port est généralement choisi aléatoirement par la machine source de l'attaquant, c'est-à-dire pour le même type d'action on peut avoir un port source différent pour chaque attaque, la même chose pour les adresses IP sources et destinations, ils n'ont aucun sens dans une action élémentaire, puisque ces deux adresses se changent pour chaque nouvelle attaque même s'il s'agit du même type d'action.
- ***Signature (Signature de la charge utile du paquet de l'action) :*** ce champ joue un rôle très important dans la détermination de l'action élémentaire, il va contenir les données brutes du protocole de la couche application. En effet, il y a un très grand nombre de protocoles d'applications parmi ceux qui sont standards comme ***HTTP***, ***Telnet***, ***FTP***, etc., et ceux qui ne le sont pas. Donc, il est assez difficile d'analyser les paquets de tous ces protocoles pour générer des scénarios d'attaque. Pour résoudre ce problème, on prend la charge du protocole telle qu'elle est, et on la met dans le champ «Signature de la charge du paquet de l'action».

IV.4.3 Problème de similarité des signatures des actions élémentaires

Il peut arriver que la même action élémentaire puisse avoir plusieurs signatures différentes, ce problème arrive le plus souvent lorsque la charge utile du paquet contient des informations dynamiques sur la machine source ou destination comme l'adresse IP, l'URL, etc. Donc, si on change l'adresse on va avoir une autre signature alors qu'il s'agit de la même action élémentaire. On peut voir ce problème par exemple dans le protocole DNS, c'est la même action (action de résolution d'un nom) avec des adresses de résolutions différentes c'est-à-dire

avec un contenu différent. Pour résoudre ce problème, nous proposons une technique de décision basée sur ***le calcul du pourcentage de changement dans la signature*** (***calcule de similarité entre les signatures des deux actions***). Pour calculer la similarité de deux signatures Sig_1 et Sig_2, nous devons tout d'abord extraire les parties identiques et celles différentes dans les deux signatures, dont chaque partie est repérée par le numéro de son premier octet dans la signature. Les parties identiques dans les deux signatures sont toujours de même taille deux à deux, alors que les parties différentes dans les deux signatures peuvent avoir des tailles différentes, ce qui résulte un décalage dans les repères des parties identiques, comme le montre l'exemple de la figure 18.

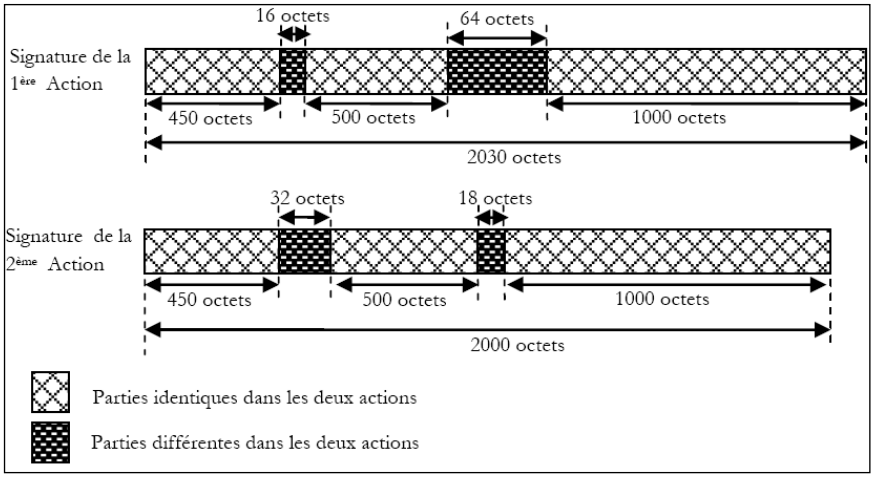


Figure 18 : Exemple de deux signatures différentes pour une même action élémentaire

Cette figure nous montre deux signatures différentes pour deux actions élémentaires candidates représentant la même action élémentaire. Ces deux signatures portent deux parties différentes contenants deux informations dynamiques par exemple le nom de la machine en question et le nom de l'utilisateur, ces deux informations ont des tailles dynamiques. Par exemple : si le nom donné à la machine de la première signature est : «*MyHomeComputer*» (14 octets) et celui donné à la machine de la deuxième signature est «*Top*» (3 octets). Pour décider s'il s'agit de la même action élémentaire ou non, on doit d'abord calculer *le facteur de similarité (FS)* des deux signatures à partir de la formule suivante :

$(Sig_{1,2}) = \frac{S(Sig_1) + S(Sig_2)}{2}$,

avec :

$$(\text{Sig}_1) = \frac{\sum_{i=1}^{n} Taille(PI_{1_i})}{\sum_{i=1}^{n} Taille(PI_{1_i}) + \sum_{i=1}^{m} Taille(PD_{1_i})},$$

et ,

$$(\text{Sig}_2) = \frac{\sum_{i=1}^{n} Taille(PI_{2_i})}{\sum_{i=1}^{n} Taille(PI_{2_i}) + \sum_{i=1}^{l} Taille(PD_{2_i})},$$

PI_{1_i} : Partie identique *i* de la première signature,

PI_{2_i} : Partie identique *i* de la deuxième signature,

PD_{1_i} : Partie différente *i* de la première signature,

PD_{2_i} : Partie différente *i* de la deuxième signature,

n : Nombre de parties identiques,

m : Nombre de parties différentes dans la première signature,

l : Nombre de parties différentes dans la deuxième signature,

Par la suite, on fixe un seuil ε (dans notre cas ε = 0.8) dans l'intervalle [0.5, 1] dont à partir duquel on décide si les deux signatures représentent la même action ou non, en suivant la relation suivante :

Si FS (Sig_1, Sig_2) ≥ ε alors Sig_1 est identique à Sig_2, si non, ils ne le sont pas.

Par exemple, dans la figure précédente :

$(Sig_1) = \frac{450 + 500 + 1000}{2030} = 0.960$

$(Sig_2) = \frac{450 + 500 + 1000}{2000} = 0.975$

$(Sig_{1,2}) = \frac{0.960 + 0.9752}{2} = 0.967$

Dans cet exemple nous avons : FS (Sig_1, Sig_2) ≥ 0.8 alors, on peut dire que les deux signatures représentent la même action élémentaire. La valeur du seuil peut être entré par l'utilisateur pour jouer sur la précision du système, plus la valeur est élevée plus le système est précis.

La valeur du seuil va être dynamique choisi par l'utilisateur pour régler et configurer la sensibilité de notre système, plus cette valeur est élevée plus le système est sensible, le contraire est aussi vrai.

IV.5 Calcul probabiliste dans le module *ProbSys*

Pour prédire et détecter les menaces et les risques potentiels liés au trafic circulant sur le réseau, *ProbSys* effectue des calculs et des mesures quantitatives probabilistes en se basant sur les modèles n-grammes appelés aussi les modèles de *Markov* d'ordre *n-1*. Ce sont des modèles présentés pour la première fois dans les travaux de *Shannon* [44] pour la prédiction de caractères en fonction des autres chaines de caractères précédemment entrées. Ce sont des prédictions basées sur un corpus d'apprentissage contenant des informations statistiques sur les fréquences d'apparition des séquences de caractères dans un échantillon de textes. Le modèle de *Markov* est le plus utilisé actuellement dans le domaine du traitement automatique des langages naturels (*NLP : Natural Language Processing*) [45,46, 47]. Les mesures probabilistes vont permettre à notre système de prédire quel est le scénario d'attaque en cours et quelle est la prochaine action élémentaire de ce scénario d'attaque. Cette prédiction est faite en se basant sur les séquences d'actions élémentaires déjà capturées et leurs fréquences d'apparition dans le corpus d'apprentissage. Avec un modèle de Markov d'ordre $n-1$, nous pouvons prédire une séquence S de k composants ($k \geq n$) si nous connaissons les $n-1$ composants précédents. La probabilité de l'action en cours ne dépends que des $n-1$ composants qui la précédent. Ainsi, la probabilité de la séquence S est donnée par la formule suivante :

$$P(S = c_1, c_2, ..., c_k) = \prod_{i=1}^{k} P(c_i | c_{i-n+1}, ..., c_{i-1})$$

$$= P(c_1)P(c_2|c_1)P(c_3|c_1, c_2)...P(c_k|c_{k-n+1}, ..., c_{k-1})$$

la séquence c_{i-n+1}, ..., c_{i-1} représente l'historique du composant c_i à prédire.

Ainsi, pour un 2-gramme (*bi-gramme*) modèle appelé aussi modèle de Markov d'ordre 1, la probabilité du composant actuel dans une séquence ne dépend que du composant précédent, sa probabilité est donnée par la formule suivante :

$$P(S = c_1, c_2, ..., c_k) = \prod_{i=1}^{k} P(c_i|c_{i-1}) = P(c_1)P(c_2|c_1)P(c_3|c_2)...P(c_k|c_{k-1})$$

Nous utilisons le modèle de *Markov* en considérant un scénario d'attaque comme une séquence *S* de *n* actions élémentaires A_1,A_2, ...,A_n qui sont générées pendant la phase de capture et d'analyse des paquets TCP/IP du système *ProbSys*.

La probabilité d'un scénario est donnée par la formule suivante :

$$P(S = A_1, A_2, ..., A_n) = \prod_{i=1}^{n} P(A_i|A_{i-1}, ..., A_1)$$

$$= \prod_{i=1}^{n} P(A_i|A_1^{i-1}) \qquad Ici, A_1^{i-1} = A_1, ..., A_{i-1}$$

avec :

$$P(A_i|A_1^{i-1}) = \frac{freq(A_1^{i-1}A_i)}{\sum_{A_x} freq(A_1^{i-1}A_x)} = \frac{freq(A_1^{i-1}A_i)}{freq(A_1^{i-1})}$$

$freq(A_1^{i-1}A_i)$: est le nombre d'occurrences de la séquence $A_1^{i-1}A_i$ dans le corpus d'apprentissage. Donc,

$$P(A_i|A_{i-1}) = \frac{freq(A_{i-1}A_i)}{\sum_{A_x} freq(A_{i-1}A_x)} = \frac{freq(A_{i-1}A_i)}{freq(A_{i-1})}$$

À partir de cette règle et la règle des probabilités conditionnelles, on peut déduire la règle suivante :

$$P(S|A_1) = P(A_1, ..., A_n|A_1) = \frac{P(A_1, ..., A_n, A_1)}{P(A_1)}$$

$$= \frac{P(S)}{P(A_1)} = \frac{\prod_{i=1}^{n} P(A_i|A_1^{i-1})}{P(A_1)}$$

$$= \frac{P(A_1)P(A_2|A_1^1)P(A_3|A_1^2)...P(A_n|A_1^{n-1})}{P(A_1)}$$

$$= \frac{P(A_1)\prod_{i=2}^{n} P(A_i|A_1^{i-1})}{P(A_1)} = \prod_{i=2}^{n} P(A_i|A_1^{i-1})$$

Donc, on peut généraliser cette règle comme suite :

$$P(A_1, ..., A_n|A_1, ..., A_m)_{m<n} = P(A_1^n|A_1^m)_{m<n}$$

$$= \prod_{i=m+1}^{n} P(A_i|A_1^{i-1})_{m<n}$$

En se basant sur toutes ces règles, notre système probabiliste *ProbSys* est capable de calculer les deux mesures quantitatives suivantes :

- $P(S_i = A_1, ...,A_n|A'_1, ...,A'_m)$: la probabilité du scénario $S_i = A_1, ...,A_n$ sachant que nous avons déjà capturé la séquence $[A'_1, ...,A'_m]$.
- $P(A_i|A'_1, ...,A'_m)$: la probabilité de l'action A_i sachant que nous savons la séquence $[A'_1, ...,A'_m]$.

Le calcul de ces deux mesures ce fait selon les deux formules suivantes :

$$P(S_i = A_1, ..., A_k|A'_1, ..., A'_m)_{m<k} = \frac{P(A_1, ..., A_k, A'_1, ..., A'_m)}{P(A'_1, ..., A'_m)}$$

$$= \begin{cases} 0 & [A_1, ..., A_m] \neq [A'_1, ..., A'_m] \\ \prod_{i=m+1}^{k} P(A_i|A_1^{i-1})_{m<k} \end{cases}$$

$$= \begin{cases} 0 & [A_1, ..., A_m] \neq [A'_1, ..., A'_m] \\ \prod_{i=m+1}^{k} (\frac{freq(A_1^{i-1}A_i)}{freq(A_1^{i-1})})_{m<k} \end{cases}$$

$$P(A_i|A'_1, ..., A'_m) = \frac{freq(A_1^{'m}A_i)}{\sum_{A_x} freq(A_1^{'m}A_x)} = \frac{freq(A_1^{'m}A_i)}{freq(A_1^{'m})}$$

Exemple: supposons que nous avons les scénarios suivants : S_1 [aab] avec une fréquence 5, S_2 [bcba] avec une fréquence 5, S_3 [abbc] avec une fréquence 5, S_4 [bcbb] avec une fréquence 5 et S_5 [aac] avec une fréquence 5; et supposons que ProbSys a capté la séquence [abb], le calcul de la probabilité dans chaque étape de sa lecture est expliqué dans le tableau 3 suivant :

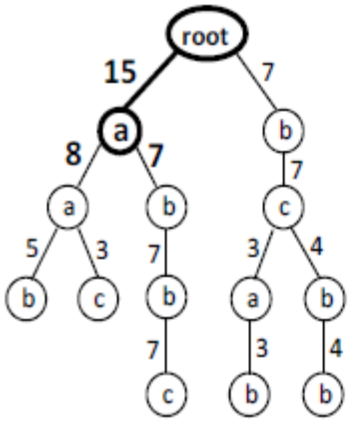
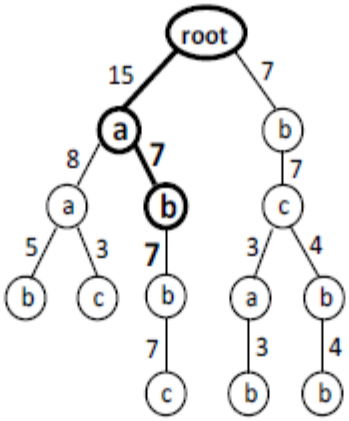
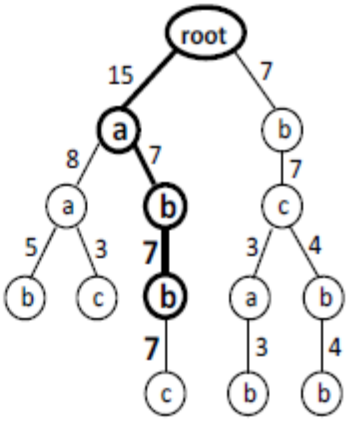
After receiving the action "a"	After receiving the action "b"	After receiving the action "b"
		
$P(S_1 = aab\|a) = \left(\frac{freq(aa)}{freq(a)}\right)$ $\left(\frac{freq(aab)}{freq(aa)}\right) = \left(\frac{8}{15}\right)\left(\frac{5}{8}\right)$ $= {}^{5}/_{15} \cong 0.33$ $P(S_2 = bcab\|a) = 0$ $P(S_3 = abbc\|a) = {}^{7}/_{15} \cong 0.47$ $P(S_4 = bcbb\|a) = 0$ $P(S_5 = aac\|a) = {}^{3}/_{15} = 0.2$	$P(S_1 = aab\|ab) = 0$ $P(S_2 = bcab\|ab) = 0$ $P(S_3 = abbc\|ab) =$ $\left(\frac{freq(abb)}{freq(ab)}\right)\left(\frac{freq(abbc)}{freq(abb)}\right)$ $= \left(\frac{7}{7}\right)\left(\frac{7}{7}\right) = 1$ $P(S_4 = bcbb\|ab) = 0$ $P(S_5 = aac\|ab) = 0$	$P(S_1 = aab\|abb) = 0$ $P(S_2 = bcab\|abb) = 0$ $P(S_3 = abbc\|abb) =$ $\left(\frac{freq(abbc)}{freq(abb)}\right) = \left(\frac{7}{7}\right) = 1$ $P(S_4 = bcbb\|abb) = 0$ $P(S_5 = aac\|abb) = 0$
$P(a\|a) = \left(\frac{freq(aa)}{freq(a)}\right) = \left(\frac{8}{15}\right)$ $\cong 0.53$ $P(b\|a) = \left(\frac{freq(ab)}{freq(a)}\right) = \left(\frac{7}{15}\right)$ $\cong 0.47$ $P(c\|a) = 0$	$P(a\|ab) = 0$ $P(b\|ab) = \left(\frac{freq(abb)}{freq(ab)}\right)$ $= \left(\frac{7}{7}\right) = 1$ $P(c\|ab) = 0$	$P(a\|abb) = 0$ $P(b\|abb) = 0$ $P(c\|abb) = \left(\frac{freq(abbc)}{freq(abb)}\right)$ $= \left(\frac{7}{7}\right) = 1$

Tableau 3: Exemple du calcul probabiliste

Après avoir reçu l'action «a» *ProbSys* constate qu'elle appartient au scénario S_3 avec une probabilité de 0.47, et au S_1 avec une probabilité de 0.33 et au S_5 avec une probabilité de 0.2. Lorsqu'il reçoit la deuxième action élémentaire «b», il constate qu'elle appartient au scénario S_3 avec une probabilité de 1, et la même chose pour l'action «b».

IV.6 Conclusion

L'architecture proposée pour notre système se base sur le principe d'installer un outil en parallèle au circuit normal entre un réseau local et le monde externe (Internet) pour identifier les attaques et génère les alertes au moment où les attaques se déroulent. L'image qui se ressemble à notre système pour simplifier et rapprocher le schéma, et celle de l'emplacement d'un galvanomètre ou un voltmètre dans un circuit électrique.

CHAPITRE V

IMPLÉMENTATION ET INTERPRÉTATION DES RÉSULTATS EXPÉRIMENTAUX

Dans ce chapitre nous allons montrer les différents modules implémentés qui composent chaque système et comment se communiquent entre eux pour mettre le système en production. Vers la fin nous allons discuter les résultats de notre expérimentation.

V.1 Introduction

Notre système est formé par plusieurs modules qui sont responsables à l'exécution des différentes tâches qui s'unissent pour compléter le travail de notre système probabiliste.

Nous avons implémenté notre système en utilisant le langage C#.net (*Visual Studio net 2008*), *HoneyLens* est implémenté en 5 000 lignes de code et le *ProbSys* est en 12 000 lignes de codes.

Nous avons implémenté et intégré des outils statistiques dans le système lui-même pour les raisons suivantes :

- Voir le déroulement et la prise de décision du système en temps réel.
- Étudier et analyser les attaques avec plus de détails.
- Générer des rapports statistiques à partir des résultats obtenus.

Pour tester l'efficacité et la fiabilité de notre système et comparer son rendement avec d'autres systèmes de détection d'intrusion, nous avons installé et configuré l'IDS *Snort* juste avant le réseau locale. Les alertes générées par *Snort* est notre système sont interpréter par le module statistique en affichant des rapports détaillés.

Malgré qu'il y'a des ajouts que nous devons le faire pour le système, nous étions capable d'avoir des résultats considérables qui augmentent la valeur de notre système.

V.2 Les modules et les composants des deux systèmes HoneyLens et ProbSys

V.2.1 Honeylens

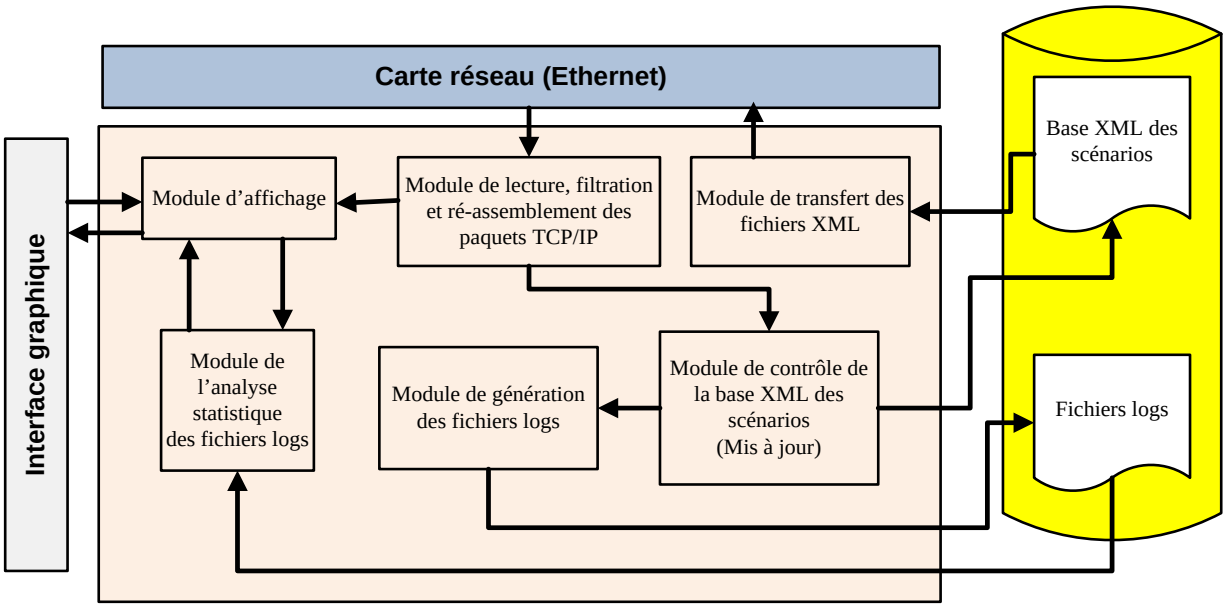


Figure 19 : Les différents modules du *HoneyLens*

La figure 19 montre les différents modules du *HoneyLens*, chacun d'eux est responsable à faire une tâche bien déterminée :

- *Module de lecture, filtration et ré-assemblement des paquets TCP/IP* : ce module est responsable à la lecture du trafic de la carte réseau ou à partir d'un fichier **.pcap*, il fait aussi l'extraction des éléments d'attaques (action élémentaire, sous-scénario et scénario). L'utilisateur doit entrer quelques paramètres pour guider et filtrer la capture du trafic, comme le montre la figure suivante :

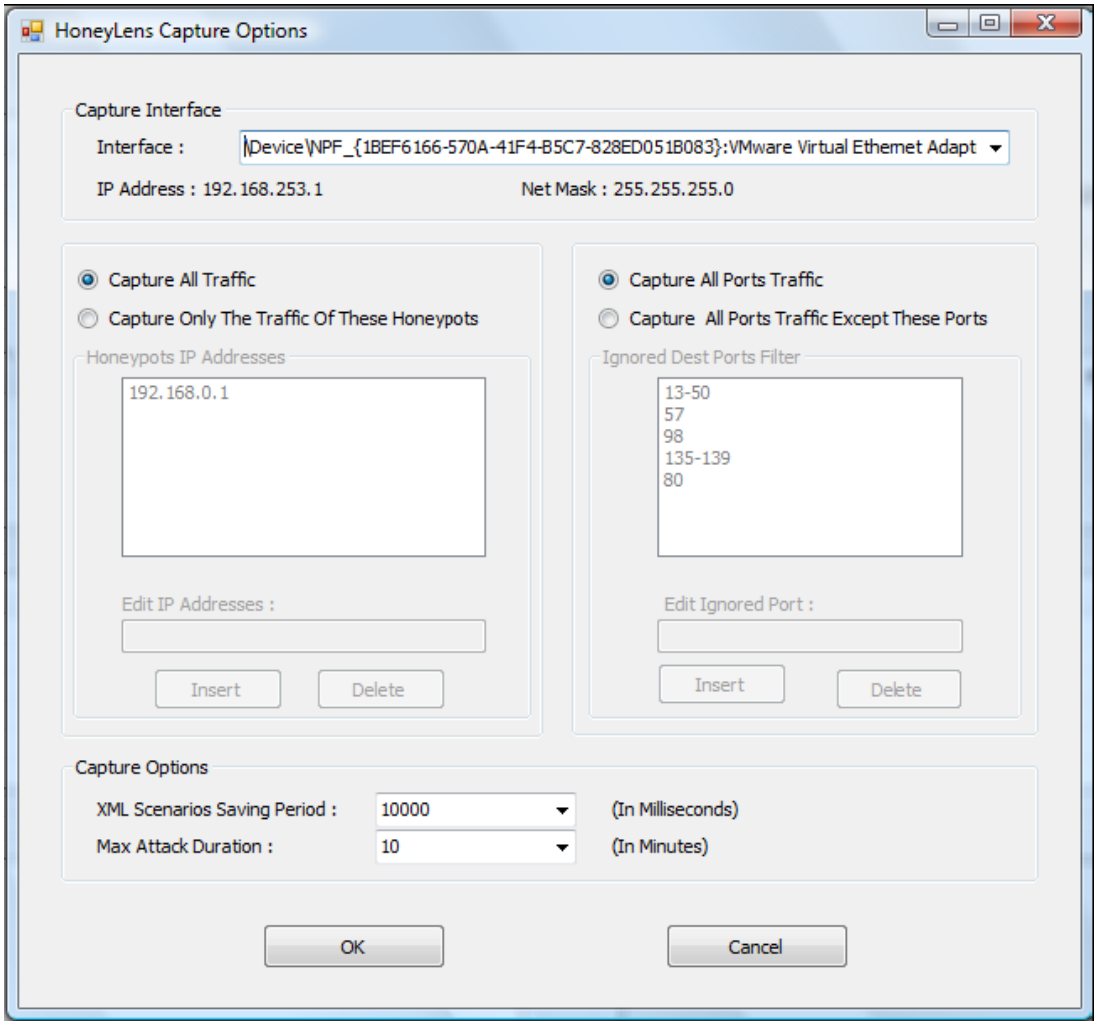


Figure 20 : Paramètres et options de la capture

- *Module de contrôle de la base XML des scénarios* : une fois les éléments d'attaques sont identifiés, ce module va les ajouter dans les fichiers XML appropriés après avoir faire le test de similarité comme c'est indiqué dans le chapitre précédent.
- *Module de génération des fichiers logs* : ce module s'occupe à enregistrer les informations sur les attaques dans des fichiers logs.
- *Module de transfert des fichiers XML* : ce module joue le rôle d'un serveur pour répondre aux requêtes qui viennent du *ProbSys* pour le transfert des fichiers XML à travers un Socket.
- *Module de l'analyse statistique des fichiers logs* : ce module fait l'analyse des fichiers logs et envoie les informations au module d'affichage.

- *Module d'affichage* : ce module s'occupe à afficher les éléments d'attaques avec leurs fréquences et leurs contenus en temps réel.

V.2.2 ProbSys

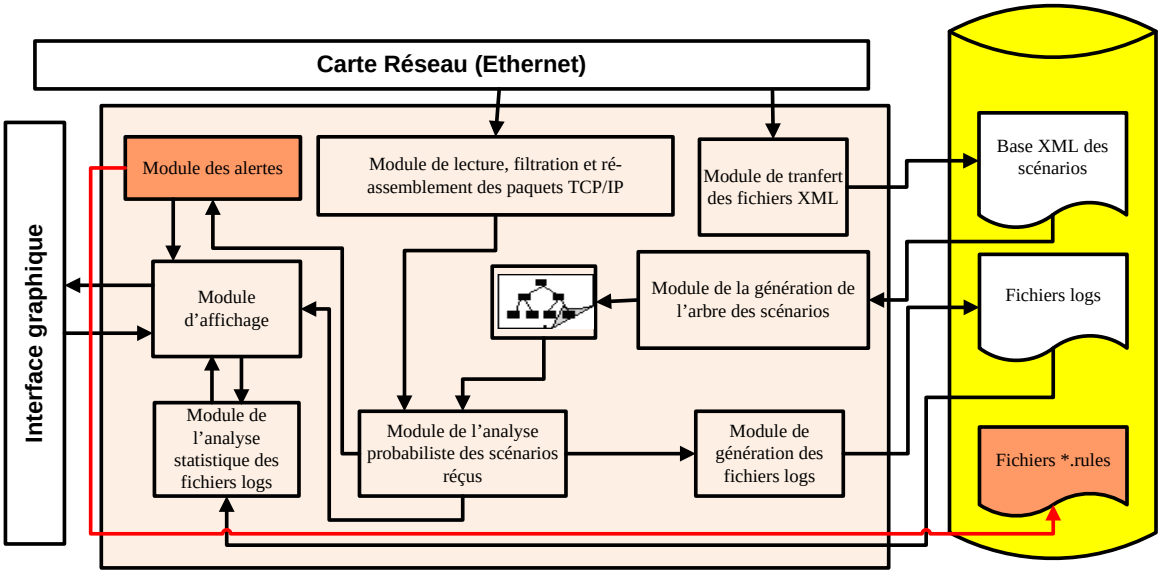


Figure 21: Les différents modules du *ProbSys*

Le *ProbSys* est notre IDS qui a plus de fonctionnalité que le *HoneyLens* ses modules sont présentés dans la figure précédente.

- *Module de lecture, filtration et ré-assemblement des paquets TCP/IP* : ce module va faire la lecture et l'extraction des éléments d'attaques (action élémentaire, sous-scénario et scénario) - à partir du trafic entrant au réseau local – en suivant la même approche expliqué dans le chapitre précédent que le *HoneyLens* utilise pour analyser le trafic du *honeypot*.
- *Module de transfert des fichiers XML* : ce module joue le rôle du client en faisant des requêtes régulières pour importer les fichiers XML du *HoneyLens*. La figure suivante montre comment se passe le transfert entre les deux systèmes

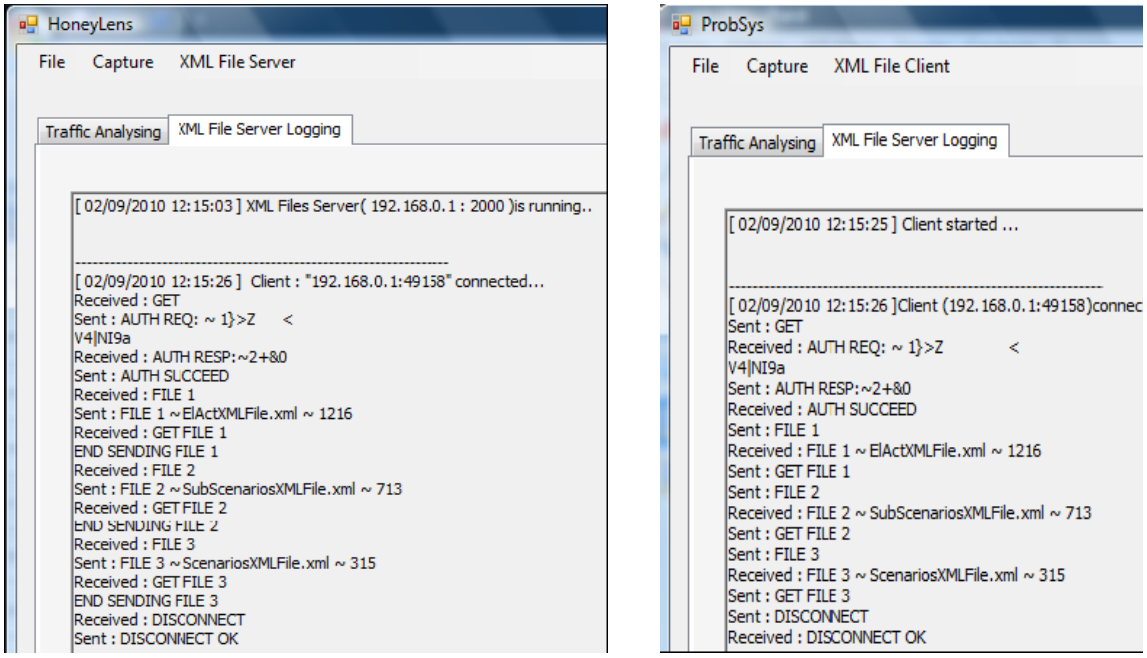


Figure 22: Procédure de transfert des fichiers XML

- *Module de génération de l'arbre des scénarios* : lorsqu'il reçoit les informations sous format XML, il fait le mis à jour ou la création de la structure arborescente. (Voir le paragraphe suivant).
- *Module de l'analyse probabiliste des scénarios reçus* : ce module va suivre la procédure expliquée dans le chapitre précédent.
- *Module de génération des fichiers logs* : ce module va faire la distinction du trafic suspect, non suspect et le trafic qui correspond aux attaques, puis il les envoie dans des fichiers logs différents. (Voir paragraphe capture et journalisation du trafic).
- *Module des alertes* : ce module va générer les alertes, dans un fichier log et il va créer une règle *Snort* dans un fichier appelé *probsys.rules* dans le répertoire *rules*.
- *Module de l'analyse statistique des fichiers logs* : ce module va faire la lecture des fichiers logs selon les besoins de l'utilisateur et va générer les courbes appropriés. (Voir paragraphe Module statistique d'apprentissage).
- *Module d'affichage* : ce module permet aux utilisateurs de voir comment se passent les différentes étapes de l'extraction des éléments d'attaque et de suivre les calculs statistiques en temps réel : plusieurs rapports peuvent être générés sous forme des graphes.

V.3 Procédure d'extraction et de structuration des éléments d'attaques (Scénario, sous-scénario et action élémentaire)

V.3.1 Caractéristiques des fichiers XML

Nous avons choisi d'enregistrer les informations dans des fichiers XML, car ils peuvent facilement maintenir les données sous une structure arborescente, et ainsi pour les raisons suivantes :

- Une séquence d'élément d'attaque diffère d'une attaque à une autre par sa taille (nombre d'action par séquence) et par l'ordre de ces éléments, ce qui rend l'utilisation des bases de données ou n'importe quelle forme matricielle déconseillée.
- Les fichiers XML enregistrent les données sous forme des nœuds, les nœuds qui appartiennent à la même branche forment un *XmlElement*. Un arbre peut être facilement reflété par un fichier XML, la structure et la longueur des branches peuvent être identiques ou différentes. L'accès au fichier XML se fait par l'itération des nœuds, dans notre cas l'itération va se faire par sélection : c'est-à-dire en ayant le nœud approprié on va chercher seulement dans ses nœuds fils, ce qui rend la recherche dans cette structure beaucoup plus rapide que l'itération du fichier en entier.
- La lecture et la mise à jour d'un fichier XML se fait de manière plus précise et plus rapide : nous sommes capables de faire le changement de n'importe quel nœud par la simple itération mentionnée ci-dessus. Si nous voulons ajouter une nouvelle séquence, nous devons faire la lecture des valeurs des nœuds en faisant la comparaison avec ceux qui existent dans le fichier XML, dès que nous trouvons une valeur différente nous construisons une nouvelle branche de l'arbre. Ce processus simplifie et localise la lecture et la mise à jour de notre structure arborescente, de plus il empêche l'augmentation énorme de la taille de ces fichiers.
- On peut utiliser les fichiers XML pour assurer le filtrage et le regroupement des différents éléments des attaques par la simple itération en allant de la racine vers les feuilles.
- Cette structure est capable de sauvegarder des valeurs de fréquence pour chaque nœud, par la suite les calculs de fouilles de données (*Data Mining*) seront beaucoup plus

faciles - pour ce type de données - que n'importe quel autre système qui offre ces services comme *le Microsoft SQL Server (Version Entreprise)*.

V.3.2 Enregistrement et structuration des données

La structuration et la prévention des actions élémentaires ou des sous-scénarios se font de la même manière : chaque nœud dans le fichier XML va contenir la valeur de l'élément et la fréquence qui représente le nombre de fois que cet élément est vu.

Chaque catégorie des éléments d'attaque est sauvegardée dans un fichier XML séparé comme le montre la figure 24, la structure arborescente se génère à partir de ces fichiers (figure 23).

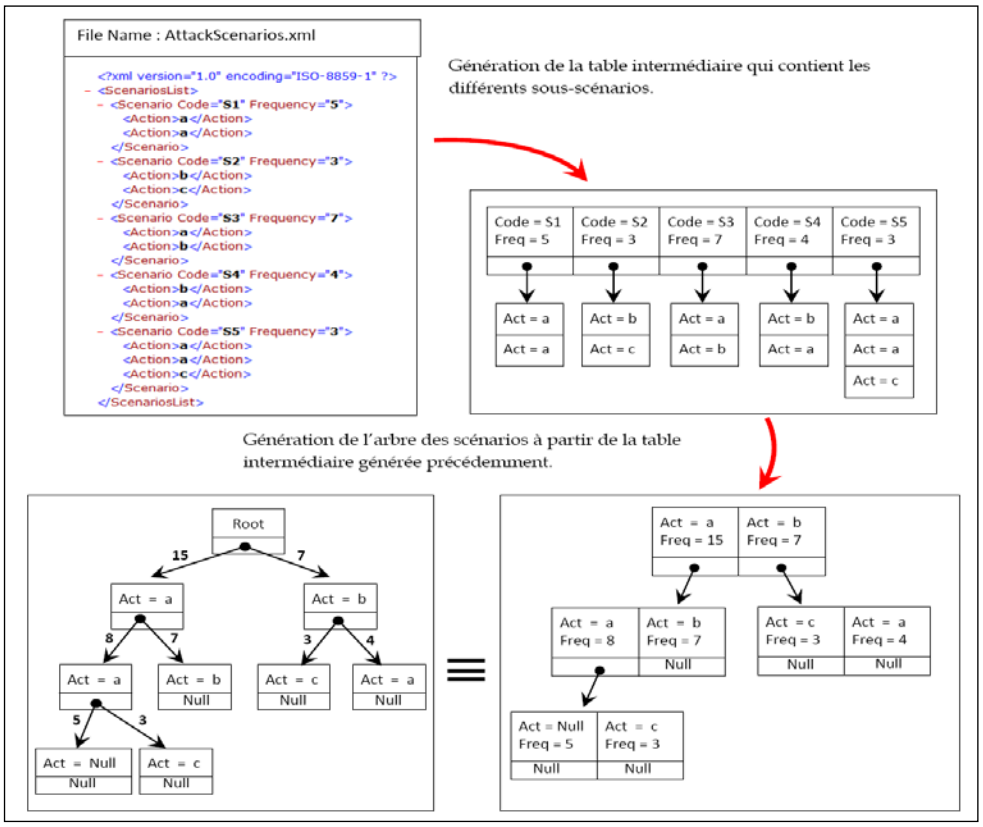


Figure 23 : Étapes de construction de la structure arborescente

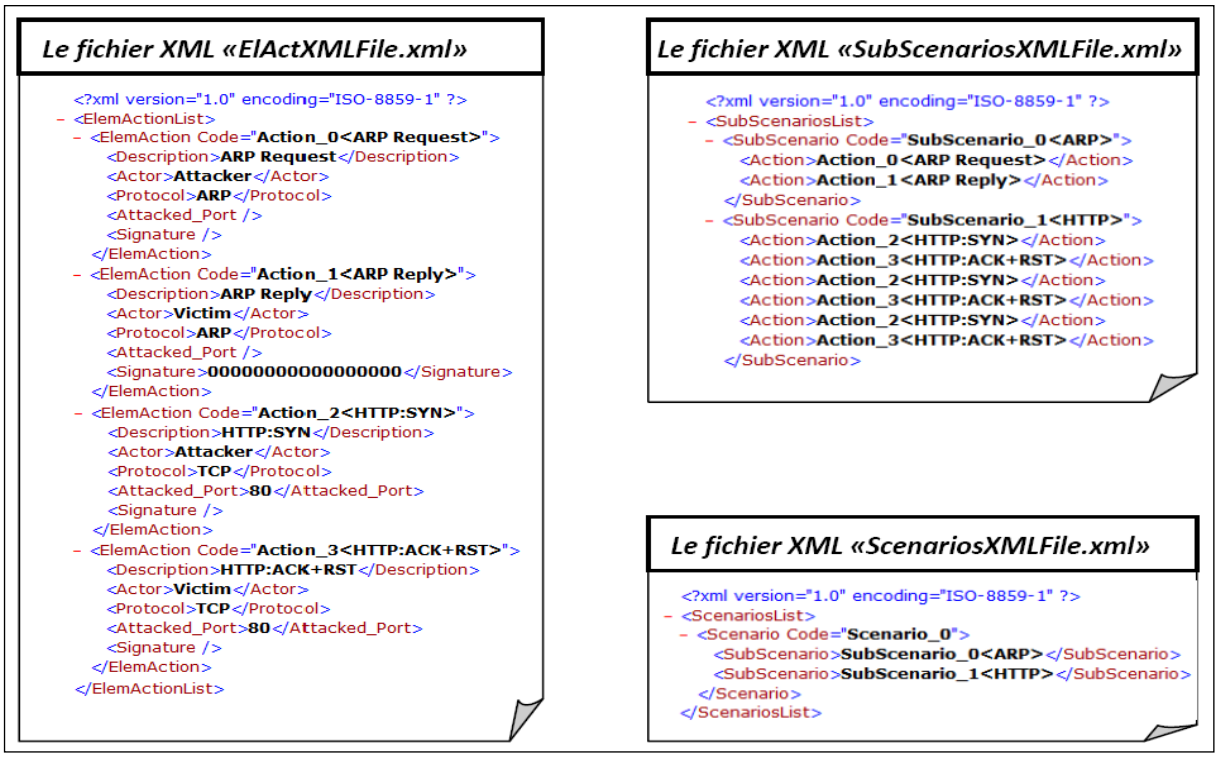


Figure 24 : Les trois types des fichiers XML générés par le *HoneyLens*.

La figure 24 montre les trois fichiers XML généré par le *HoneyLens* de chaque type des éléments d'attaque :

1. Le premier est le fichier «*ElActXMLFile.xml*» qui contient la liste de toutes les actions élémentaires appliquées par les différents attaquants dans une ou plusieurs attaques, chaque action élémentaire est représentée dans le fichier XML par un nœud `<ElemAction>` qui contient un ensemble de nœuds fils représentants les différents champs d'une action élémentaire (le code, la description, l'acteur, le protocole, le port de destination et la signature).

2. Le deuxième est le fichier «*SubScenariosXMLFile.xml*» qui contient la liste des sous-scénarios d'une ou plusieurs sessions d'attaques, dont chaque sous-scénario est constitué d'une suite ordonnée d'actions élémentaires représentées par leurs codes uniques donnés dans le premier fichier, chaque sous scénario d'attaque est représentée dans le fichier XML par un nœud `<SubScenario>` contenant une suite de nœuds fils `<Action>`. Dans notre exemple, l'attaquant a appliqué deux sous scénarios (deux sessions port : deux ports sont attaquées), le premier sous scénario (`<SubScenario Code= "SubScenario_0<ARP>">`) pour chercher l'adresse MAC de la victime

(envoi/réception d'une requête/réponse ARP), alors que le deuxième sous scénario (<SubScenario Code= "SubScenario_1<HTTP>">) est pour essayer d'ouvrir une connexion TCP vers le port 80 de la victime (il a fait trois tentatives).

3. Le troisième est le fichier «*ScenariosXMLFile.xml*» qui contient la liste des scénarios, dont chaque scénario contient tous les sous-scénarios appliqués à un moment donné par le même attaquant; par exemple, si un attaquant α attaque un port *X* en suivant un scénario *A*, puis s'il attaque un autre port *Y* en suivant un autre scénario *B*, alors dans ce cas, notre système va stocker la suite *A-B* dans ce fichier pour signaler que le scénario *A* est généralement suivi par le scénario *B*. Chaque scénario est représenté dans le fichier XML par un nœud <Scenario> contenant une suite de nœuds fils <SubScenario>, dont chacun d'eux porte le code d'un sous scénario du deuxième fichier. Dans notre exemple, nous avons un seul scénario (<Scenario Code= "Scenario_0">) contenant deux sous scénarios, le premier pour chercher l'adresse MAC de la victime (ARP) et le deuxième pour ouvrir une connexion vers le port 80 (HTTP).

Une fois la capture commence, la mis à jour des fichiers XML et l'affichage des listes des différentes catégories se fait automatiquement dans l'interface principale de notre système comme le montre la figure 25. La première liste de l'interface représente la liste des scénarios d'attaques capturés (chaque scénario est l'ensemble de tous les paquets ayant le même adresses IP source et le même IP destination), la deuxième liste représente la liste des sous-scénarios du scénario sélectionné dans la première liste (chaque sous-scénario regroupe les actions qui portent les mêmes adresses IP source et destination **et** le même port destination), la troisième liste représente la liste des actions élémentaires du sous-scénario sélectionné dans la deuxième liste. La quatrième et la cinquième liste de l'interface représentent les détails de l'action élémentaire sélectionnée.

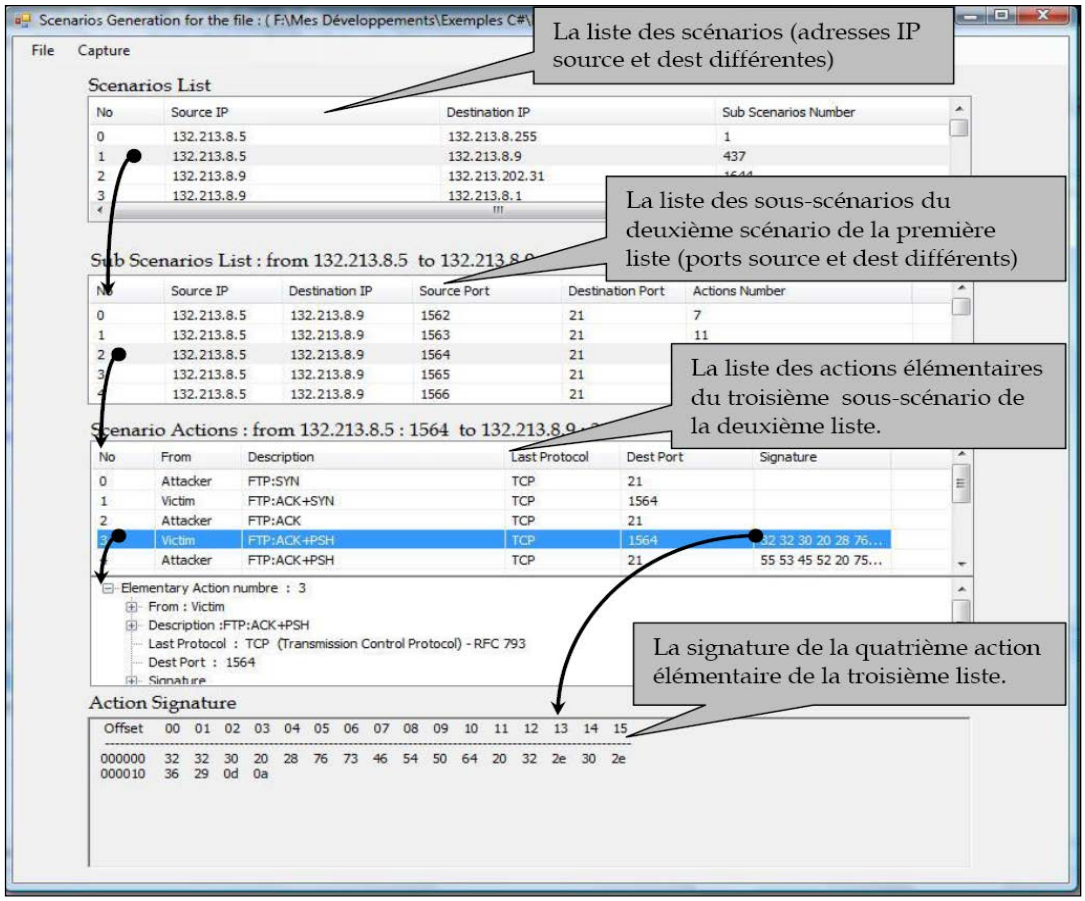


Figure 25 : Interface du *HoneyLens* montrant différents éléments qui forment les scénarios d'attaques

V.4 Capture et journalisation du trafic

La distinction entre les différents éléments d'attaque se fait de la même façon chez les deux systèmes. Mais de plus, lors d'une attaque, il va générer des alertes sous forme des nouvelles règles *Snort* enregistrées dans un fichier *.rules. *ProbSys* est capable d'identifier et de préciser si un tel trafic est normal, suspect ou appartient à une vraie attaque :

- **Trafic total** : l'ensemble de tout le trafic brut recueilli sans filtration ou classification.
- **Trafic suspect** : c'est l'ensemble des séquences des sous-scénarios ou des actions élémentaires qui ont une partie d'intersection avec le début d'une ou plusieurs attaques.
- **Trafic correspondant à des attaques** : c'est l'ensemble de toutes les séquences qui font partie (inclus ou égal) avec d'autres séquences qui correspondent au attaques.

ProbSys va sauvegarder ces informations dans trois types de fichiers logs qui se trouvent dans trois répertoires, dont chacun contient un ensemble de fichiers log de même type, chaque fichier représente le trafic d'une journée différente comme le montre la figure suivante.

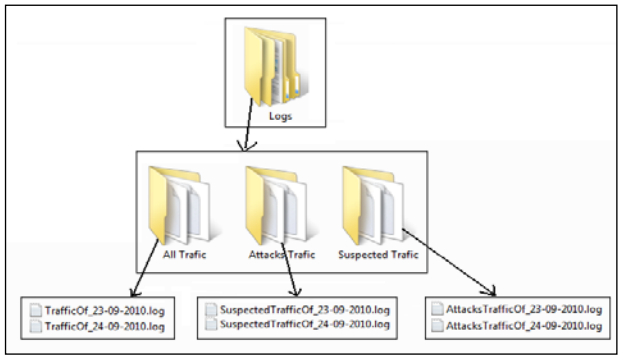


Figure 26 : Journalisation des différents types de trafic

En mode actif, les différents services du *ProbSys* travaillent en parallèle, les différents éléments d'attaque vont être affichés dans l'onglet (*Suspected Traffic*) comme le montre la figure 27. La première liste de l'interface représente la liste des sous-scénarios du trafic suspect capturé; en cliquant sur un élément de cette liste, les actions élémentaires vont être affichées dans la deuxième liste, l'avant-dernière colonne représente la prochaine action élémentaire dont sa probabilité est indiquée dans la dernière colonne. La troisième liste représente la liste des prochaines actions élémentaires avec leurs probabilités.

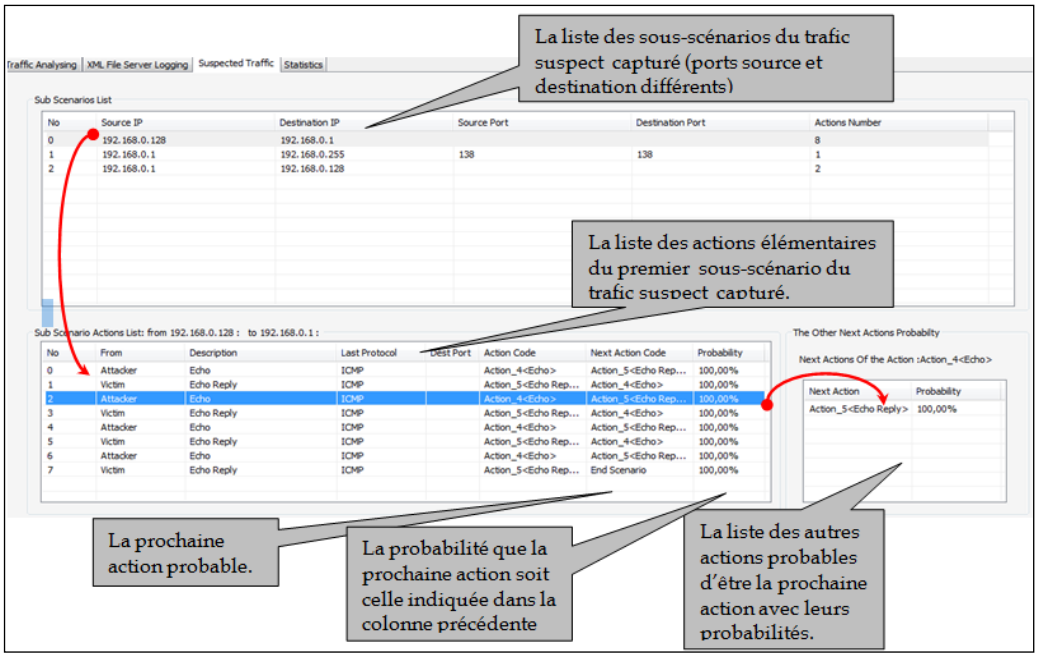


Figure 27 : Trafic suspect identifié par le ProbSys

V.4.1 Processus de journalisation (un exemple)

Supposons que nous avons la structure des scénarios «aa», «aac», «ab», «bc» et «ba» avec les fréquences 5, 3, 7, 3 et 4 respectivement qui existe dans la figure 28 en structure arborescente**,** et que nous avons le trafic reçu suivant **:** [aac], [bcd], [ab] et [badb], [dbacf], [cab] et [dca]. À partir de cet arbre, le *ProbSys* est capable d'effectuer les calculs statistiques pour prédire les prochaines actions élémentaires, et de déterminer si l'un de ces trafics est suspect ou normal.

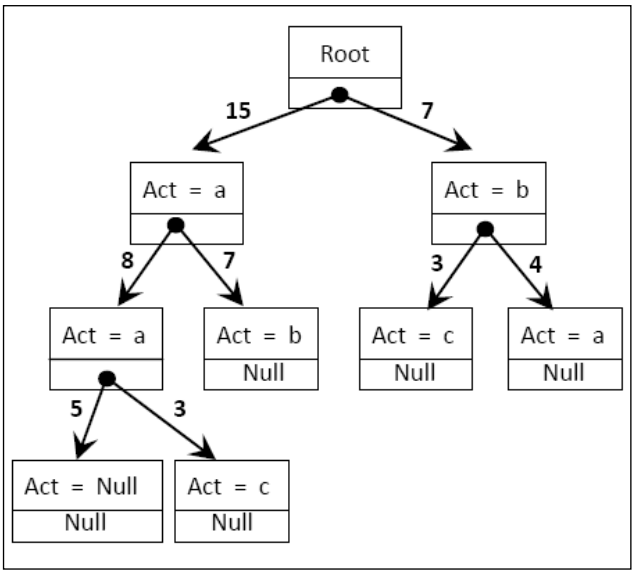


Figure 28 : Scénarios d'attaques représentés en une structure arborescente

Le traitement de ces sous-scénarios par le *ProbSys* se fait comme suit :

Réception du premier sous-scénario [**dbacf**] : Lorsqu'il reçoit la première action élémentaire «d», il teste si cette action représente un début d'un sous-scénario connu ou non, il trouve qu'il s'agit d'un nouvel élément, donc il va décider directement que c'est un trafic non suspect (normal), par la suite il va l'ajouter dans le log file des trafics non suspects.

Réception du deuxième sous-scénario [**aac**] : lorsqu'il reçoit la première action élémentaire «a», il trouve qu'il y'a des scénarios qui commencent par cette action, donc il s'agit d'un trafic suspect, mais c'est tôt de dire si c'est une vraie attaque ou non. À partir de cet arbre, les calculs de prédiction montrent que les prochaines actions possibles sont «a» avec une probabilité de (8/15)*100 = 53% et «b» avec une probabilité de (7/15)*100 = 47%. Il passe la lecture au deuxième niveau, il trouve que la prochaine action possible (après «aa») n'est que le «c» mais avec une probabilité de (3/8)*100 = 37.5% et non pas 100%, car il y'a des sous-scénarios qui sont formés de «aa» seulement, et que leur probabilité est de (5/8)*100 =

62.5%. En terminant la lecture de «c», le *ProbSys* décide que la séquence «aac» correspond à une vraie attaque donc il va le sauvegarder dans le log des vraies attaques.

Le même test s'effectue pour le reste des sous-scénarios.

Après avoir finir la lecture des sept sous-scénarios, nous trouvons quatre qui sont suspects : [aac], [bcd], [ab] et [badb], parmi eux nous avons deux qui correspondent au vraies attaques : [aac] et [ab], et trois non suspects qui sont : [dbacf], [cab] et [dca].

V.4.2 Module statistique d'apprentissage

Pour analyser les fichiers logs du trafic capturé, nous avons développé un module qui analyse tout le trafic sauvegardé dans les trois fichiers logs et affiche le résultat d'analyse sous forme d'un graphique dans l'onglet *Statistics* comme la montre la figure suivante :

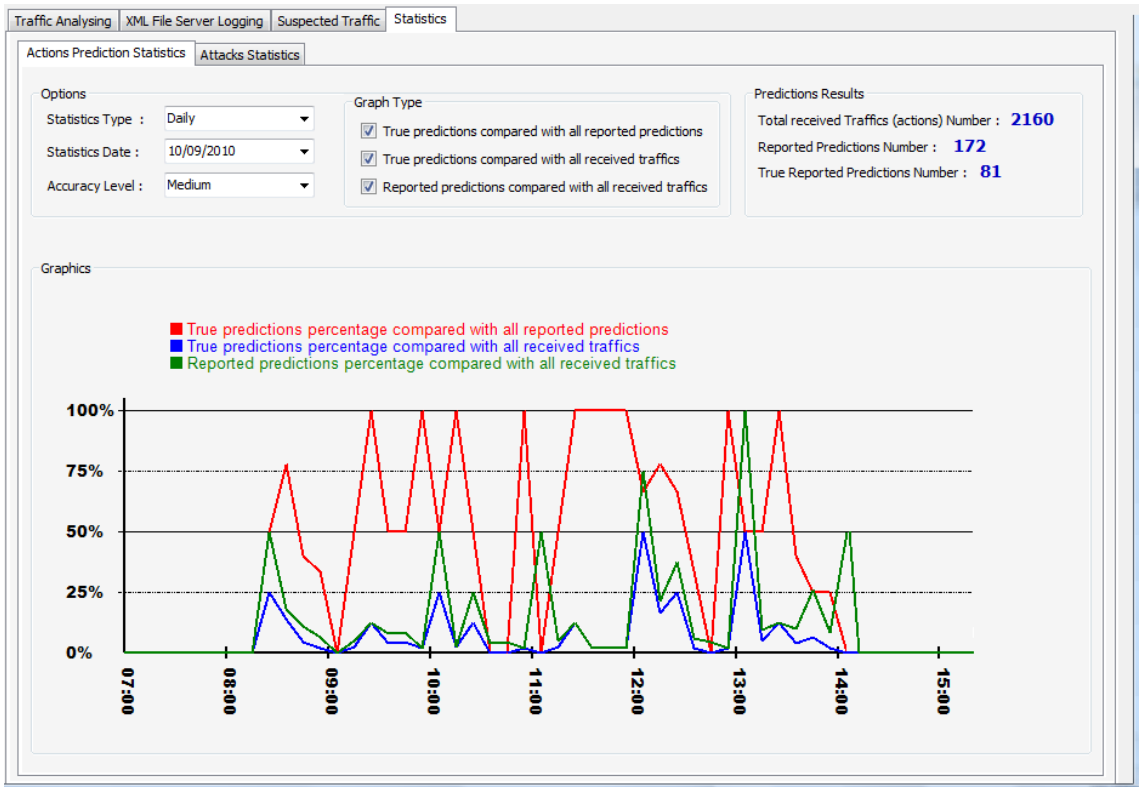


Figure 29 : Affichage des calculs statistiques

Dans cet onglet nous avons deux types d'analyse représentés dans les deux sous-onglets : *Actions Prediction Statistics* et *Attacks Statistics.* Dans le premier on affiche les statistiques concernant la prédiction de la prochaine action élémentaire. Le graphique nous montre les pourcentages suivants : le nombre de prédictions réussies par rapport au nombre de prédictions annoncées (la ligne rouge), le nombre de prédictions réussies par rapport à toutes

les actions de tous les trafics capturés y compris le trafic suspect et non suspect (la ligne bleue) et le nombre de prédictions annoncées par rapport aux actions de tous les trafics capturés y compris le trafic suspect et non suspect (la ligne verte).

Dans le deuxième type d'analyse affiché dans l'onglet *Attacks Statistics*, on affiche les statistiques concernant les attaques détectées. Dans ce graphique nous pouvons distinguer : le nombre de sous-scénarios correspondants à des attaques détectées par rapport au nombre de sous-scénarios annoncés comme suspects (ligne rouge), le nombre de sous-scénarios correspondants à des attaques détectées par rapport à tous les sous-scénarios capturés y compris les sous-scénarios suspects et non suspects (ligne bleue) et le nombre de sous-scénarios annoncés comme suspects par rapport à tous les sous-scénarios capturés y compris les sous-scénarios suspects et non suspects (va être représenté par une ligne verte). Pour chacun de ces deux types d'analyse, nous avons deux options : analyse quotidienne (*Daily*) uniquement pour le trafic capturé durant une seule journée et analyse mensuel (*Monthly*) pour le trafic capturé durant un mois.

De plus nous avons implémenté d'autres options qui vont nous permettre de préciser la nature d'analyse qui doit être affichée dans le graphe en choisissant ou non les types d'analyse désirés, comme nous pouvons également préciser le niveau d'exactitude du graphe.

V.4.2.1 Statistiques sur la prédiction de la prochaine action élémentaire (à partir de l'exemple précédent V 5.1)

Le nombre de prédictions réussies par rapport au nombre de prédictions annoncées pour le premier et le deuxième sous-scénario : dans le cas du premier sous-scénario [dbacf], le *ProbSys* n'a annoncé aucune prédiction, car c'est trafic qui ne correspond pas à aucune attaque. Pour le deuxième sous-scénario [aac], le *ProbSys* a fait trois prédictions, dont deux (la première et la dernière) étaient correctes alors qu'une prédiction (la deuxième) était incorrecte. Donc, ici le nombre de prédictions réussies par rapport au nombre de prédictions annoncées est : (2/3)*100 = 66%.

Le nombre de prédictions réussies par rapport à toutes les actions de tous les trafics capturés y compris le trafic suspect et non suspect pour le premier et le deuxième sous-scénario : pour le premier [dbacf], nous avons cinq actions et aucune prédiction correcte, pour le deuxième [aac], nous avons trois actions et deux prédictions correctes. Donc, le nombre de prédictions

réussies par rapport à toutes les actions de tous les trafics capturés y compris le trafic suspect et non suspect est : (2/8)*100 = 25%.

Le nombre de prédictions annoncées par rapport à toutes les actions de tous les trafics capturés y compris le trafic suspect et non suspect pour le premier et le deuxième sous-scénario : pour le premier, nous avons cinq actions et aucune prédiction, pour le deuxième, nous avons trois actions et trois prédictions. Donc, le nombre de prédictions annoncées par rapport à toutes les actions de tous les trafics capturés y compris le trafic suspect et non suspect est : (3/8)*100 = 37%.

Statistiques sur les attaques détectées

Le nombre de sous-scénarios correspondants à des attaques détectées par rapport au nombre de sous-scénarios annoncés comme suspects est : (2/4)*100 = 50%.

Le nombre de sous-scénarios correspondants à des attaques détectées par rapport à tous les sous-scénarios capturés y compris les sous-scénarios suspects et non suspects est : (2/7)*100 = 28%.

Le nombre de sous-scénarios annoncés comme suspects par rapport à tous les sous-scénarios capturés y compris les sous-scénarios suspects et non suspects est : (4/7)*100 = 57%.

V.5 Expérimentation et analyse des résultats

V.5.1 Instruments matériels et périphériques

Nous avons installé et configuré notre système au LRSI (Laboratoire de recherche en sécurité informatique) derrière un pare-feu pour des règles et des mesures de sécurité suivis par l'UQO, ce qui réduit et bloque plusieurs attaques venants de l'extérieur. Pour notre système nous avons utilisé deux machines hôtes physiquement séparées et reliées par un *Hub* qui réalise aussi leurs connexions avec l'Internet. Nous avons choisi le *Hub* pour qu'il n'y ait pas de filtrage et pour que le *ProbSys* et le *HoneyLens* soient capables de voir tout le trafic passant à travers le réseau. Pour tester la réaction de notre système, contre des attaques spécifiques, nous avons utilisé une autre machine pour réaliser quelques attaques à partir d'un *live-USB* contenant le *Backtrack* comme système d'exploitation.

V.5.2 Installation et configuration des différents composants

Sur la première machine hôte H_1 (*Windows 7 Pro.*), nous avons utilisé le *VMware* pour faire la configuration et l'installation des deux *pots de miel* et une autre machine pour l'installation du *Honeywall (CDROM ROO)* : le premier *honeypot1* est un *Windows XP* et le deuxième *honeypot2* était un *Fedora Core 13*. Sur les deux machines, nous avons mis en marche quelques services comme le *Telnet*, *FTP*, *IIS* pour le *windows XP* et l'Apache pour le *Fedora*. Au début nous avons mis en marche les trois machines virtuelles pour que le *honeynet* soit en production. Puis nous avons installé le *HoneyLens* pour qu'il sniffe la carte réseau et analyse tout le trafic, le *HoneyLens* est configuré pour qu'il ne capte et analyse que le trafic entrant vers les deux pots de miel, la figure qui suit montre comment nous pouvons entrer ces paramètres. Enfin, nous avons lancé le serveur XML pour qu'il soit prêt pour répondre aux requêtes de transfert des fichiers logs (XML).

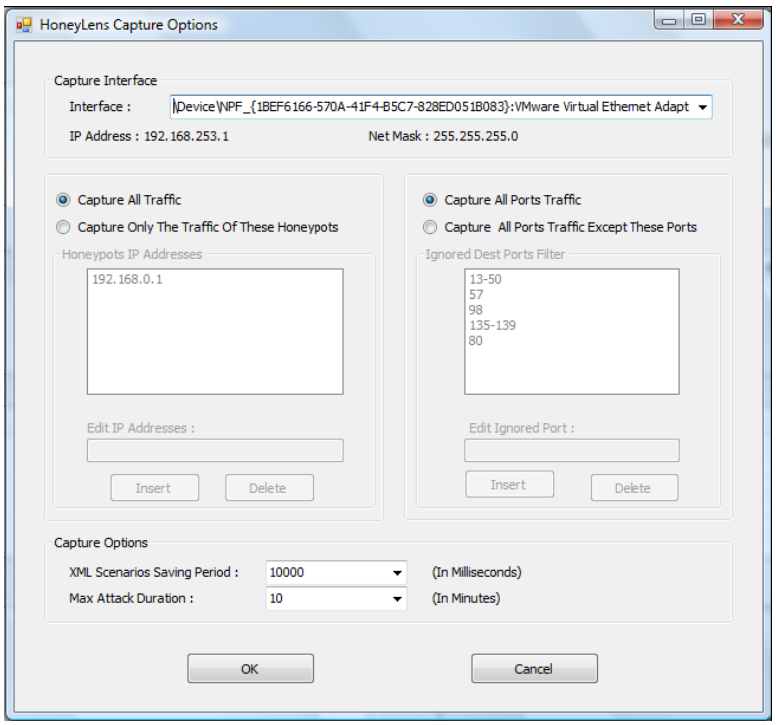


Figure 30 : Affichage des calculs statistiques

Sur la deuxième machine hôte H_2 (*Windows XP Pro.*) nous avons mis en marche les mêmes services se trouvant dans les *pots de miel*. Puis, nous avons installé et configuré *Snort* en faisant les changements appropriés sur le fichier *config* et le fichier *logs*. Dans le répertoire *"rules*" nous avons ajouté un fichier appelé *ProbSys.rules* qui va contenir les règles générées par le *ProbSys*, ce fichier est considéré comme l'augmentation concrète par notre système de

la base d'information de *Snort*. De plus, nous avons installé le *ProbSys*, puis nous avons lancé le client XML. Dans les deux machines hôtes il faut installer le *WinPcap* pour lire le trafic à partir de la carte réseau, et le *Microsoft .Net Compact Framework 3.5* pour le fonctionnement du *HoneyLens* et *ProbSys*. La figure 31 montre les différents composants de notre système lors de l'installation au lab.

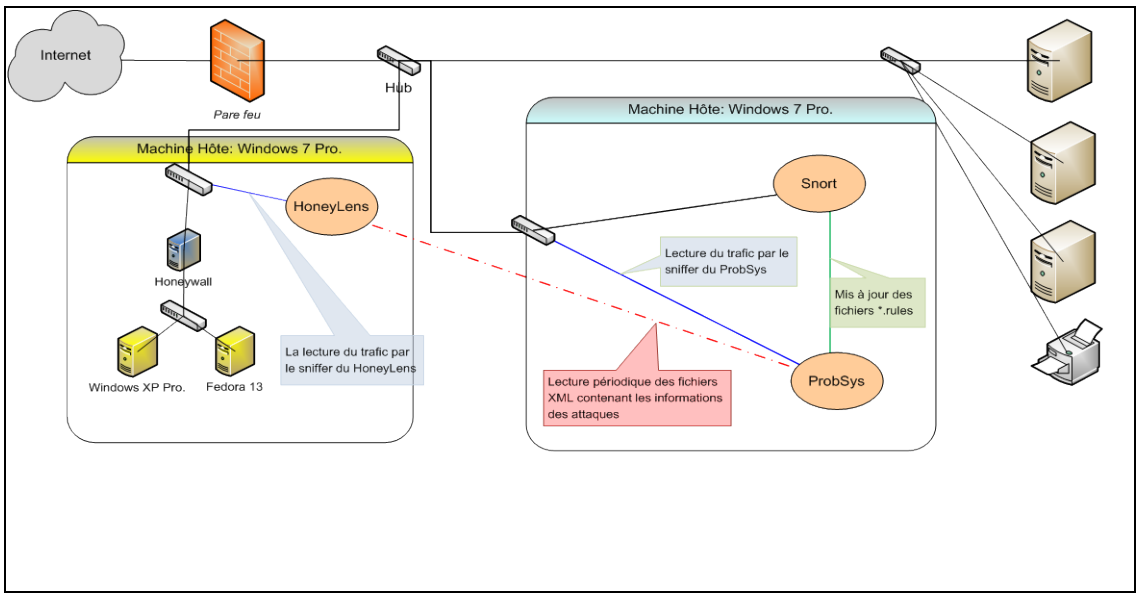


Figure 31 : Les différents éléments de notre système lors de l'expérimentation

Tout le système est mis en marche pendant une période de deux semaines, durant cette période nous avons contrôlé la démarche de différentes parties du système et nous avons lancé quelques attaques d'une troisième machine (*Backtrack*).

Note : nous avons mis le système en exécution avec une base d'information nulle sans aucune information recueillie en avance c'est-à-dire avec des fichiers XML vides. Par contre *Snort* a commencé avec une collection des règles se trouvant sur plusieurs fichiers **.rules*.

V.6 Analyse des résultats obtenus

Durant la période d'exécution, le *ProbSys* a généré 55 alertes par contre *Snort* a généré 61 alertes, le module d'affichage intégré dans le *ProbSys* offre les deux graphes suivants :

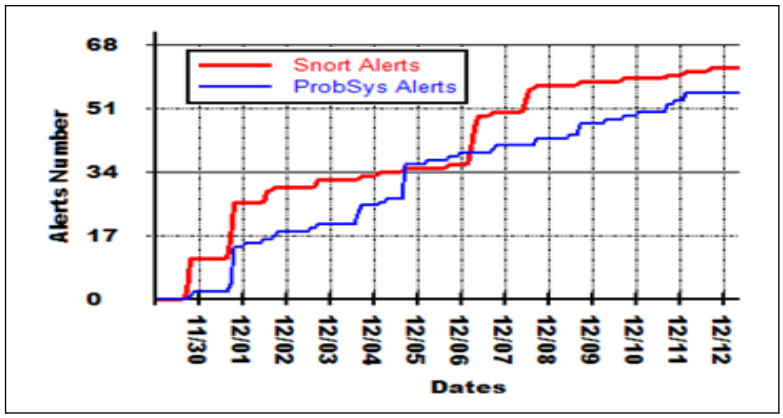


Figure 32 : Graphe cumulatif des alertes générées par *ProbSys* et *Snort*

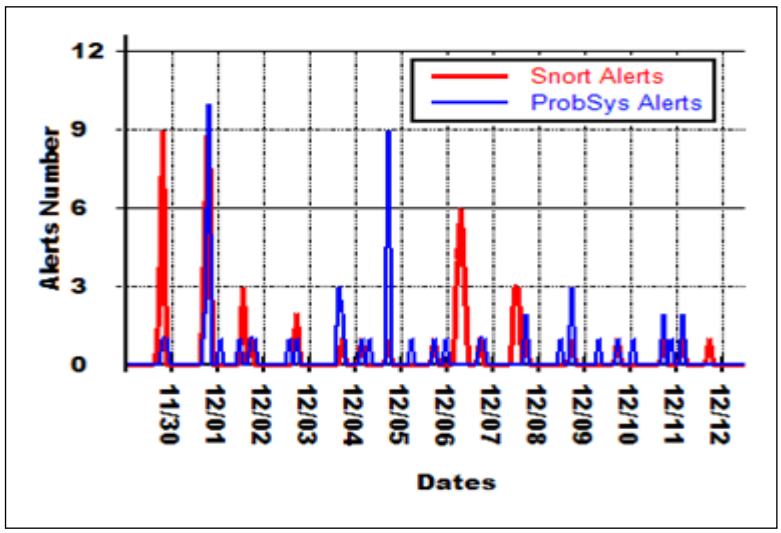


Figure 33 : Graphe comparatif des alertes générées par *ProbSys* et *Snort*

Les résultats illustrés dans les deux graphes précédents montrent clairement l'efficacité de notre système, il est capable de détecter les mêmes attaques vues par *Snort* même s'il commence avec zéro information. Au début, le nombre des attaques vues par notre système est faible, car il est en état d'apprentissage ce qui agrandit le nombre des *faux négatifs*. Néanmoins, par le temps, *ProbSys* commence à augmenter sa base d'information, son pouvoir s'améliore de plus en plus jusqu'à ce qu'il devient capable de capter la majorité des attaques vues par *Snort*, et il serait capable de détecter d'autres que ce dernier ne les détecte pas. Cette augmentation montre aussi comment le taux des *faux négatifs* se réduit. La figure 33 prouve clairement ce que nous venons de le dire, regarder le nombre des attaques au 1er, le 5 et entre le 8 et le 11 décembre. Au premier et le 5 décembre nous avons attaqué le *HoneyLens* par des attaques de scans en utilisant le *Backtrack*, puis nous avons fait les mêmes attaques contre la

machine où *Snort* est installé, le *ProbSys* a généré le même nombre d'alertes que *Snort* à la première date, par contre ce dernier n'a déclenché qu'une seule alerte sur 9 générées par *ProbSys*. La figure 33 montre aussi que la répartition des alertes générées par le *ProbSys* et presque équivalentes à celle des alertes de *Snort* ce qui prouve que ces alertes sont vrais d'où le manque des *faux positifs*.

V.7 Discussion

Les résultats interprétés au-dessus sont obtenus à partir d'un système qui a commencé avec zéro information : les fichiers XML ont été vides et nous avons lancé le *ProbSys* et le *HoneyLens* au même temps; par contre *Snort* a commencé avec des centaines de règles prédéfinies et elles sont ajoutées manuellement par *Snort Team* ou peuvent être mises à jour par un administrateur. Ces règles sont téléchargées du site web de *Snort*, elles sont communes pour n'importe quel type de réseau et pour tous les différents types de serveurs. Le mis à jour et la configuration pour qu'il s'adapte à un tel réseau doit être faite par des spécialistes et doit être faite manuellement.

Snort a une vision microscopique, les règles sont écrites en se basant sur le contenu des paquets, de plus il ne détecte que quatre types de protocoles (TCP, UDP, ICMP et IP). Ces caractéristiques limitent l'efficacité de *Snort* et augmentent le taux des *faux négatifs*. Par contre notre système n'a aucune de ces limites mentionnées au-dessus, c'est un système à apprentissage automatique et plus précis que *Snort*.

En général, notre système se distingue de *Snort* et de n'importe quel autre système de détection d'intrusion par les points suivants :

- La structure arborescente qui maintient les informations sous ce format est la bonne façon de sauvegarder ce type d'information, car les séquences des attaques ne contiennent pas les mêmes éléments et n'ont pas les mêmes longueurs. Cette structure facilite les calculs statistiques et peut être sauvegardée sous un fichier XML qui est simple à lire, à parcourir et à transporter.
- L'ajout des informations par le *HoneyLens* peut se faire de deux façons : par la lecture directe à partir de la carte réseau (*sniffing*) ou par l'ouverture d'un fichier **.pcap* contenant une collection de trafic sur des vraies attaques. Donc, il est possible de ne

pas commencer par zéro information au début, il suffit d'ouvrir des fichiers *.*pcap* contenant des informations sur des trafics venants d'autres *pots de miel*.

- Notre système est complètement autonome, la mis à jour de sa base de connaissance se fait de manière automatique, aucune intervention humaine est demandée à moins que pour ajouter un fichier *.*pcap*, pour le faire le *HoneyLens* est capable de faire tout le processus d'analyse et de distinction des différents éléments des attaques.
- Des paquets peuvent avoir un comportement normal dans un trafic non suspect, mais à partir d'un trafic de scan par la commande ***nmap***, d'une force brute ou de déni de service ils peuvent être vus et interprétés de manières différentes. Exemple : après avoir analysé un trafic de scan, nous avons noté que la majorité des paquets [SYN] sont suivis par des parquets [SYN, RST], le nombre de ces séquences peut aller jusqu'aux centaines ou milliers. Donc, l'écriture d'une règle *Snort* en se basant sur le contenu de ces paquets n'est pas recommandé, par ce qu'il va augmenter le taux des faux positifs.
- Notre système est entièrement flexible, adaptable et simple à la configuration, si une compagnie veut protéger quelques services contre les attaques et les intrusions, en utilisant notre système, il suffit d'installer dans les *pots de miel* les mêmes services pour avoir une protection parfaite. Par contre pour *Snort* il suffit d'écrire les règles qui conviennent à ces derniers, ce qui demande un administrateur compétent et qui maitrise bien *Snort* et le service à protéger. L'auto-apprentissage, la précision pour la prédiction et la prévention des prochains éléments devient de plus en plus exacte et s'accroit proportionnellement avec le nombre des attaques trouvées, au contraire le nombre des *faux négatifs* et des *faux positifs* sera très réduit.

Contrairement à *Snort*, notre système peut réagir différemment en allant d'un environnement à un autre, car le trafic qui traverse un réseau n'est pas nécessairement le même pour tous les autres. Il est recommandé de ne pas commencer à zéro information, mais il est préférable de commencer avec des fichiers XML qui contiennent quelques informations venants des fichiers *.*pcap* ou par un autre fichier XML ayant ces informations.

CONCLUSION GÉNÉRALE

De nos jours plusieurs technologies sont développées pour assurer la sécurité des systèmes informatiques. Chacune de ces technologies a ses avantages et ses inconvénients et les utilisateurs malveillants essaient toujours d'exploiter leurs faiblesses pour réaliser des attaques informatiques. Les faiblesses de ces outils de ne sont pas les mêmes, ils diffèrent d'un système à l'autre, chacun d'eux seul ne peut donc pas assurer la sécurité de son environnement.

La faiblesse majeure des IDSs est le contenu et la mise à jour de sa base d'informations, car il demande un niveau de compétence très important pour intervenir au bon moment et avec l'information exacte. D'un autre coté, les *honeypots* sont les meilleurs outils pour fournir des informations approfondies et détaillées sur les attaques informatiques. Malheureusement, ce outil est passif et ne peut prévenir ces attaques. Plusieurs travaux ont, donc, investi la problématique de collaboration ces deux technologies pour avoir un système fiable et efficace de détection et de prévention des attaques informatiques. Notre contribution s'inscrit dans cette voie. Pour cela, nous avons présenté une approche d'un système quantitatif probabiliste pour la détection et la prévention des intrusions. Un pot de miel est utilisé comme source d'information sur les attaques, le module *HoneyLens* s'occupe de la lecture et l'analyse du trafic pour augmenter d'une manière autonome la base d'information de notre IDS *ProbSys* avec les bonnes informations. Le module *ProbSys* est un IDS flexible et adaptable et qui fonctionne de manière complètement automatique et autonome. *ProbSys* effectue des calculs statistiques probabilistes qui se basent sur des approches de *Data Mining*, permettant de prévenir la prochaine action élémentaire la plus probable et détecter des attaques avant leurs complétions (pouvoir de prévention).

RÉFÉRENCES

[1]. L. Zpitzner, Honeypots: Tracking Hackers, *Addison Wasley Professional, ISBN-10: 0321108957*, (septembre 2002).

[2]. Canadian Honeypot Security Research, http://www.honeynet.ca, Consulté le 23 mars 2008.

[3]. Security Wizardry, http://www.networkintrusion.co.uk/honeypots.htm, Consulté le 1er avril 2008.

[4]. A. Boulaiche, Technologies Honeypots, Mémoire de maîtrise, Université Abderrahmene Mira de Béjaia, (2008).

[5]. Know your enemy, http://www.honeynet.org, Consulté le 2 avril 2007.

[6]. T. Holz, F. Raynal, Detecting honeypot and other suspiciouus environments, *Workshop on Information Assurance and Security, United States Military Academy, West Point, NY*, (15-17 juin 2005).

[7]. M. Dornseif, T. Holz, C. Klein, NoSEBrEaK – Attacking Honeynets. *Proceedings of the 2004 IEEE: Workshop on Information Assurance and Security, West Point, NY*, (7-8 juin 2004).

[8]. J. Göbel, J. Holds, T. Holz, Advanced honeypot-based intrusion detection. http://www.usenix.org/publications/login/2006-12/pdfs/goebel.pdf et https://pi1.informatik.uni-mannheim.de/filepool/publications/honeypot-ids_pre-crc.pdf Consulté le 13 juin 2008.

[9]. K. Iyad, S. Malek, A Dynamic honeypot Design for Intrusion Detection. *IEEE ACS/ International, ISBN: 0-7803-8577-2, Washington, DC, USA*, (juillet 2004).

[10]. G. Portokalidis, H. Bos, SweetBait: Zero-Hour Worm Detection and Containment Using Honeypots, *Journal on Computer Networks, Special Issue on Security through Self-Protecting and Self-Healing System*, (2007).

[11]. D. David, Q. Xinzhou, G. Guofei, L. Wenke, G. Julian, L. John, O. Henry, HoneyStat: Local Worm Detection Using Honeypots, *In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID) page 39-58*, (2004).

[12]. G. Wicherski, Medium Interaction Honeypots, *German Honeynet Project* (avril 2006).

[13]. G. Held, Ethernet Netwok: Design, *Implementation, Organization & Management, Wiley; 4 edition, ISBN-13: 978-0470844762* (septembre 2002).

[14]. K. Burns, TCP/IP Analysis and Troubleshooting Toolkits. *Wiley; 1st edition, ISBN-13: 978-0471429753*, (juillet 2003).

[15]. A. Raven, B. Jacob, D. Adam, C. F. James, K, Toby, R. Michael, Snort 2.1 Intrusion Detection, *Syngress second edition, ISBN-13: 978-1931836043*, (juin 2004).

[16]. M. Wood and M. Erlinger, Intrusion detection message exchange requirements, sur le site web The RFC Archive: http://www.rfc-archive.org/getrfc.php?rfc=4766, Consulté le 19 mai 2010.

[17]. H. Lamia, Génération automatique de scénario d'attaques pour les systèmes de détection d'intrusion, *Mémoire de maîtrise Béjaia, Université Abderrahmene Mira de Béjaia, Algérie*, (2005).

[18]. J.P. Anderson, Computer Security Threat Monitoring and Surveillance, *Technical report, James P. Anderson Company, Fort Washington, Pennsylvania*, (1980).

[19]. S. Riebach, B. Toedtmann, E. Rathgeb. Combining IDS and Honeynet Methods for Improved Detection and Automatic Isolation of Compromised Systems, *Computer Networking Technology Group, Institute for Experimental Mathematics, University Duisburg-Essen, Germany*, (2006).

[20]. M. Benjamin, Corrélation d'alertes issues d'outils de détection d'intrusions avec prise en compte d'informations sur le système surveillé. *Thèse de doctorat, Institut National des Sciences Appliquées de Rennes. Rennes, France*, (février 2004).

[21]. G.Jakobson, M.D. Weissman, Alarm Correlation and Network Fault Resolution using the Kohonen Self-organising Map, *Global Telecommunications Conference, GLOBECOM'97 IEEE, pages: 1398 – 1402 vol.3, ISBN: 0-7803-4198-8*, (1997).

[22]. T.R. Gruber, Toward Principles for the Design of Ontologies Used for Knowledge Sharing, *International Journal of Human-Computer Studies, 43, 907 – 928*, (1995).

[23]. J. Tian, J. Wang, X. Yang, R. Li, A Study of Intrusion Signature Based on Honeypot, *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05), pages 125 – 129*, (2008).

[24]. C, Chi, M. Li, D. Liu, A Method to Obtain Signatures from Honeypot Data, *Lecture Notes in Computer Science, Volume 3222/2004, 435-442, DOI: 10.1007/978-3-540-30141-7_61*, (2004).

[25]. S. Singh, C. Estan, G. Varghese, S. Savage, The EarlyBird System for Real-time Detection of Unknown Worms, *Technical Report CS2003-0761, CSE Department*, (août 2003).

[26]. C. Kreibich, J. Crowcroft, Honeycomb – Creating Intrusion Detection Signatures Using Honeypots, *ACM SIGCOMM Computer Communication Review, 34, 51 – 56*, (2004).

[27]. S. S. Muhammad, S. H. Choong, A Novel Architecture for Real-time Automated Intrusion Detection Fingerprinting using Honeypot, *27th KIPS Spring Conference, Korea, pp.1093-1095*, (mai 2007).

[28]. H. Richard, Enhancing IDS using Tiny Honeypot, *SANS Institute InfoSec Reading Room*, (novembre 2006).

[29]. S. Khamadja, Élaboration de techniques d'exploration passive des réseaux et systèmes informatiques, *Faculté des sciences exactes, Université Abderrahmane Mira Béjaia, Algérie*, (2008).

[30]. B, Zhu, A. Ghorbani, Alert Correlation for Extracting Attack Strategies, *International Journal of Network Security, Volume: 3, Issue: 3, Pages: 244-258*, (2006).

[31]. S. Noel, S. Jajodia, B. O'Berry, M. Jacobs, Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs, *ACSAC'03 Proceedings of the 19th Annual Computer Security Applications Conference, IEEE Computer Society Washington, DC, USA, ISBN:0769520413*, (2003).

[32]. K. Ingols, R. Lippmann, K. Piwowarski, Practical Attack Graph Generation for Network Defense, *ACSAC'06 Proceedings of the 22nd Annual Computer Security Applications Conference , 121-130*, (2006).

[33]. J. M. Wing, Scenario Graphs Applied to Network Security, *Information Assurance: Survivability and Security in Networked Systems , Chapter 9 , Yi Qian, James Joshi, David Tipper, and Prashant Krishnamurthy, editors, Morgan Kaufmann Publishers, Elsevier, Inc., pp. 247-277*, (2008).

[34]. V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time, *Computer Networks, Proceedings of the 7th USENIX Security Symposium San Antonio, Texas*, (janvier 1998).

[35]. [35] W. W Cohen. Fast effective rule induction. *In A. Prieditis and S. Russell (Eds.), The 12th International Conference on Machine Learning, Tahoe City, CA, pp. 115123. Morgan Kaufmann.* (9-12 juillet 1995).
[36]. [36] S. J. Stolfo Lee, W. and K. W. Mok. Adaptive intrusion detection: A data mining approach,. *Artificial Intelligence Review 14 (6), 533567*. (2000).
[37]. [37] F. Neri. (2000). Comparing local search with respect to genetic evolution to detect intrusion in computer networks. *The 2000 Congress on Evolutionary Computation CEC00*, La Jolla, CA, pp. 238243. IEEE Press, (16-19 juillet, 2000).
[38]. [38] F. Neri. Mining tcp/ip traffic for network intrusion detection. *M'antaras and E. Plaza (Eds.), Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Volume 1810 of Lecture Notes in Computer Science, Barcelona, Spain, pp. 313322. Springer*, (31 mai- 2 juin, 2000).
[39]. [39] D. Dasgupta and F. A. Gonzalez. An intelligent decision support system for intrusion detection and response. *International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS), St.Petersburg. Springer-Verlag,*, (21-23 mai, 2001).
[40]. [40] A. Chittur. Model generation for an intrusion detection system using genetic algorithms. *High School Honors Thesis, Ossining High School. In cooperation with Columbia Univ*, (2001).
[41]. [41] M. Crosbie and E. H. Spafford. Active defense of a computer system using autonomous agents. *Technical Report CSD-TR- 95-008, Purdue Univ., West Lafayette, IN*, (15 février 1995).
[42]. [42] J. E. Dickerson and J. A. Dickerson. Fuzzy network profiling for intrusion detection. *NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, Atlanta, pp. 301306. North American Fuzzy Information Processing Society (NAFIPS)*, (juillet 2000).
[43]. [43] Vaughn RB G. Florez, SM. Bridges. An improved algorithm for fuzzy data mining for intrusion detection. *Annual Meeting of The North American Fuzzy Information Processing, Society Proceedings NAFIPSFLINT 2002 Cat No 02TH8622, (2002)* .
[44]. [44] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, (1948).
[45]. [45] Robert L. Solso. Bigram and trigram frequencies and versatilities in the english language. *In Behavior Research Methods & Instrumentation, volume 11(5), pp 475-484*, (1979).
[46]. [46] Wen Wang et Dimitra Vergyri. The use of word ngrams and parts of speech for hierarchical cluster language modeling. *In ICASSP 2006, pp 1057-1060*, (2006).
[47]. [47] H. Lei et N. Mirghafori. Word-conditioned phone n-grams for speaker recognition. *Proc. ICASSP, Honolulu,* E-ISBN: 1-4244-0728-1, (15 – 20 avril 2007).