UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté Science de l'Ingénieur

Département d'Electronique

THESE DE DOCTORAT

Spécialité : Electronique

RESEAUX BAYESIENS POUR LA DETECTION D'INTRUSIONS DANS UN RESEAU INFORMATIQUE

par

Mr Merouane MEHDI

devant le jury composé de :

A. GUESSOUM	Professeur, U. de Blida	Président
S. OUKID	Maître de Conférences, U. de Blida	Examinateur
M. IOUALALEN	Professeur, U.S.T.H.B.	Examinateur
M. GUERTI	Maitre de Conférences, ENP	Examinateur
M. HALIMI	Maitre de Recherche, CSC	Examinateur
M. BENSEBTI	Professeur, U. de Blida	Rapporteur

Blida, Avril 2010

RESUME

L'accroissement du nombre de réseaux interconnectés à l'Internet provoque une augmentation des menaces de sécurité et des cyber-crimes. Nous nous intéressons à des problèmes sensibles rencontrés par la détection d'intrusions à savoir les nombreuses fausses alertes. Ce problème fait qu'il est extrêmement difficile de trouver dans la masse des activités douteuses, les tentatives de compromissions réelles ou autres activités malhonnêtes. A cet effet, notre objectif s'inscrit dans le cadre d'une détection d'intrusions probabiliste basée sur l'apprentissage de réseaux bayésiens afin de détecter précocement des attaques réseaux. Notre architecture de filtrage d'alarmes est basée sur une approche de classification bayésienne permettant de déterminer si le réseau est réellement attaqué. L'idée est de construire un modèle de référence et d'utiliser l'apprentissage à base du réseau bayésien associé à l'algorithme d'apprentissage, pour comparer le comportement temps réel avec celui de référence. La validation des résultats, l'implémentation et le test sont réalisés sur *Snort* open source software, afin de montrer comment utiliser efficacement ce genre de modèle dans le cadre de diagnostic.

Mots Clés : Détection d'Intrusion, Apprentissage automatique, Classification, Réseaux Bayésiens, Snort.

ABSTRACT

The increase in the number of interconnected networks to the Internet has led to an increase in security threats and cyber-crimes. Intrusion detection systems (IDSs) have been widely used to overcome security threats in computer networks. Anomaly-based approaches have the advantage of being able to detect previously unknown attacks, but they suffer from the difficulty of building robust models of acceptable behavior which may result in a large number of false alarms caused by incorrect classification of events in current systems. The work presented here describes the use a learning bayesian networks in order to detect distributed network attacks as early as possible. This work indicates how probabilistically Bayesian network detects communication network attacks, allowing for generalization of Network Intrusion Detection Systems (NIDSs). It consists of building a reference behaviour model and the use of a Bayesian classification procedure associated to learning algorithm to evaluate the deviation between current and reference behaviour. The accuracy of the event classification process is significantly improved using our proposed approach for reducing the missing-alarm, Experimental results in Snort Open Source show that the accuracy of the event classification process is significantly improved using our proposed approach.

KeyWords : Intrusion detection, Machine Learning, Classification, Bayesian Networks, Snort.

REMERCIMENT

La réalisation d'un doctorat est une tranche de vie à part entière qui laisse paraître un profond égoïsme en fin de parcours mais qui s'est nourrie, tout au long, de nombreuses et diverses influences. Beaucoup de personnes participent directement ou indirectement à sa concrétisation et méritent d'être remerciés quel qu'en soit leur degré d'investissement. Malheureusement, ici tout le monde ne peut apparaître nommément et je me dois de faire une sélection subjective.

D'un point de vue professionnel, je commencerai par remercier mon directeur de thèse le Professeur ***BENSEBTI Messaoud*** *pour le travail qu'il a effectué à mes côtés et qui était emprunt tout autant de conseils pour orienter mes recherches que de critiques nécessaires à la pertinence de mon étude.*

Ensuite, je souhaite remercier mes rapporteurs pour avoir accepté d'évaluer mon travail et pour les commentaires qu'ils ont formulés afin de le perfectionner et qui mon permis de considérer mes travaux sur des angles nouveaux. Je tiens aussi à exprimer ma gratitude envers le professeur ***GUESSOUM Abderrazak et Mme OUKID Saliha****, qui ont eu la gentillesse de lire l'intégralité de mon manuscrit et qui ont usé de leur expertise pour déceler certains aspects qui m'avaient échappés.*

Je tiens aussi à remercier les nombreux collègues qui ont partagé mon quotidien. En particulier, l'ensemble des membres du Centre réseaux, systèmes et téléenseignement ainsi que les enseignants du département d'électronique.

D'un côté plus personnel, je ne saurais montrer trop de reconnaissance aux membres de ma famille pour leur soutien inconditionnel durant toutes ces années.

M.MEHDI

TABLE DES MATIERES

INTRODUCTION GENERALE

1. Introduction

De nos jours les entreprises s'appuient de plus en plus sur Internet pour fournir leurs services et coopérer avec d'autres compagnies. Les réseaux sont devenus alors des ressources vitales et déterminantes pour le bon fonctionnement des entreprises, mais l'ouverture facile au monde extérieur via des réseaux connectés à Internet rend l'entreprise plus vulnérable aux attaques. Ces attaques peuvent avoir de graves conséquences comme en témoigne ces dernières années. Par exemple en mars 1997 un pirate suédois a désactivé le système d'urgence 911 en Floride [1], onze états ont été touchés. Un autre évènement marquant est l'attaque par déni de services répartis (DoS) qui a paralysé en février 2006 plusieurs sites web populaires dont CNN, Amazon.com, Yahoo ! et eBay [1].

D'après une étude faite à l'Université de Carnegie Mellon [2], les programmes d'attaques deviennent de plus en plus dangereux et nécessitent moins d'expertise pour leur utilisation, ce qui expose les entreprises à des menaces d'intrusions supplémentaires. La figure 1 montre cette évolution rapide, qui aux années 80, concerne des outils développés par des attaquants programmeurs alors que de nos jours, elle repose sur des outils automatiques à la portée des utilisateurs ordinaires. Ceci encourage les gamins scripteurs (Script-kiddies) à employer aveuglément les outils d'attaques sans pour autant avoir la maîtrise des programmes et la conscience des dégâts causés sur le réseau cible.

En outre, les attaquants profitent des logiciels point-à-point (Pear to Pear) en pleine expansion tels que Kazaa, Napster, eMule et plein d'autres. Ces utilitaires ouvrent des chemins directs pour accéder à des machines distantes appartenant au réseau cible, de plus ces utilitaires contribuent à la propagation rapide des virus et des vers sur les réseaux d'entreprises et facilite l'échange de connaissances et d'outils d'attaques entre les intrus.

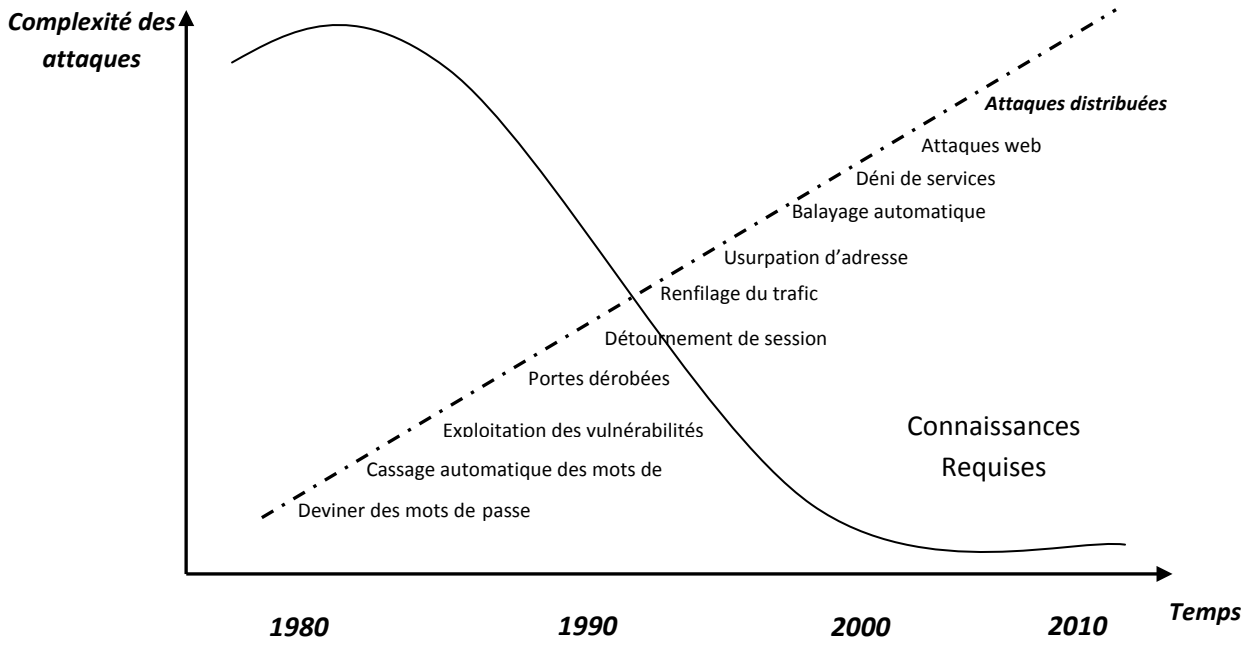


Figure 1 : Evolution des techniques d'attaques. **[10]**

Par conséquent sécuriser les accès réseaux, les données confidentielles et les serveurs devient un des premiers soucis de l'entreprise. Les dirigeants investissent de plus en plus afin de mieux protéger les réseaux informatiques. Ils acquièrent les nouvelles technologies de routeurs et de pare-feux aux coûts élevés. Ils s'intéressent également à la détection d'intrusions pour déceler les attaques à l'entrée du réseau et saisir les intrusions à l'intérieur du réseau local ou celles passées inaperçues à travers d'autres outils de sécurité. En effet, les systèmes de détection d'intrusions (SDI) analysent en permanence les données disponibles sur le système informatique. Ils découvrent les scénarios d'attaques et les exploitations non conformes du système informatique. Cette surveillance contribue à éviter le renouvellement des attaques en bloquant les sources d'attaque et en corrigeant les vulnérabilités du système.

Les SDIs sont déployés dans des zones précises du réseau ou sur des machines particulières pour compléter le travail des pare-feux [3] et détecter les attaques passées inaperçues. Considérés comme une dernière barrière de sécurité, les IDS « Intrusion Detection System » sont capables de comprendre la nature et les caractéristiques du trafic réseau afin de mieux détecter les intrusions.

2. Problématique

Le problème principal que découvrent les administrateurs lorsqu'ils installent des IDS est la quantité importante d'alertes générées. Sur un réseau de taille moyenne, plusieurs milliers d'alertes sont générées quotidiennement, rendant quasiment impossible l'exploitation des résultats. La conséquence est que l'administrateur est obligé de revoir sérieusement à la hausse son seuil de tolérance, ce qui va le conduire à passer à côté de beaucoup de problèmes réels et permettre à un pirate de haut niveau de réussir une attaque suffisamment discrète pour ne pas être détectée du tout. Ainsi le principal problème d'un IDS n'est pas de laisser passer certaines attaques, mais de noyer l'administrateur sous un flot d'information. En effet, l'IDS n'est pas capable de juger de la pertinence, de la gravité et de la corrélation des attaques. Il génère tellement d'alertes qu'il va être très difficile de détecter les problèmes graves au milieu de toutes les alarmes.

En effet, les IDS génèrent un grand nombre de faux positifs ''occurrence d'alerte en absence d'attaque'' (Figure 2) causé généralement par le modèle de référence dans le cas d'une détection par anomalie (comportemental). Le problème est difficile à gérer d'autant plus qu'une politique trop restrictive de déclenchement des alertes peut manquer de véritables attaques ce qui mène au problème inverse de faux négatif ''occurrence d'attaque en absence d'alerte''. Par suite, une analyse améliorée du trafic est indispensable pour augmenter les possibilités de détection et réduire le nombre de fausses alertes.

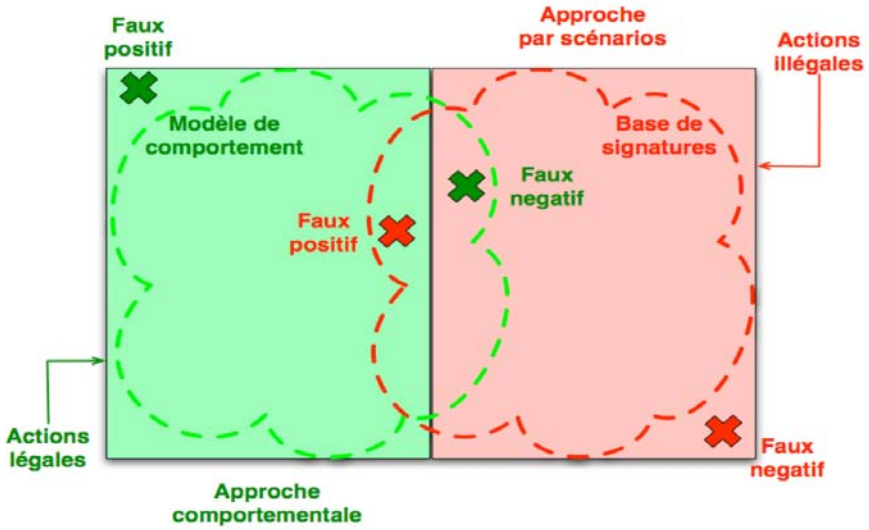


Figure 2 : IDS, Faux positif et faux négatif. [12]

3. Contribution

Afin de remédier aux différents problèmes évoqués plus haut, la détection d'intrusions doit s'orienter vers de nouvelles stratégies pour mieux assurer la sécurité des réseaux. Nous proposons ici un nouveau concept de détection d'intrusion d'anomalie pour le filtrage d'alarmes basé entièrement sur un modèle probabiliste et plus précisément sur la classification Bayésienne.

Le but de notre système est de partir des alarmes générées par un HIDS (Host Intrusion Detection System), et d'essayer de filtrer les alarmes pour déterminer s'il y a eu une attaque sur la machine. En considérant qu'un scénario d'attaque consiste en une série d'évènements se déroulant dans un intervalle de temps. Ces évènements nous donnent un résumé sur le comportement de toutes les machines externes (attaquantes) à destination de toutes les adresses IP internes (attaquées). Nous partons ensuite du principe que ce comportement peut être similaire pour plusieurs machines externes (qui tenteraient le même genre d'attaque vers une même machine interne), ou à destination de plusieurs machines internes. Nous allons donc regrouper ces comportements en un certain nombre de comportements-types, en utilisant une technique de classification bayésienne.

En fait les réseaux bayésiens permettent de modéliser des situations dans lesquelles la causalité joue un rôle, mais où la connaissance de l'ensemble des relations entre les phénomènes est incomplète, de telle sorte qu'il est nécessaire de les décrire de manière probabiliste. Dans ce modèle, les attaques connues constituent des hypothèses qui peuvent expliquer les faits observés. On considère qu'une attaque donnée se traduit par des symptômes, pouvant apparaître sous forme d'évènements dans l'audit, mais aussi par des données statistiques comme dans le cas de la détection d'anomalies (occupation mémoire, charge CPU, etc.). L'utilisation d'outils de raisonnement probabiliste comme les réseaux bayésiens devra être efficace pour détecter des problèmes réels.

Dans cet esprit, nous proposons de développer un modèle probabiliste basé sur la classification bayésienne pour la détection d'intrusion en se basant sur l'estimation des densités de probabilité de classes pour avoir un bon taux de reconnaissance. L'application du Modèle de Mélange Gaussien (GMM) avec l'algorithme Expectation/Maximization (EM) en utilisant la librairie « ***Bayesian Network tools*** in Java (***BNJ) »*** pour la détection d'intrusion, nous permettra de vérifier l'efficacité de cette méthode.

L'ensemble des algorithmes seront adaptés et implémentés dans le logiciel ***Snort open source*** pour la phase de test et de validation des résultats, le but est d'améliorer le système de diagnostic de celui-ci.

4. Organisation de la thèse

Ce mémoire est organisé de la manière suivante :

- ✓ Dans le premier chapitre, nous dressons un état de l'art du domaine de la sécurité informatique. Ce premier chapitre s'intéresse aux menaces et techniques de protection contre les attaques réseaux.
- ✓ Dans le deuxième chapitre, nous nous intéressons à l'évolution de la sécurité informatique dans le domaine de la détection d'intrusions. Tout d'abord, nous présentons une classification des systèmes de détection d'intrusions. Nous passons ensuite en revue les différentes approches développées avec leurs limites.
- ✓ Dans le troisième chapitre, nous présentons la terminologie utilisée tout le long travail, relative à la classification bayésienne. Quelques notions sur le calcul probabiliste et les réseaux bayésiens seront également rappelées.
- ✓ Le quatrième chapitre s'intéresse à l'implémentation d'un prototype et on va se focaliser sur la notion de la classification a base de réseaux bayésiens, nous détaillons la mise en œuvre du système, objet de ce travail. Ainsi on va aborder la procédure d'apprentissage utilisée, pour présenter par la suite quelques statistiques et manipulations sur les données qu'on va utiliser pour mener cette procédure. On va donner aussi les différentes formules et algorithmes utilisés pour l'apprentissage de la structure et des paramètres des réseaux Bayésiens.
- ✓ Le cinquième et dernier chapitre portera sur la partie test et validations des résultats avec une description détaillée du logiciel *Snort* open source IDS. Ce chapitre présentera les résultats de performance du système de filtrage des alertes dans différentes configurations, testé sur des données réelles (trafic temps réel).

Enfin, il nous restera à revenir sur le travail réalisé et envisager les pistes de recherche qui permettraient d'améliorer notre système.

CHAPITRE 1
Menaces des Attaques Réseaux : Techniques de Protection

1.1 Introduction

La sécurité informatique constitue la réplique des administrateurs pour contrarier les attaques couramment menées sur les réseaux informatiques. Les systèmes de sécurité essayent de prévenir de telles attaques et de corriger les vulnérabilités exploitées lors des dernières intrusions. Il est alors nécessaire pour sécuriser le réseau d'entreprise d'identifier les menaces potentielles et de connaître les procédés des attaquants. La compréhension de l'activité intrusive et des techniques utilisées permet de définir des méthodes efficaces de surveillance et de détection des attaques.

Nous consacrons la première partie de ce chapitre à ausculter l'anatomie d'une attaque puis nous expliquerons un scénario possible pour attaquer le réseau de l'entreprise. Ensuite nous présentons les solutions possibles de protection du système informatique. Nous analysons les limites de ces techniques ce qui permet de souligner la complexité de la prévention complète des attaques. D'ou l'intérêt de la détection d'intrusions pour retrouver les mauvaises exploitations du système à la recherche de nouvelles brèches qui mènent à la violation de la politique de sécurité de l'entreprise.

1.2 Attaques Informatiques

Dans sa plus large définition, une attaque informatique est toute activité malveillante qui vise un système informatique ou bien les services qu'il fournit. Des exemples d'attaques sont les virus, l'utilisation du système par un individu non autorisé, les dénis de service par l'exploitation d'un bug ou d'un abus d'un dispositif, l'envoie des sondes vers un système pour recueillir des informations concernant ce dernier, et les attaques physiques contre le matériel d'un poste de travail.

1.2.1. Stratégies d'attaques

Dans ce qui va suivre on va présenter quelques exemples des stratégies avec lesquelles un pirate informatique peut avoir accès à un système ou nie l'accès légitime pour les autres utilisateurs.

- ☒ *Ingénierie Sociale* : Un attaquant peut atteindre un système en dupant un utilisateur légitime pour qu'il fournisse les informations d'accès à ce dernier. Par exemple, un attaquant peut appeler un individu par téléphone personnifiant l'administrateur réseau afin d'essayer de convaincre l'individu à indiquer des informations confidentielles tels les mots de passe, les noms de dossiers, des détails au sujet des politiques de sécurité, etc. Un attaquant peut aussi fournir une application logicielle à un utilisateur d'un système qui est en réalité un Cheval de Troie contenant un code malveillant qui donne au pirate l'accès au système.
- ☒ *Bugs d'Implémentation* : Des bugs dans des programmes s'exécutant sur un système informatique peuvent être exploités par un pirate pour avoir un accès non autorisé à ce dernier. Des exemples spécifiques de bugs d'implémentation sont les débordements des mémoires tampons, les conditions concurrentielles, et la mauvaise manipulation des fichiers temporaires.
- ☒ *Abus de Dispositifs* : Il y a des actions légitimes qu'on peut exécuter une fois qu'on est arrivé à l'extrémité du fonctionnement d'un dispositif et peuvent mener à l'échec du système. Des exemples incluent l'ouverture des centaines de connexions TELNET vers une machine pour remplir sa table de processus, ou remplir le spool du courrier électronique par des emails indésirables.
- ☒ *Mauvaise Configuration d'un Système* : Un pirate peut avoir l'accès en exploitant une erreur dans la configuration d'un système. Par exemple, la configuration par défaut de quelques systèmes inclut un compte « *invité* » qui n'est pas protégé par un mot de passe.
- ☒ *Déguisement* : Dans certains cas, il est possible de duper un système pour avoir l'accès avec une fausse identification. Un exemple est l'envoi d'un paquet TCP qui a une adresse source forgée, avec laquelle le paquet semblera venir d'un hôte de confiance.

1.2.2. Anatomie d'une Attaque

Le squelette d'une attaque informatique est constitué de cinq composantes reconnues généralement dans la littérature par les *« Cinq P »*, Probe, Penetrate, Persist, Propagate, Paralyze [4]. Dans la suite on va analyser le détail de chaque composante.

- ☒ *Probe :* C'est la phase de la collecte d'informations à partir du système cible, qui peut s'effectuer de plusieurs manières comme le scan des ports pour déterminer la version des logiciels utilisés, ou bien le scanne des vulnérabilités. Par exemple, pour les serveurs web, il existe des outils qui permettent la recherche des failles ou les problèmes connus de sécurité.
- ☒ *Penetrate :* Consiste en l'utilisation des informations récoltées pour s'infiltrer dans un système, en outrepassant par exemple les protections par mot de passe en utilisant des failles applicatives.
- ☒ *Persist :* C'est l'étape de la création d'un compte avec des droits de super utilisateur pour pouvoir s'infiltrer ultérieurement, en installant par exemple une application de contrôle à distance.
- ☒ *Propagate :* Consiste à observer ce qui est accessible et disponible sur le réseau local pour se propager dans ce dernier.
- ☒ *Paralyze :* Cette phase est généralement constituée de plusieurs actions. Le pirate peut par exemple utiliser le serveur pour mener des attaques sur d'autres machines, détruire les données ou bien encore endommager le système d'exploitation.

1.2.3. Classification des Attaques

Plusieurs classifications des attaques ont été proposées selon plusieurs critères tels la gravité de l'attaque, les vulnérabilités exploitées par cette dernière, et la manière de procéder. La classification la plus utilisée dans la littérature est celle adoptée par le *Massachusetts Institute of Technology (MIT),* qui comporte cinq classes d'attaques et qu'on va présenter dans ce qui va suivre.

a. Dénis de Service (Denial Of Service DOS)

Un déni de service est une attaque dans laquelle le pirate rend certaines ressources du traitement ou du stockage indisponibles ou bien trop occupées pour pouvoir répondre aux demandes des utilisateurs légitimes d'une machine. Il y a plusieurs types d'attaques par déni de

service *(DOS)* [5]. Certaines attaques comme *« mailbomb, neptune, ou smurf »* abusent parfaitement les dispositifs légitimes. D'autres *« teardrop, ping of death »* créent des paquets mal formés qui confondent la pile TCP/IP de la machine cible, cette dernière va essayer de reconstruire ces paquets par la suite. Encore d'autres *« apache2, dos, syslogd »* tirent profit des bugs dans le réseau.

b. Attaques Utilisateur vers Administrateur (User To Root U2R)

Sont une classe d'exploit dans laquelle l'attaquant commence par un accès à un compte utilisateur normal sur le système (obtenu en sniffant les mots de passe, en utilisant par exemple l'ingénierie sociale), ensuite il essaie d'exploiter la vulnérabilité sur ce système pour obtenir un accès administrateur. Il y a plusieurs types d'attaques U2R [5], La plus commune est l'attaque *«buffer overflow»* qui se produit quand un programme copie beaucoup de données dans une zone mémoire tampon statique sans vérifier si la taille du cette dernière est suffisante, ce qui provoquera un débordement. Les données débordées seront stockées dans la pile du système, recouvrant ainsi les prochaines instructions qui devaient être exécutées. En manipulant soigneusement les données qui débordent sur la pile, un pirate peut causer l'exécution de quelques commandes arbitraires par le système d'exploitation qui vont l'aider à obtenir ce qu'il cherche. Une autre classe des attaques U2R exploite les programmes qui donnent des idées sur l'environnement dans lequel ils s'exécutent, un bon exemple d'une telle attaque est l'attaque *«load module»*. Une autre classe d'attaques U2R exploite des programmes qui ont une mauvaise gestion des fichiers temporaires.

c. Attaques Distant vers poste Local (Remote To Local R2L)

Une attaque R2L se produit quand un pirate qui peut envoyer des paquets vers une machine à travers un réseau mais qui n'a pas de compte sur cette machine exploite une vulnérabilité afin d'obtenir un accès local comme utilisateur de cette machine. Il y a plusieurs manières avec lesquelles un attaquant peut aboutir à son objectif [5]. Certaines attaques exploitent le débordement des tampons causé par les logiciels du serveur du réseau *« imap, named, sendmail »*. Les attaques *« dictionary, ftp-write, guest et xsnoop »*, essaient d'exploiter la faiblesse ou la mauvaise configuration des politiques de sécurité du système. L'attaque *« xlock »* utilise l'ingénierie sociale, pour réussir, l'attaquant doit parodier les opérateurs humains à fournir leurs mots de passe aux économiseurs d'écran qui sont en réalité des

chevaux de Troie.

d. Attaques par Sondes (Probe)

Ces dernières années, on distribue un nombre de plus en plus important de programmes qui peuvent automatiquement balayer un réseau d'ordinateurs pour recueillir des informations utiles ou pour trouver des vulnérabilités connues [5]. Ces sondes réseau sont tout à fait utiles pour un pirate qui prépare une future attaque. Un attaquant avec un mappage des machines et des services disponibles sur un réseau peut employer ces informations pour rechercher tous les points faibles de ce dernier. Certains de ces outils de balayage *« satan, saint, mscan »* permettent, même à un pirate débutant, d'examiner très rapidement des centaines ou des milliers de machines sur un réseau.

e. Attaques de Données (Data)

Les attaques de données impliquent quelqu'un (utilisateur ou administrateur) qui effectue certaines actions, dont il a la possibilité de les exécuter sur un poste donné, mais selon la politique de sécurité du site où il opère, il n'a pas les permissions suffisantes pour le faire. Souvent, ces attaques impliquent le transfert des fichiers de données secrètes de ou vers des sources auxquels elles n'appartiennent pas.

1.3 Exemple d'un scénario d'attaque sur le réseau

Les attaquants peuvent appliquer un plan d'attaque bien précis pour réussir leurs exploits [60]. Leurs objectifs sont distincts et multiples. On distingue l'attaquant hacker, qui dans un but d'approfondissement de connaissances, essaie de découvrir les failles de sécurité dans un système informatique. Cette personne partage librement ses découvertes et évite la destruction intentionnelle des données. Le deuxième type d'attaquant, appelé cracker, cherche à violer l'intégrité du système. Généralement, il est facilement identifiable à cause de ses actions nuisibles. Néanmoins il faut distinguer un expert qui cherche les exploits et conçoit lui même les programmes, d'un gamin scripteur (script kiddy) qui utilise la technologie existante pour ses fins malveillantes.

Les différents types d'attaquants cherchent à découvrir les propriétés du réseau cible avant de lancer les attaques. On parle généralement de la reconnaissance qui peut être passive ou active. Ayant récolté les informations nécessaires, ils lancent leurs vraies attaques pour

exploiter le système. Ensuite ils créent des portes dérobées pour garantir des futurs accès faciles au système compromis. Enfin ils effacent leurs traces des journaux de sécurité. Nous détaillons dans la suite ces différentes étapes en les illustrant par des vrais scénarios d'attaques.

a. Reconnaissance passive

Il s'agit d'une phase d'attaques où l'intrus n'effectue pas une action pour collecter les informations. Il se restreint à observer passivement les évènements afin d'en tirer les conclusions. Une des attaques les plus répandues est l'écoute du trafic (sniffing). Le principe consiste à installer une sonde sur le réseau pour capter le trafic et le sauvegarder dans des fichiers journaux. L'analyse de ces fichiers permet de connaître les machines installées sur le réseau et de déterminer les ports ouverts et les systèmes d'exploitation utilisés. L'attaque est considérée lente si l'attaquant cherche une information précise sur une machine particulière du réseau. En revanche, elle est si discrète qu'il est difficile de la détecter.

En analysant les fichiers journaux, les attaquants cherchent les valeurs par défaut des champs des protocoles. Ces valeurs dépendent des systèmes d'exploitation. Les intrus profitent alors de ces différences pour distinguer les systèmes d'exploitation et s'informer des services réseaux offerts par l'entreprise. Parmi les champs surveillés, on distingue :

- la taille initiale d'une fenêtre TCP (Window),
- la durée de vie d'un paquet (TTL),
- la taille maximale d'un segment TCP (MSS),
- le bit de non fragmentation (DF),
- le facteur multiplicateur de la fenêtre de réception (WS),
- l'acquittement sélectif (SAck, SAckOK),
- l'option "aucune opération" (NOP),
- l'option d'estampille de temps (TS).

Le Tableau 1.1 présente des divergences dans l'implantation de deux options WS et TS de TCP [178]. Ainsi, l'analyse de quelques paquets IP permet de caractériser les systèmes d'exploitation des machines installées sur le réseau.

b. Reconnaissance active

L'objectif de la reconnaissance active est similaire à celui de la reconnaissance passive. Il s'agit d'acquérir des informations utiles sur le réseau cible. La reconnaissance active parait néanmoins plus fructueuse puisque l'attaquant ne se restreint pas à inspecter les données échangées entre les différents hôtes.

Tableau 1.1 Options TS et WS en fonction des systèmes d'exploitation

Système d'exploitation	**WS sans TS**	**WS avec TS**
Linux 2.4.0	5840	5792
Windows NT 2003	64240	65160
MAC OS 10.1	32768	33000
Open BSD 3.3	64240	65160
Free BSD 9.2	16384	17520
Free BSD 4.6	57344	57344

Cependant, il initie lui-même des connections réseaux pour tester le comportement des machines. Il cherche des informations précises concernant les hôtes accessibles, l'emplacement des routeurs et des pare feux, les systèmes d'exploitation, les ports ouverts, les services fournis et les versions des applications exécutées. Parmi les techniques les plus utilisées pour acquérir ces informations nous évoquons les utilitaires PING, NsLookup, DIG et Traceroute.

1.3.1 Exploitation du système

Les deux types de reconnaissance passive et active permettent à un attaquant de localiser les applications vulnérables pour exploiter ensuite leurs faiblesses. L'intrus cherche à gagner un accès au réseau cible, élever son privilège ou lancer des attaques par "déni de services". Pour accéder au système, il peut se baser sur les mauvaises installations des services. En effet, certaines configurations par défaut activent toutes les options disponibles pour prouver la richesse de l'application. Seulement ces options présentent le plus souvent des failles de sécurité facilement exploitables pour mener une attaque. Par exemple, le premier daemon ***inetd*** activait tous ses services dont une bonne partie était vulnérable.

L'exploitation des interactions entre deux programmes constitue également un moyen pour accéder aux systèmes informatiques. Diverses failles apparaissent si une application à fort privilège ne protège pas ses méthodes, utilise des communications inter processus défaillantes ou crée des fichiers temporaires accessibles par d'autres programmes. La Figure 1.1 montre un scénario où un utilisateur à fort privilège accède au fichier "/tmp/race". La période qui sépare la vérification du privilège et l'ouverture du fichier est critique puisque un utilisateur malveillant peut remplacer le fichier "/tmp/race" par un lien symbolique vers un autre fichier plus important comme "/etc/passwd". Ainsi, en manipulant le lien symbolique, la victime peut corrompre involontairement les fichiers systèmes ou ajouter des lignes qui correspondent à des nouveaux comptes utilisateurs.

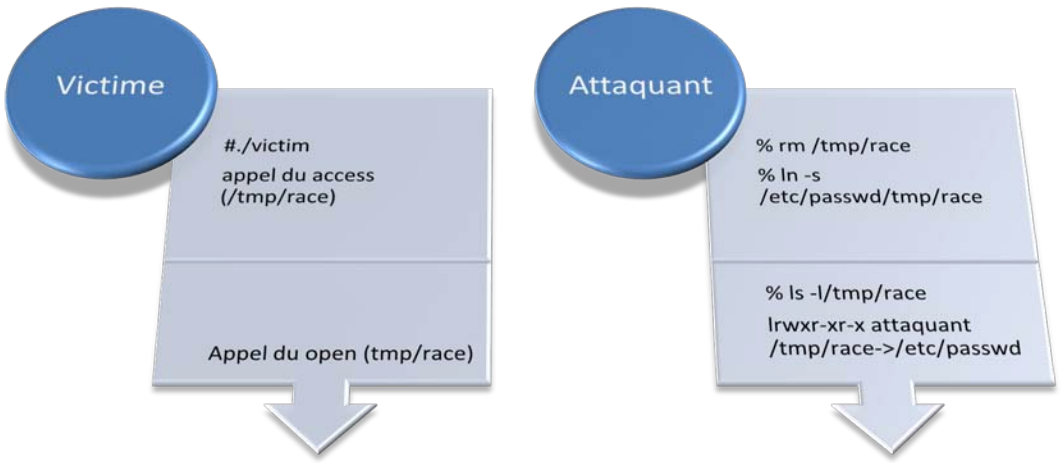


Figure 1.1 : Accès concurrents au contenu du fichier.

Par ailleurs un attaquant peut introduire des données imprévues pour tromper les applications mal conçues. Il réussit ainsi des dépassements de tampon, altère des requêtes SQL et détourne des mécanismes d'authentification. Nous présentons dans la Figure 1.2 le mécanisme de débordement de tampon. Il s'agit de copier dans la pile une grande quantité de données dans un espace mémoire assez petit, prévu pour contenir les variables locales des fonctions. Les données en excès contiennent du code malveillant et pénètrent dans des espaces mémoires voisins. De plus, le pointeur retour de la fonction se voit écrasé par une autre

adresse qui pointe directement vers le code malveillant. Par conséquent, l'attaquant exploite le privilège de l'application pour créer de nouveaux comptes utilisateurs, élever son privilège ou copier des données sensibles du système.

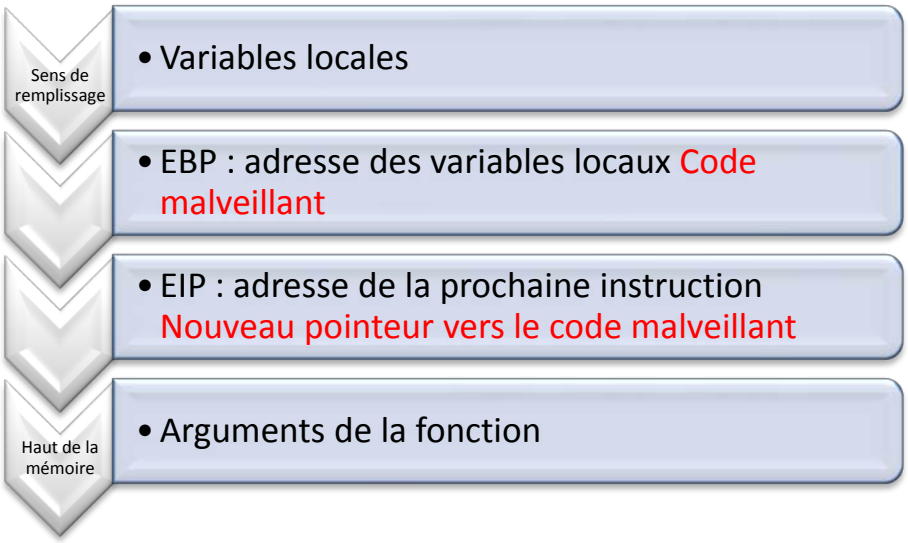


Figure 1.2 : Débordement de tampon.

Contourner les mécanismes d'authentification est encore possible via les entrées imprévues. Considérons par exemple la requête SQL (1.1). Cette requête permet de vérifier le mot de passe de l'utilisateur $*varNom* en consultant la table utilisateur. Seulement un intrus peut s'identifier avec un nom d'un utilisateur légitime puis entrer un mot de passe de la forme *cmoi OR TRUE*. La requête SQL (1.1) se transforme en (1.2) et sera toujours vérifiée si l'utilisateur $*varNom* existe dans la table utilisateur.

*select * from utilisateur where nom = $varNom and mot2passe = $varPasse* *(1.1)*

*select * from utilisateur where nom = $varNom and True* *(1.2)*

En outre, un intrus utilise les attaques par déni de services (DOS) pour exploiter les systèmes informatiques. Il empêche l'exécution normale des services par la saturation de la bande passante ou l'épuisement des ressources systèmes. La Figure 1.3 montre le détournement de session réalisé par ***Mitnick*** [6]. Ce dernier a pu, via une inondation SYN,

isoler la machine A qui maintient des relations d'approbation avec la machine B. Ensuite il accède à la machine B en détournant la session de A.

Les différents ‘’exploits ‘’ présentés permettent d'accéder au système cible ou d'élever le privilège d'un utilisateur. Il existe cependant une solution plus rapide qui assure simultanément les deux attaques. Il s'agit des "œufs" qui comportent un code spécial contenant à son tour un autre code actif d'attaque. Le premier objectif est d'exploiter un programme doté d'un faible privilège puis d'attaquer et exploiter par l'intermédiaire du code actif, un morceau du code aux droits d'accès plus élevés.

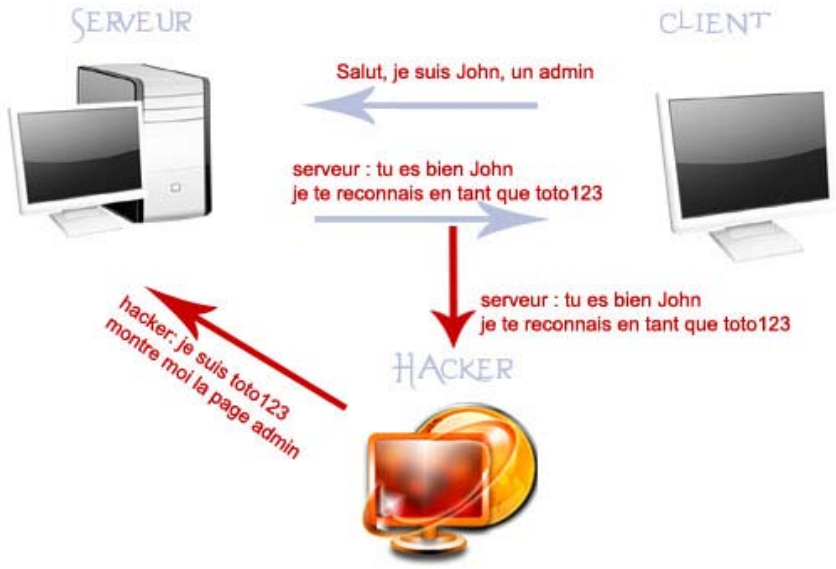


Figure 1.3 : Attaque de détournement de session.

1.3.2 Préservation de l'accès

Les attaquants installent des portes dérobées pour retourner facilement aux systèmes compromis. Par exemple, ils créent de nouveaux comptes et les utilisent lors des prochains accès. Seulement cette procédure est facilement détectable si un administrateur vérifie constamment l'intégrité des fichiers des mots de passes.

Un attaquant prudent utilise des chevaux de Troie qu'il télécharge aussitôt qu'il gagne son premier accès. Pour ce faire, il dispose de plusieurs sources d'informations comme l'IRC, les pages Web, le partage point à point et les dépôts "FTP". La trace présentée dans [7] montre

un intrus qui installe le rootkit "lrk4". Il s'agit de plusieurs programmes modifiés dont l'un remplace le binaire "/bin/login". Le nouvel exécutable accepte par défaut le compte root "rewt" avec le mot de passe "saton". Ainsi les futures connexions " TELNET " utilisant ce compte réussissent.

- ***Effacement des traces***

Un attaquant qui compromet une machine et crée une porte dérobée, cherche aussitôt à effacer ses traces. Il nettoie ces entrées des fichiers journaux de sécurité. Une telle approche déclenche néanmoins des alarmes en vérifiant l'intégrité des fichiers. L'attaquant essaie de restituer les mêmes propriétés des fichiers (date de création, de modification, dernière utilisation, etc.) pour garder la même signature. Ceci force les administrateurs à enregistrer les évènements suspects sur des machines distantes pour mieux protéger les fichiers de sécurité. Ils multiplient également les utilitaires d'archivage pour résister aux intrus qui tentent de désactiver ces fonctionnalités ou de les modifier par d'autres programmes dès le premier accès à la machine.

1.4 Protection du système informatique

Nous avons présenté dans la section 1.2 un scénario possible d'attaque pour compromettre une machine sur un réseau distant. Nous constatons que les attaquants disposent de plusieurs moyens pour réussir chaque phase d'attaque. La disponibilité des outils d'attaques et la richesse des sources d'informations accentuent le risque des intrusions. Par conséquent les administrateurs sécurisent de plus en plus leurs systèmes informatiques. Ils s'appuient sur diverses solutions comme les pare feux, la cryptographie, les scanners de vulnérabilités et les systèmes de détection d'intrusions. Nous détaillons dans la suite chacune de ces méthodes et nous soulignons leurs limites.

1.4.1. Pare feux

Un pare feu (Firewall) est un système physique ou logique qui inspecte les flux entrant et sortant du réseau [4]. Il se base sur un ensemble de règles appelées ACL afin d'autoriser ou interdire le passage des paquets. Il existe principalement trois types de pare feux :

- pare feu avec filtrage des paquets : ce pare feu filtre les paquets en utilisant des règles statiques qui testent les champs des protocoles jusqu'au niveau transport. Ces

fonctionnalités sont généralement incorporées dans un routeur appelé "routeur à écran",

- pare feu à filtrage des paquets avec mémoire d'états : ce modèle conserve les informations des services utilisés et des connexions ouvertes dans une table d'états. Il détecte alors les situations anormales suite à des violations des standards protocolaires,
- pare feu proxy : ce pare feu joue le rôle d'une passerelle applicative. En analysant les données jusqu'au niveau applicatif, il est capable de valider les requêtes et les réponses lors de l'exécution des services réseaux.

Malgré leurs grands intérêts, les pare feux présentent quelques lacunes. En effet, un attaquant peut exploiter les ports laissés ouverts pour pénétrer au réseau local. Ce type d'accès est possible même à travers des pare feux proxy. Il suffit d'utiliser un protocole autorisé tel que HTTP (Port 80) pour transporter d'autres types de données refusées. Ainsi l'opération supplémentaire d'encapsulation/décapsulation des données permet à l'attaquant de contourner le pare feu. Les scripts constituent aussi des sources d'intrusion que les pare feux échouent à détecter.

Par exemple la vulnérabilité du RDS (Remote Data Service) sur les serveurs web IIS (Internet Information Server) de Microsoft permet aux intrus d'exécuter des commandes à distance sur des stations Serveur NT [8]. Le script "msadc.pl" de Rain Forest Puppy (RFP) exploite cette vulnérabilité. Il emploie des méthodes valides du protocole HTTP telles que GET et POST pour pouvoir passer inaperçu à travers un pare feu proxy.

1.4.2. Scanners de vulnérabilités

Les scanners de vulnérabilités automatisent la découverte des failles de sécurité. Ils sont utilisés par les attaquants pour localiser les faiblesses du réseau cible. De plus les administrateurs peuvent en tirer profit pour corriger les vulnérabilités de leurs systèmes informatiques. Nous citons à titre d'exemple "Nessus", "Whisker" et "Saint" [9].

Cependant les scanners présentent quelques limites qui peuvent être résumées en 3 points : l'exhaustivité, la mise à jour et l'exactitude. En effet, malgré le grand nombre de vulnérabilités détectées, les scanners d'aujourd'hui sont inaptes à déterminer toutes les faiblesses possibles.

De plus, la mise à jour de ces produits ne suit pas le rythme de la découverte des nouvelles vulnérabilités. Enfin, la modification des bannières des services scannés permet de dissuader facilement le scanner ce qui entraîne parfois un responsable de sécurité à chasser des vulnérabilités fantômes.

1.4.3 Outils d'archivage

La plupart des systèmes d'exploitation fournissent des utilitaires d'archivage. Par exemple le daemon *syslogd* d'Unix enregistre dans des fichiers journaux de sécurité les opérations intéressantes exécutées sur le système. Parmi les fichiers log créés, trois sont susceptibles d'être manipulés par les attaquants à savoir "wtmp, utmp et lastlog" [7].

- ✓ wtmp : contient un historique des connexions/déconnexions avec l'heure, le service et le terminal concerné,
- ✓ utmp : liste les utilisateurs connectés à un moment donné,
- ✓ lastlog : contient un historique des dernières connexions.

Les attaquants effacent souvent les entrées des journaux de sécurité et principalement des trois fichiers évoqués ci-dessus. De leur côté, les administrateurs vérifient l'intégrité de ces fichiers afin de détecter les éventuelles modifications. Ils dupliquent également les fichiers sur des machines distantes inconnues par les attaquants. Enfin, et pour résister aux arrêts intentionnels des daemons d'archivage, les responsables de sécurité varient les outils de sauvegarde.

Par conséquent les journaux de sécurité constituent une source intéressante pour analyser et détecter les attaques. Cependant, ces fichiers contiennent beaucoup d'informations normales et anormales. La taille énorme de ces fichiers pose souvent des problèmes de stockage et d'exploration du contenu. Les administrateurs fournissent aussi l'effort pour localiser les activités anormales, comprendre les objectifs des attaquants et déterminer les vulnérabilités exploitées du système.

1.4.4. Cryptographie

La cryptographie garantit la confidentialité, l'intégrité, la non-répudiation et l'authenticité des données. Elle est fréquemment utilisée dans diverses applications réseaux telles que la messagerie, les connexions à distance, les réseaux privés et les serveurs web. Les

administrateurs l'utilisent pour sécuriser leurs systèmes informatiques mais elle ne constitue pas une solution unique et suffisante. Effectivement, diverses implémentations des protocoles de sécurité se sont révélées vulnérables. De plus la sécurité peut être rompue via plusieurs types d'attaques. Par exemple l'homme du milieu constitue une menace lors des créations des clés. Par ailleurs les courts et les simples mots de passes utilisés comme des clés de sécurité par les algorithmes symétriques sont facilement cassables via des attaques par "dictionnaires" ou de "recherche exhaustive". Il y a aussi le risque de vol des clés privées suite à un mauvais stockage de ces informations ou la cryptanalyse appliquée sur des algorithmes à faible encodage tels que le XOR et la Base 64.

En outre la cryptographie empêche l'analyse aisée du contenu des paquets et rend donc difficile la détection des attaques si elles sont déjà insérées dans des protocoles réseaux. Elle constitue même un moyen pour camoufler les attaques et par suite contourner les pare feux et les systèmes de détection d'intrusions.

1.4.5 Pots de miel

Un pot de miel est une machine qui présente ou simule des failles de sécurité très répandues [10]. Disposant des moyens renforcés de surveillance, la machine peut servir d'appât pour apprendre la stratégie des attaquants et construire des signatures exactes d'attaques. Par ailleurs la simulation du comportement d'une machine doit être aussi réaliste pour ne pas éveiller les soupçons des attaquants.

Un pot de miel dispose de plusieurs outils de surveillance et d'archivage, nécessaires pour collecter les informations des activités suspectes. Ces outils doivent être maintenus en permanence puisqu'ils sont déployés dans un environnement fréquenté principalement par des attaquants. De plus, l'isolation du pot de miel du reste du réseau est indispensable pour qu'il ne se transforme pas en une base pour compromettre d'autres machines.

1.4.6 Systèmes de détection d'intrusions

Une intrusion est toute activité qui menace la politique de sécurité de l'entreprise et mène à sa violation [5]. L'origine de l'intrusion est multiple et peut être due à un espionnage industriel ou des attaques lancées pour des gamins scripteurs. Ainsi un système de détection d'intrusions (IDS) tente d'identifier les menaces dirigées contre le réseau de l'entreprise. Il

s'appuie sur plusieurs sources d'informations comme les fichiers d'audit, les journaux de sécurité et le trafic réseau.

1.5 Conclusion

Nous avons étudié dans les sous sections précédentes diverses solutions pour sécuriser le réseau informatique et mentionné leurs intérêts et leurs limites. Il s'est avéré que ces outils ne peuvent pas prévenir toutes les attaques et par suite assurer seuls une sécurité idéale du réseau. Etant donnée l'impossibilité de stopper toutes les attaques, les systèmes de détection d'intrusions constituent une bonne solution pour détecter celles qui passent inaperçues. Placés après les pare feux, les IDS constituent la dernière barrière de sécurité. Ils analysent le trafic qui passe à travers les pare feux et supervisent les activités des utilisateurs sur le réseau local. Par ailleurs, placés avant les pare feux, les IDS découvrent les attaques à l'entrée du réseau.

Les IDS s'appuient généralement sur deux sources d'information : les paquets transitant sur le réseau et les informations collectées sur les machines. On parle alors de deux types de systèmes de détection d'intrusions : les IDS basés réseau (NIDS) et les IDS basés hôte (HIDS). Ces deux catégories d'IDS emploient généralement deux principes de détection : " l'approche comportementale " et " l'approche basée sur la connaissance ".

La détection par la connaissance définit des signatures d'attaques qui décrivent les intrusions. Ces signatures ne sont autres que les empreintes laissées par les intrus au cours de leurs exploits. La deuxième approche de détection, c'est-à-dire *comportementale*, se réfère au comportement normal et habituel des différents acteurs du système à protéger (application, utilisateur, etc.), ensuite une déviation importante par rapport au profil normal représente une activité suspecte et révèle éventuellement une attaque. Nous détaillons les deux approches de détection ainsi que leurs limites dans le Chapitre 2.

CHAPITRE 2
Détection d'Intrusions : Approches et Limites

2.1. Introduction

Nous avons présenté au premier chapitre un scénario possible d'attaque sur le réseau de l'entreprise. Ce plan d'attaque souligne la nécessité et la complexité de sécuriser un système informatique. De nos jours, diverses méthodes de prévention existent. Cependant, malgré leur grand intérêt, elles présentent des lacunes dues à la nature des données qu'elles traitent d'une part et aux techniques évoluées pour mener et dissimuler les attaques d'autre part. Ainsi et dans l'impossibilité de stopper toutes les attaques, la détection d'intrusions tente de déceler celles qui passent inaperçues à travers les autres systèmes de sécurité. Par ailleurs déployés à l'entrée du réseau, les systèmes de détection d'intrusions détectent les différentes attaques lancées contre le réseau de l'entreprise. La compréhension des origines des intrusions et des intentions des pirates permet de développer les moyens efficaces de prévention des attaques.

Ce chapitre aura pour but l'exploitation des qualités requises des IDS. Nous présentons un état de l'art des approches proposées dans la littérature et des limites actuelles de détection. Nous introduisons ensuite notre procédé basé sur une méthode d'inférence probabiliste. Cette technique de détection comportementale sera développée dans les prochains chapitres de notre thèse.

2.2. Les systèmes de détections d'intrusions

Nous présentons dans la première partie de cette section une classification des systèmes de détection d'intrusions. Ensuite, nous développons les qualités requises de ces équipements.

2 .2.1 Classification des systèmes de détection d'intrusions

La détection d'intrusions a commencé en 1980 avec le travail d'Anderson [11]. Il a proposé la surveillance des échecs d'authentification pour détecter les tentatives de pénétration au réseau local. De plus, l'observation des fichiers et des programmes manipulés permet de

détecter des attaques internes. Enfin l'usurpation d'une identité est repérable en comparant l'activité courante au comportement habituel d'un utilisateur. Ensuite, Denning [12] a proposé en 1987 le premier modèle de la détection d'intrusions. Son approche statistique construit des profils qui relient des sujets à des objets. Un sujet est l'initiateur des actions. Il peut être un utilisateur ou un processus qui manipule des objets c'est-à-dire des programmes ou des fichiers. Suite à ce premier modèle, plusieurs approches de détection d'intrusions ont été proposées. Elles reposent sur diverses techniques de détection. Etant donnée la diversité de ces méthodes, des schémas de classification ont été proposés par Kruegel [13] et Debar [14, 15].

Debar [37] utilise cinq critères pour classer les systèmes de détection d'intrusions (Figure 2.1) :

- Méthode de détection : deux principes de détections existent. L'approche comportementale modélise le comportement normal des utilisateurs, du système informatique et de l'activité réseau. Ensuite, toute déviation par rapport à la normale constitue un évènement suspect.
- La deuxième approche, appelée détection par connaissances, recherche explicitement les signatures des attaques connues dans les fichiers de sécurité et le trafic réseau.
- Source d'information : les données analysées partagent les systèmes de détection d'intrusions en deux catégories : les systèmes de détection d'intrusions réseaux et les systèmes de détection d'intrusions hôtes. La première catégorie des IDS filtre le trafic réseau et se déploie généralement dans des endroits précis du réseau, par exemple dans la zone démilitarisée, un brin réseau contenant des serveurs internes ou juste avant ou/et après un pare feu. La deuxième catégorie des IDS analyse les données des journaux de sécurité établis par les systèmes d'exploitation et les applications qui tournent sur les machines. Ces IDS sont déployés directement sur les hôtes du réseau.
- Réponses des IDS : les systèmes de détection d'intrusions émettent des réponses actives qui influent directement la source d'attaque, comme ils peuvent se restreindre à des réponses passives qui inscrivent l'évènement suspect. Une liste de réponses actives et passives est présentée dans le Tableau 2.1 [6]. Par ailleurs et afin de sélectionner la meilleure réponse, *Cuppens* introduit la notion d'anti-corrélation et l'applique sur le but de l'intrusion [16].

- Paradigme de détection : la détection d'intrusions s'effectue en analysant l'état courant du système ou en supervisant les transitions des états normaux aux états dangereux. Durant ces deux types d'inspection, l'IDS récupère les informations en interrogeant directement le système ou en écoutant passivement les évènements.
- Mode de supervision : l'analyse assurée par un système de détection d'intrusions peut être continue ou périodique dans le temps.

Les systèmes de détection d'intrusions utilisent diverses techniques de détection et définissent plusieurs modules qui effectuent des tâches distinctes comme la collecte des informations, l'analyse des données, la corrélation des évènements et la génération des alarmes.

Afin d'homogénéiser les communications entre les différents équipements, des travaux de standardisation ont été développés et d'autres sont en cours de développement. On note en particulier le CIDF [17] qui développait des protocoles et des API permettant la réutilisation facile des composants des IDS.

Tableau 2.1 : Réponses aux attaques des systèmes de détection d'intrusions

Réponse passive	Réponse active
Emettre un rapport	*Bloquer le compte d'un utilisateur*
Générer une alarme	*Suspendre des processus malveillants*
Activer un archivage plus détaillé	*Terminer une session*
Activer un archivage à distance	*Bloquer une adresse IP*
Créer des fichiers de sauvegarde	*Arrêter la machine*
	Déconnecter la machine du réseau
	Mettre hors services les ports et les services attaqués
	Avertir l'utilisateur
	Tracer l'origine de la connexion
	Restreindre les activités d'un utilisateur

Par ailleurs l'IETF a créé un groupe de travail IDWG (Intrusion Detection Working Group) [18] pour définir le format des données et des échanges entre les systèmes de détection et de réponse aux intrusions. Le but du travail est de partager facilement l'information entre les systèmes de détection d'intrusions et de contrôler tout système interagissant avec eux [18].

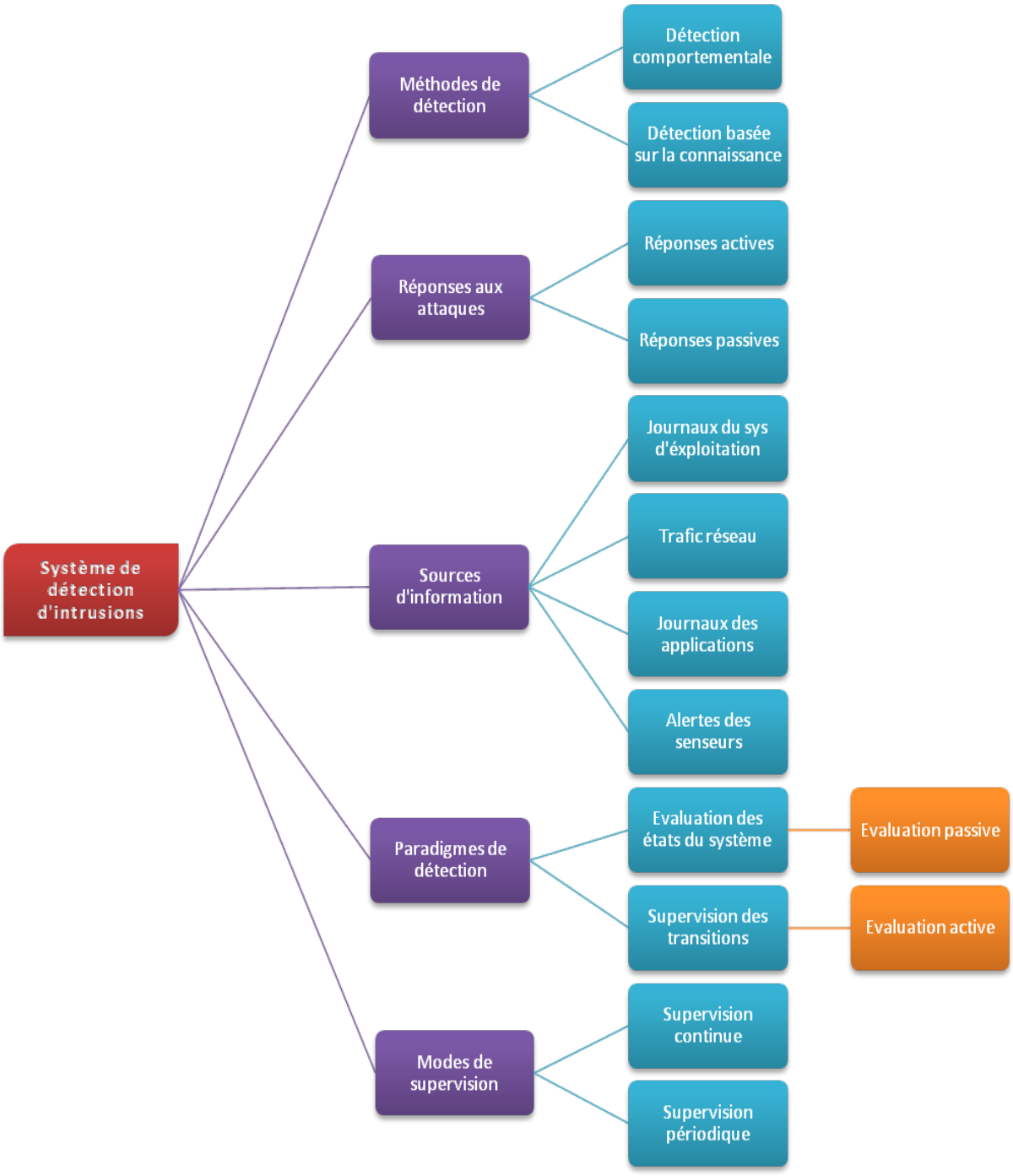


Figure 2.1 : Classification des systèmes de détection d'intrusions [7]

2.2.2 Qualités requises des systèmes de détection d'intrusions

Les systèmes de détection d'intrusions actuels tendent à garantir les cinq propriétés suivantes [7] :

- Exactitude de détection: elle se traduit par une détection parfaite des attaques avec un risque minimal de faux positifs.
- Performance: une détection rapide des intrusions avec une analyse approfondie des évènements est indispensable pour mener une détection efficace en temps réel.
- Complétude: une détection exhaustive des attaques connues et inconnues.
- Tolérance aux fautes : les systèmes de détection d'intrusions doivent résister aux attaques ainsi qu'à leurs conséquences.
- Rapidité: une analyse rapide des données permet d'entreprendre instantanément les contre mesures nécessaires pour stopper l'attaque et protéger les ressources du réseau et du système de détection d'intrusions.

Afin d'assurer ces qualités, les systèmes de détection d'intrusions implantent différentes approches de détection. Nous choisissons le critère "méthode de détection" pour présenter dans la Section 2.3 un état de l'art des travaux de recherche du domaine.

2.3. Méthodes de détection d'intrusions

Nous présentons dans cette section quelques techniques de détection d'intrusions. Nous nous intéressons d'abord à la détection comportementale puis nous traitons le cas de la détection basée sur la connaissance.

2.3.1 Détection d'intrusions comportementale

La détection d'intrusions comportementale ou par anomalie repose sur l'hypothèse qu'une attaque provoque une utilisation anormale des ressources ou manifeste un comportement étrange de la part de l'utilisateur. Par suite, les différentes approches qui ont été proposées apprennent le comportement normal pour pouvoir détecter toute déviation importante. On peut classer ces travaux en cinq catégories à savoir l'approche statistique, l'apprentissage automatique, l'approche immunologique, la spécification du comportement et la fouille de données.

2.3.1.1 Approche statistique

L'analyse statistique du comportement normal du système est l'une des premières approches adoptées en détection d'intrusions. Denning [12] présente un modèle dans lequel un profil relie via une variable aléatoire un sujet (utilisateur, processus) à un objet (ressources). Si après la création du profil, la valeur de la variable aléatoire dépasse le seuil toléré alors le comportement est considéré anormal. Divers systèmes de détection d'intrusions utilisent ce concept. NIDES [19] calcule des valeurs d'anomalie de plusieurs activités (temps CPU, bande passante, nombre et nature des services sollicités, etc.). Il effectue ensuite la pondération des carrés de ces valeurs afin de calculer un score d'anomalie global S (Eq. 2.1). Le score S est toujours positif et s'il dépasse le seuil toléré M, alors il s'agit d'un évènement suspect.

$$S = a_1 S_1^2 + a_2 S_2^2 + \ldots\ldots + a_n S_n^2 \qquad (2.1)$$

Pearl [20] utilise également l'approche statistique. Il effectue un ensemble de mesures $\boldsymbol{X = (x_1, x_2, \ldots, x_n)}$ puis représente sous forme d'un vecteur de *Bernoulli* ***''v''*** toute mesure $\boldsymbol{x_i}$ qui dépasse un seuil prédéfini. Ensuite le vecteur ***''v''*** est multiplié par un vecteur poids ***''p''*** spécifique à chaque intrusion. Le produit obtenu représente le score d'anomalie et il est considéré élevé si la probabilité d'obtenir des scores plus petits est grande.

Si l'approche statistique bénéficie d'un grand nombre d'outils largement étudiés, elle se heurte à la difficulté de définir adéquatement le seuil optimal d'anomalie. De plus elle doit spécifier avec précision et uniquement les mesures qui sont en relation avec l'attaque recherchée. Par ailleurs l'interdépendance des mesures doit être considérée pour mieux estimer le score global d'anomalie. Enfin l'approche est incapable d'exprimer toute seule la séquence d'évènements.

2.3.1.2 Apprentissage automatique

Une autre technique de la détection d'intrusions comportementale consiste à prévoir la séquence normale des évènements audités. Teng [21] propose une méthode inductive de génération de règles décrivant la séquence normale des activités des utilisateurs. L'équation 2.2 représente un exemple de règles :

$$E_1, E_2, * \longrightarrow (E_4 = 95\ \%; E_5 = 5\%) \qquad (2.2)$$

La règle 2.2 utilise le caractère * pour remplacer un évènement quelconque et indique que la séquence (*E*1, *E*2, *) est suivie par l'évènement *E*4 dans 95% des cas et par *E*5 dans 5% des situations possibles. Une déviation par rapport au comportement normal se caractérise par une apparition d'évènements autres que *E*4 et *E*5 ou une disproportion des évènements *E*4 et *E*5. La technique est capable de modéliser une variété de comportements et résiste bien à l'apprentissage progressif et malveillant des intrus. Néanmoins, elle est trop liée à la séquence d'évènements ce qui pose des problèmes dans le cas où un utilisateur effectue simultanément deux ou plusieurs tâches.

2.3.1.3 Approche immunologique

Forrest [22] était la première à utiliser l'approche immunologique pour modéliser les processus sur une machine. Sa méthode consiste à décrire le comportement normal ou le "soi" via une séquence finie d'appels systèmes. Les séquences appelées N-gram servent de base pour comparer les appels systèmes des processus lors d'une phase de surveillance. Cette comparaison énumère les différences entre les paires dans une fenêtre de taille ***k*** (tide) [23] ou utilise des règles de ***r*** bits contiguës (stide). Wespi, Dacier et Debar [1] considèrent un cas plus général en analysant les évènements d'audit. Ils génèrent des séquences d'évènements de tailles variables pour modéliser l'état normal du système. Ensuite un motif est sélectionné s'il existe *d* motifs qui le suivent directement sinon le score d'anomalie est incrémenté de 1 et une alarme est déclenchée lorsque le score dépasse le seuil toléré.

Feldmann [24] optimise la représentation des N-gram sous forme de graphes acycliques orientés (DAG) ce qui permet de réduire la base de profils définie par Forrest. De plus, elle utilise le mécanisme de fenêtre glissante pour comparer les motifs. Ruschitzka [25] étudie les caractéristiques des traces des appels systèmes et remarque que les différences entre les motifs apparaissent dans des régions de tailles fixes. En divisant la trace en 3 parties : début, corps et fin, il réussit à générer de nouvelles séquences de motifs représentées en des machines à états finis. La méthode permet de réduire le nombre de séquences.

Han et Cho comparent dans [26] quatre approches immunologiques : la séquence simple d'évènements (stide), la séquence d'évènements menue des fréquences d'apparition (tstide), la génération automatique des règles inductives via RIPPER et le modèle caché de Markov (HMM). Ils concluent qu'en moyenne la modélisation HMM présente des meilleures

performances. Mais il ne s'agit pas d'une supériorité absolue puisque les résultats des expériences dépendent des programmes testés.

La lenteur d'apprentissage constitue le principal inconvénient de la modélisation HMM. Afin de résoudre ce problème, Cho [26] s'intéresse uniquement aux évènements qui sont en relation avec le changement de privilège. Il réduit ainsi le nombre d'appels systèmes audités de l'ordre de 75%. Par ailleurs et afin de diminuer les faux positifs très répandus dans le cadre d'une détection d'intrusions comportementale, l'auteur combine plusieurs HMM et utilise diverses mesures dont les informations utiles sont raffinées via une méthode d'auto-organisation de données (SOM).

2.3.1.4 Spécification des programmes

La spécification des exécutions normales des processus constitue une autre technique de la détection d'intrusions comportementale. Ko et Redmond [27] s'intéressent aux programmes avec privilège "super utilisateur" et développent un langage de spécification qui se base sur la logique des prédicats et les expressions régulières. Cependant la réalisation de ces spécifications est une tâche assez difficile. Les auteurs proposent une méthode utilisant une logique inductive pour synthétiser directement les spécifications à partir des traces valides.

Dans ce contexte, Zimmermann [28] construit un diagramme de transition d'états pour représenter les changements de privilège. Il définit ainsi un ensemble de règles dont la violation révèle la présence d'une attaque.

Enfin Sekar [29] utilisent un langage spécifique BMSL (Behavioral Monitoring Specification Language) pour modéliser les propriétés importantes des évènements à analyser. Ils expriment via ce langage des règles de la forme *motif* → *action*. Les motifs sont des expressions régulières d'évènements qui seront traduites sous forme d'une machine à états finis quasi déterministe. Les actions sont des opérations d'affectation à des variables d'états ou de simples appels à des routines extérieures pour stopper les attaques en cours.

2.3.1.5 Fouilles de données et théorie d'information

Lee et Xiang [30] utilisent la théorie d'information pour comprendre la nature des données auditées et par suite construire des modèles de détection d'intrusions

comportementale. Les techniques de fouilles de données « Data Mining » permettent également de construire des modèles de détection adaptatifs. Les algorithmes utilisées par Mé [2, 8] divisent les données en deux catégories : des données normales et des données anormales. Cette classification permet de construire des règles qui expriment des relations entre les enregistrements des fichiers de sécurité.

Par exemple, pour un utilisateur particulier, l'éditeur Xemacs est le plus souvent associé à des fichiers ".c". Lee [38] souligne que l'extraction des évènements fréquents permet de mieux analyser les traces d'évènements. Ainsi une méta-classification des analyses de plusieurs IDS garantie une meilleure détection avec moins de faux positifs. De plus l'analyse de données porte sur des traces normales pour assurer une détection comportementale ou bien sur des traces d'intrusions. Elle contribue donc à construire des règles de détection d'attaques utilisables lors d'une détection d'intrusions par connaissances.

ADAM est autre système de détection d'intrusions qui utilise la fouille de données. Il est basé sur les travaux de *Barbara* [31] et effectue deux étapes d'apprentissage. La première étape utilise des données hors ligne pour construire des règles d'association modélisant les profils normaux. La deuxième étape considère des données en ligne et emploie les règles d'association déjà construites pour créer un classificateur d'évènements suspects. L'objectif de cette phase est de rendre le système de détection d'intrusions plus apte à distinguer les vraies attaques des faux positifs.

2.3.2 Détection d'intrusions basée sur la connaissance

Contrairement à la détection d'intrusions par anomalie qui apprend le comportement normal, la détection d'intrusions basée sur la connaissance cherche directement les activités intrusives, diverses approches ont été proposées. Elles peuvent être classées en cinq catégories : les systèmes experts, les automates, les algorithmes génétiques, la fouille de données et le filtrage de motifs.

2.3.2.1 Systèmes experts

Les premiers prototypes des systèmes de détection d'intrusions utilisent les systèmes experts pour détecter les scénarios d'attaques. La base de règles traduit les connaissances des experts ce qui permet de transformer les faits extraits des sources d'audit et d'en déduire la

présence des attaques. Plusieurs langages de définition de règles sont proposés. Par exemple, P-BEST (Production Based Expert System Toolset) a été initialement développé pour le système MIDAS [17] puis employé par IDES [29], NIDES [19] et EMERALD [12]. Par ailleurs, *Charlier* définit le langage RUSSEL (Rule based Sequence Evaluation Language) pour décrire les signatures d'attaques dans le système ASAX [32]. Néanmoins l'utilisation de ces deux langages nécessite une certaine expertise. De plus, définir de nouvelles signatures d'attaques nécessite du temps surtout lorsque la description d'une attaque requiert l'ajout de plusieurs règles.

Afin d'améliorer la détection des attaques en tenant compte des activités courantes sur le système, Debbar et Mé [15] ajoutent au système expert le concept du raisonnement sur les modèles. En effet, à chaque modèle d'attaque est associé des indicateurs d'activités. Ainsi, la supervision des activités permet d'activer seulement quelques modèles et par suite de tenir compte uniquement des scénarios d'attaques susceptibles d'être vérifiés.

Les systèmes experts présentent l'avantage de séparer la description des attaques des méthodes de détection. Néanmoins, la modélisation des connaissances sous forme de règles de la forme "si _ alors" s'avère parfois insuffisante. De plus la construction des règles exige un niveau minimum d'expertise. La tâche devient plus difficile si on ajoute les preuves d'occurrence (raisonnement) aux modèles d'attaques.

2.3.2.2 Automate et logique temporelle

Plusieurs approches de détection d'intrusions utilisent les automates pour représenter les scénarios d'attaques. Ces travaux se basent sur les machines à états finis, les réseaux de pétri et la logique temporelle de chemins.

a. Machines à états finis

Quinlan [33] représente les signatures d'attaques sous forme de machines à états finis. Les états de la machine correspondent aux états du système en voie de compromission et ils contiennent des assertions qui doivent être vérifiées pour pouvoir transiter d'un état à un autre. Les arcs sont étiquetés par des évènements qui forment le scénario d'attaque. Le système de détection d'intrusions USTAT [22] implante cette approche pour détecter les attaques sur le système Unix. Par ailleurs Eckmann et Vigna [34] adaptent le système pour détecter des

attaques réseaux. Leur modélisation transforme l'infrastructure réseau en un hypergraphe dont les nœuds sont des interfaces réseaux et les arcs sont des hôtes et des liens réseau. Nous constatons que la description des scénarios d'attaques sous forme de machines à états finis est un moyen plus facile que la définition de règles dans un système expert. Cependant, la méthode ne permet d'exprimer que de simples relations qui portent sur des séquences d'évènements.

b. Réseau de Pétri

Kumar [35] utilise les réseaux de PETRI pour représenter les scénarios d'attaques. Les transitions sont étiquetées par des appels systèmes alors que les jetons évoluent chaque fois qu'un évènement permet de tirer une transition. Un jeton possède une couleur qui est une évaluation des variables. De plus, les transitions sont gardées pour vérifier certaines conditions. Les réseaux de pétri offrent un bon pouvoir expressif puisqu'ils permettent de représenter l'existence et la séquence d'évènements. De plus ils expriment des expressions régulières munis d'un opérateur de parallélisme. Néanmoins leur mise en ouvre pose quelques problèmes. En effet, à chaque commande on crée de nouveaux jetons, ce qui augmente le nombre de jetons présents et alourdit le processus de vérification.

c. Logique temporelle

Roger [36] transforme l'analyse des journaux d'audit en un problème de vérification des propriétés dans une logique temporelle. En effet, les scénarios d'attaques sont des formules de la logique temporelle linéaire alors que le fichier d'audit est la trace sur laquelle les algorithmes de vérification s'exécutent. Le langage de définition des attaques devient assez expressif grâce aux opérateurs de la logique temporelle (Next, Until). De plus la vérification traite des problèmes d'atteignabilité, de sûreté et d'équité.

d. Algorithmes génétiques

Le système de détection d'intrusions Gassata [37] utilise les algorithmes génétiques pour détecter les intrusions à posteriori. Cette méthode de détection vise particulièrement les attaques dont l'ordre temporel des évènements importe peu dans le déroulement du scénario d'attaque. Ludovic Mé [2] définit une matrice d'attaques/évènements notée ***A*** qui caractérise les évènements associés à chaque attaque. De plus il construit un vecteur ***R*** du risque encouru par chaque attaque et de la liste d'évènements ***O*** observés durant une période ***T***. Le problème

de détection se transforme en un problème d'optimisation qui est résolu via les algorithmes génétiques (Eq 2.3).

$$\begin{cases} A * H \leq O \\ R * H = Maximum \end{cases} \quad (2.3)$$

La méthode de détection s'adapte bien lorsque les administrateurs s'intéressent uniquement à des attaques considérées intrusives et sévères sur leurs réseaux. De plus le temps d'analyse est satisfaisant. Cependant, la détection est fortement liée à la période T de collecte des évènements.

De plus, elle ne considère pas les évènements communs entre plusieurs attaques. Enfin, la construction de la matrice attaques/évènements exige une certaine expertise de la part de l'utilisateur.

e. Fouille de données

Les techniques de la fouille de données (Data Mining) s'appliquent aussi bien pour une détection d'intrusions par anomalie que pour une détection d'intrusions basée sur la connaissance. Nous avons présenté cette méthode dans le cadre de la détection d'intrusion comportementale. Lee [38] utilise les données d'apprentissage du programme d'évaluation DARPA pour construire automatiquement des règles de détection d'attaques.

f. Filtrage de motifs

La description sémantique d'un scénario d'attaque peut se traduire en informations explicites caractérisant l'attaque. Pour une détection basée hôte, ces informations se présentent sous forme d'évènements à retrouver à partir des journaux de sécurité. Cependant, un système de détection d'intrusions basé réseau examine les champs des protocoles réseaux et filtre le contenu des paquets à la recherche des chaînes de caractères qui sont des motifs d'attaques.

2.3.3 Avantages et inconvénients des méthodes de détection

Nous avons présenté les deux principes de la détection d'intrusions : l'approche comportementale et l'approche basée sur la connaissance. D'une part l'approche comportementale détecte toute déviation importante dans le comportement habituel d'un utilisateur. Diverses techniques de détection existent à savoir l'approche statistique,

l'apprentissage automatique, l'approche immunologique, la spécification des programmes et la théorie d'information. D'autre part, la détection basée sur la connaissance cherche directement les activités malveillantes en s'appuyant sur les descriptions des attaques connues. Parmi les techniques employées nous citons les systèmes experts, les automates, les algorithmes génétiques, la fouille de données et le filtrage de motifs. Nous résumons dans le *Tableau 2.2* les avantages et les inconvénients de ces deux principes de détection.

Tableau. 2.2 : Comparaison des deux principes de détection d'intrusions [1]

	Détection par anomalie	***Détection basée sur la connaissance***
Avantages	Détection de nouvelles attaques	- Explication facile des scenarios d'attaques - Génération minimale des positifs
Inconvénients	- Apprentissage inexacte : si ce dernier se base sur des données supposées être normales mais contenant des attaques inconnues - Evolution du comportement normal ce qui exige un apprentissage adaptatif risque d'un apprentissage progressif initié par un intrus - Génération abondante des faux positifs - Sélection cruciale des ⇨ paramètres utiles lors de la phase d'apprentissage puisque des paramètres en trop représentent un bruit alors que ceux en moins altère la qualité de détection - Détermination difficile des seuils exacts d'anomalie - Fixation difficile des durées nécessaire à l'apprentissage	- Non détections des nouvelles attaques - Expertise requise pour construire efficacement des signatures d'attaques - Mises à jour continue de la base des règles - Augmentation rapide du nombre de règles qui généralement manquent d'abstraction - Délais important entre la découverte des attaques et la définition des signatures.

2.4. Installation des Systèmes de Détection d'Intrusions

Pour pouvoir configurer un IDS d'une manière efficace, il faut bien comprendre son fonctionnement interne. Toute erreur lors de son installation pourra le rendre inefficace ou inutilisable. Ainsi on va détailler dans ce qui va suivre une étape importante dans la mise en place des IDS à savoir leurs installations techniques. Lors du déploiement d'un IDS, il faut le

configurer correctement en fonction de l'OS, des applications et du matériel utilisés, et il faut avoir à l'esprit qu'il est impensable de vouloir analyser tout le trafic d'un réseau. Il faut donc donner la priorité aux systèmes à risque. De plus, afin de réduire la charge du senseur (une sonde, placée sur le réseau, dont le rôle est d'attraper le trafic circulant sur celui- i) de l'IDS, Il est souvent préférable de le placer après le pare-feu du côté interne. Ainsi, seul les flux acceptés par le pare-feu seront analysés.

Le choix matériel a également une grande importance. Puisqu'une sonde d'un IDS doit pouvoir analyser le trafic réseau quelque soit sa destination, elle devra recevoir tous les paquets. Mais le matériel réseau pose parfois des problèmes : L'utilisation des commutateurs simples (switchs) rend la conversation avec l'IDS impossible du fait que le switch commute les paquets directement au destinataire. Il est dès lors impossible d'installer une sonde qui analysera le trafic global. De plus, Malgré que l'utilisation des concentrateurs (hubs) rende la conversation avec l'IDS possible car les concentrateurs répètent les paquets en les émettant à toutes les machines connectées, ces équipements sont peu fiables et sont donc à éviter.

La solution sera donc l'utilisation des Switchs professionnels qui copie le trafic et l'envoie sur un port spécifié (où sera placé le N-IDS) : SPAN port (Switch Port Analyzer). Un autre problème doit être pris en considération : il est impossible pour un IDS de décrypter les flux lorsque ces derniers sont cryptés (ex : par SSL, qui est le cas pour les VPN). Il faut donc utiliser un proxy SSL pour traiter ce problème. Un autre point ne doit pas être oublié qui est la sécurisation du senseur et des logs d'alertes. En effet, un pirate pourrait très bien rendre l'IDS complètement inefficace, en visant ces deux composants. Pour sécuriser les sondes et les fichiers d'alertes, il est par exemple possible de mettre en place un réseau de management très contrôlé, avec son propre pare-feu et plusieurs mesures devront être prises pour assurer son fonctionnement tels la mise à jour régulière du système d'exploitation du senseur, la mise en place d'un système d'authentification robuste pour renforcer la sécurité, le changement régulier des mots de passe.

2.5. Outils de Détection d'Intrusions

Depuis le milieu des années 80 plusieurs IDS, mettant en œuvre de nombreux travaux s'inspirant du modèle d'Anderson, ont été commercialisés. IDES (*Intrusion Detection Expert System*) était le premier IDS hybride regroupant l'approche comportementale et celle par

scénarios, utilisant à la fois les méthodes statistiques et les systèmes experts. Le prototype a été ensuite amélioré pour aboutir à NIDES (Next-generation IDES) [19]. NIDES et d'autres outils développés durant les mêmes années ont montré la possibilité d'utiliser les différents comportements d'utilisateurs sur une machine UNIX, pour détecter un éventuel essai d'intrusion.

Cependant, si le comportement de l'utilisateur est trop riche, sa modélisation devient difficile et l'approche comportementale trouve sa limite. A partir du milieu des années 90, la plupart des IDS commerciaux ont commencé à implémenter principalement l'approche par scénario, vu sa simplicité à être déployée, configurée et maintenue, sa pertinente théorique, avec la possibilité d'ajout, et de retrait des signatures. Historiquement, les premiers IDS utilisés étaient les H-IDS, destinés aux serveurs de calcul basés sur une architecture UNIX avec des utilisateurs essentiellement locaux. Les informations d'audit collectées, provenaient surtout de la machine elle-même. Aujourd'hui, la plupart des IDS commerciaux sont des N-IDS. Le tableau 2.3 montre quelques outils commerciaux et libres, ainsi que les approches de détection qu'ils utilisent.

Tableau. 2.3 : Outils de détection d'intrusion [7]

	Système	**Approches**
H-IDS	*Asax (Audit UNIX)* *Gassata (Audit UNIX)* *Sutekh*	*Par Scénarios*
N-IDS	*BlackICE* *BRO* *Diams* *Dragon* *ETrust (ex Session Wall)* *NetProwler* *NetRanger* *NFR* *Shadow* *Snort*	*Par Scénarios*
	GrIDS	*Comportementale*
	Emerald	*Comportementale + Scénarios*

2.6. Notre approche

2.6.1 Introduction

La détection d'intrusion nouvelle correspond à l'identification d'intrusion inconnue qui n'a pas été fournie lors de l'apprentissage. C'est l`un des points fondamentaux d'une bonne classification ou d'un système d'identification, lorsque quelque fois les données de test contiennent des informations sur des objets qui ne sont pas connu au moment de l'entrainement. Les anomalies peuvent être vues comme une sorte de nouveauté. Normalement, on attend d'un classifieur de donner des résultats exacts pour les données de test similaires à celles qui ont été données pendant l'entrainement. Mais dans le monde réel, les besoins sont complètement différents, car des données anormales peuvent survenir, changeant alors la nature du problème.

Comparé au problème classique de classification en deux classes, un système de détection d'anomalie est entrainé avec seulement des configurations normales et tente de prédire ensuite sur des données anormales seulement à partir des modèles construits grâce aux données normales. Il existe plusieurs catégories de méthodes de détections nouvelles qui ont été utilisées sur différents ensembles de données. Il n'existe *à priori* pas de meilleure méthode conseillée pour tous les cas, car le succès de la technique employée dépend non seulement de la méthode mais aussi des propriétés statistiques des données en question. Pourtant il est vrai qu'en expérimentant les différentes méthodes d'apprentissage et de détection, pour un problème précis telle que l'analyse de *log* comme dans notre cas, il est possible de se rendre compte que certaines fonctionnent et d'autres non. Nous allons maintenant étudier la technique de base utilisée par notre approche pour pallier à ce type de problème.

2.6.2 Réseaux Bayésiens

Les algorithmes d'inférence probabiliste, fondés sur le principe d'inférence de Bayes [40], permettent de créer des détecteurs de scénarios pour remédier à la rigidité et à la difficulté d'exploitation à grande échelle de certaines méthodes comme le Pattern Matching. En fait les réseaux Bayésiens permettent de modéliser des situations dans lesquelles la causalité joue un rôle, mais où la connaissance de l'ensemble des relations entre les phénomènes est incomplète, de telle sorte qu'il est nécessaire de les décrire de manière probabiliste. Dans ce modèle, les attaques connues constituent des hypothèses qui peuvent expliquer les faits observés. On considère qu'une attaque donnée se traduit par des symptômes,

pouvant apparaître sous forme d'évènements dans l'audit, mais aussi par des données statistiques comme dans le cas de la détection d'anomalies (occupation mémoire, charge CPU, etc.). Étant donné un ensemble de symptômes, l'inférence bayésienne permet de calculer la probabilité de chaque scénario d'attaque connue. Lorsqu'un scénario annonce une probabilité élevée, une alerte sera levée. Pour cela la règle d'inférence de Bayes permet de calculer la probabilité de l'existence d'une attaque *A* étant donné un symptôme *S*, à partir de la connaissance de la probabilité que l'attaque *A* fasse apparaître le symptôme *S* et de la probabilité générale de l'occurrence de l'attaque *A*. Cependant, en pratique, il est très rare que *A* représente directement une attaque et *S* un symptôme. C'est pourquoi un détecteur d'intrusions à inférence bayésienne construit donc récursivement un arbre de décision. Les données de l'algorithme de construction de cet arbre, appelé aussi base des attaques, contient l'ensemble des hypothèses, dont certaines désignent les probabilités générales des attaques connues, ainsi que les probabilités de l'apparition des symptômes connaissant ces attaques.

Puisque un scénario d'attaque est constitué d'une présence combinée d'un ensemble de symptômes et non comme une séquence particulière d'évènements, l'élaboration d'un scénario non détectable par un attaquant reste une tâche très difficile surtout avec une base d'attaques non triviale. Ainsi seules les attaques réellement inconnues dans la base ne seront pas détectées. Ce qui représente l'un des avantages principaux de cette méthode. En plus, le principe d'inférence permet néanmoins de détecter le nombre de variantes d'une attaque donnée, et se montre performant notamment dans le cas où l'attaquant cherche à noyer son attaque avec du bruit, en générant un grand nombre d'opérations anodines. En pratique, l'efficacité de cette approche dépend principalement de la qualité de la base d'attaques, c'est-à-dire : l'exhaustivité des symptômes définis, la pertinence des hypothèses formulées et le réalisme des probabilités associées à ces hypothèses. Il faut noter aussi que la construction de la base nécessite à la fois un travail important d'expert (formulation des hypothèses) et une étude statistique de cas réels (calcul des probabilités).

2.6.3 Approche Probabiliste/GMM

Cette première étape est basée sur le modèle statistique de données et estime que les données de test possèdent la même distribution ce qui permet de générer les données de l'entrainement. Il faut tout d'abord estimer la fonction de densité des données de

l'entrainement. En supposant que les données de l'entrainement sont normales, la probabilité qu'une donnée de test fasse partie de cette classe, peut être calculée. Un seuil peut être fixé pour signaler une nouvelle intrusion si la probabilité calculée est inférieure à ce seuil.

Pour les modèle GMM (Gaussian Mixture Modelling), les paramètres de ce modèle sont choisis pour maximiser la probabilité que les données de l'entrainement respectent le modèle. Cette tache sera effectuée par la technique de ré-estimation tel que l'algorithme EM. Pourtant, GMM souffre d'un problème de dimensionnement dans le sens que si la dimension des données est trop élevée, un très grand nombre d'élément de test sera nécessaire ce qui implique des calculs trop complexes.

Un moyen plus simple consiste simplement à trouver la distance entre les données de test et la moyenne de la classe. Si les données de test sont trop éloignées de la somme de cette moyenne et du seuil de variance alors on peut en déduire qu'il s'agit d'un élément nouveau.

2.6.4 Méthodes d'estimation bayésienne

Dans cette deuxième étape, les capteurs sont considérés comme un ensemble d'entités capables de fournir une décision à tout instant. Chaque capteur est alors vu comme un estimateur bayésien. Ainsi, les distributions de probabilités associées à chaque capteur sont combinées dans une seule fonction de distribution de probabilités jointes *a posteriori*, en utilisant la règle de Bayes.

La vraisemblance de cette fonction est maximisée pour obtenir la fusion finale de l'ensemble des capteurs.

2.7. Conclusion

Scruter le réseau, ausculter tout les évènements, en constituer des suites descriptives de l'état du système, les enregistrer dans un journal d'audit, l'analyser d'une manière un peu plus diligente, détecter et signaler tout outrage à un modèle comportemental normal établi comme prémisse et référence unique entre ce qui doit être fait en détail et ce qui est fait réellement; c'est en bref, outiller le système informatique d'un détecteur automatique d'intrusion fondé sur l'approche comportemental. Ceci n'est qu'une reformulation de ce qui a déjà été dit, l'objectif est de mettre l'accent sur le fait que ce n'est toujours pas suffisant pour certifier que le réseau est muni d'un IDS efficace.

En effet, la caractéristique propre à tous les systèmes de détection est leur aptitude à générer de fausses alarmes (false positive). Il faut admettre que la déviation par rapport à un comportement normal ne signifie pas toujours un comportement intrusif.

A titre d'exemple, l'utilisation de Internet Explorer pour se connecter sur une page web peut générer jusqu'à une trentaine de sessions simultanées. Du point de vue d'un IDS, ceci peut être interprété comme un scan de port en provenance du serveur web vers la machine qui héberge le navigateur. En outre, un utilisateur peut intentionnellement faire habituer progressivement le système à un comportement intrusif.

CHAPITRE 3
Classification par Réseaux Bayésiens : Application à la Détection d'intrusion pour la construction d'un modèle ''intrusif''

3.1. Introduction

Deux types d'approche, facilitant le passage de l'information à la connaissance, connaissent tout naturellement un intérêt croissant. Les méthodes statistiques tout d'abord, parce qu'elles sont précisément conçues pour basculer de l'observation à la loi, et les techniques de l'intelligence artificielle ensuite, parce que leur vocation est de permettre aux ordinateurs de traiter de la connaissance plutôt que de l'information.

Les réseaux bayésiens sont le résultat d'une convergence entre ces deux disciplines et constituent aujourd'hui l'un des formalismes les plus complets et les plus cohérents pour l'acquisition, la représentation et l'utilisation des connaissances par les ordinateurs. Encore du domaine de la recherche au début des années 90, cette technologie connaît de plus en plus d'applications, depuis le contrôle de véhicules autonomes à la modélisation des risques opérationnels, en passant par le datamining et la localisation des gènes.

Les réseaux bayésiens, qui doivent leur nom aux travaux de Thomas Bayes au seizième siècle sur la théorie des probabilités, sont le résultat de recherches qui visaient à intégrer la notion d'incertitudes dans les systèmes experts. Ils offrent ainsi un outil naturel pour résoudre deux problèmes qui apparaissent tout le temps dans les domaines des mathématiques appliquées et de l'ingénierie, l'incertitude et la complexité, et jouent un rôle de plus en plus important dans la conception et l'analyse d'algorithmes de l'apprentissage automatique [98].

Dans le contexte de la détection d'intrusions, domaine de notre travail, les méthodes d'inférence utilisées sont les réseaux Bayésiens qui représentent une approche de classification très efficace. Cette méthode sera présentée dans ce chapitre structuré comme suit : Dans la section suivante on va donner quelques notions sur le calcul probabiliste. On présentera les réseaux bayésiens ainsi que les différentes notions relatives à ce concept, ensuite on va se focaliser sur la notion de la classification en particulier les classificateurs bayésiens, qui seront l'objet d'application de notre travail, on va montrer ainsi les mesures et les procédures de l'apprentissage bayésien. Pour introduire dans la dernière section l'application des réseaux Bayésiens dans le contexte de la détection d'intrusions.

3.2. Calcul Probabiliste

La base du calcul probabiliste est les variables aléatoires (discrètes ou continues). Une variable aléatoire est une variable qui peut prendre un ensemble de valeurs (son domaine) selon des probabilités prédéfinies (la distribution des probabilités conditionnelles, CPD). Dans ce qui va suivre on va considérer:

- *P (A)* : La probabilité d'observer un évènement *A*
- *P (A, B)* : La probabilité jointe d'observer deux évènements *A* et *B* ensemble
- *P (A|B)* : La probabilité conditionnelle d'observer l'évènement *A*, sachant la valeur de l'évènement *B*
- Deux variables sont indépendantes si et seulement si *P (A|B) = P (A)*. Ceci exprime le fait que l'observation de *A* ne dépend pas de *B* : quelque soit la valeur de *B*, nous avons toujours les mêmes probabilités d'observer *A*. L'indépendance est une propriété symétrique : Si A est indépendant de B, alors, automatiquement, B est indépendant de A.
- La base de tout calcul probabiliste est le fameux théorème de Bayes [41] qui exprime une probabilité jointe comme le produit d'une probabilité conditionnelle et d'une autre probabilité jointe.

$$P(A, B) = P(A|B).P(B) = P(B|A).P(A) \qquad (3.1)$$

- Dans le cas où les deux variables aléatoires sont indépendantes le théorème de Bayes devient :

$$P(A, B) = P(A).P(B) \qquad (3.2)$$

- La règle de chaîne (Chain Rule) de Bayes permet d'appliquer le théorème de Bayes récursivement dans le cas où l'on a plus que 2 variables. Par exemple :

$$P(A, B, C) = P(C| A, B).\ P(B|A).\ P(A) \qquad (3.3)$$

3.3. Notions sur les Réseaux Bayésiens

Un réseau bayésien est un modèle graphique probabiliste employé pour modeliser des domaines contenants des incertitudes [41]. C'est un graphe acyclique dirigé (DAG) dans lequel chaque nœud représente une variable aléatoire discrète et contient les états de cette

variable et une table conditionnelle de probabilité (CPT). La CPT d'un nœud contient les probabilités d'un état spécifique de ce nœud étant donné les états de ses pères. Le rapport père fils entre les nœuds dans un réseau bayésien indique la direction de la causalité entre les variables correspondantes. C'est-à-dire, la variable représentée par le nœud fils dépend d'une manière causale de celles représentées par ses pères.

Considérant l'exemple suivant où un fermier a une bouteille de lait qui peut être soit infectée soit net. Il possède également un test qui peut déterminer avec une probabilité élevée si le lait est infecté ou non (c.-à-d., les résultats du test sont soit positifs soit négatifs). Cette situation peut être représentée avec deux variables aléatoires booléennes, *« Infecter »* et *« Positive »*. La variable *« Infecter »* est vraie quand le lait est réellement infecté et à fausse autrement. La variable *« Positive »* est vraie quand le test indique que le lait est infecté et fausse quand les résultats du test sont négatifs. Il faut noter qu'il est possible que le lait soit net quand le test donne des résultats positifs et vice versa [41].

Un réseau bayésien possible qui modélise cette situation est représentée par la *figure 3.1*. Les deux variables aléatoires sont représentées en tant que deux nœuds dans le réseau. On suppose que le fermier connaît la CPT pour la variable *« Positive »*, la probabilité d'un résultat positif étant donné que le lait est infecté et la probabilité d'un résultat de test positif étant donné que le lait est propre. Il connaît également la CPT pour la variable *« Infecter »*, qui détermine la probabilité qu'une bouteille contient du lait infecté. La flèche du nœud *«Infecter »* vers le nœud *« Positive »* indique une relation causale entre ces deux variables respectivement.

Dans ce cas, on attend que les résultats du test dépendent du véritable état du lait (infecté ou net). Les variables sans pères ne sont pas influencées directement par les autres variables. Le nœud qui représente les résultats du test dans la *figure 3.1* s'appelle souvent « *variable d'information »*.

L'état de cette variable est généralement connu ou peut être mesuré d'une manière directe. Le nœud qui représente l'état réel du lait s'appelle « *variable d'hypothèse »*. L'état d'une telle variable ne peut pas être obtenu immédiatement. Le but d'un réseau bayésien est de permettre le calcul de la probabilité des variable(s) d'hypothèse(s) étant donné l'information recueillie par les variables d'informations.

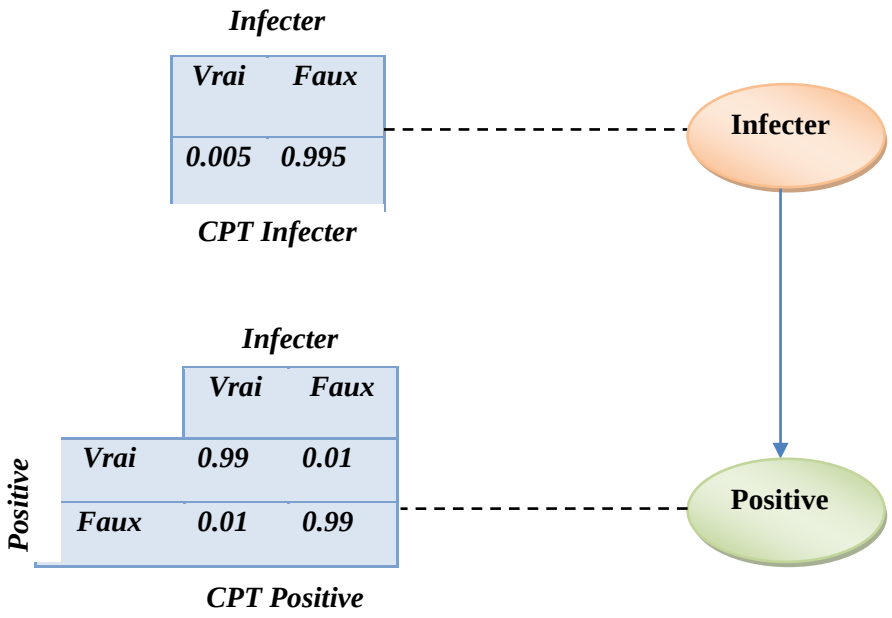


Figure 3.1 : Exemple d'un Réseau Bayésien et ses CPT [41].

Dans notre exemple, le fermier peut vouloir calculer la probabilité que le lait est infecté étant donné un résultat positif du test. En entrant l'information (par exemple, *Positive* à vraie) dans le réseau bayésien, la probabilité que « *Infecter*» soit à vraie peut être déduite. La valeur numérique de cette probabilité, appelée la probabilité « *à posteriori* », étant donnée l'information observée, est *0.33*. Intuitivement, on s'attendait à une valeur plus élevée, en particulier en considérant que le test est très précis. Cependant, la probabilité initiale faible de l'infection du lait appelée la probabilité « *à priori* », avant que toutes les observations soient faite, explique ce résultat.

3.3.1. Représentation d'un Système Stochastique avec les Réseaux Bayésiens

Pour bien illustrer la représentation d'un système stochastique avec les réseaux Bayésiens, on considèrera l'exemple suivant, dans lequel tous les nœuds sont binaires. Cet exemple modélise un micro portable et la probabilité que celui-ci soit alimenté. L'évènement «le portable est alimenté» (*Alimenter, A = vrai*) a deux causes possibles :

Soit sur batterie « *Batterie, B = vrai* », soit sur secteur *« Secteur, S = vrai »*. La force des relations est représentée dans la CPT de la *figure 3.2*. Par exemple on voit que *P (A= vrai | B = vrai, S = faux)* = la probabilité que le portable soit alimenté sachant que la batterie est chargée et que le secteur est débranché = *0.9* et comme la somme de chaque ligne doit être égale à 1, on a automatiquement *P (A = faux | B = vrai, S = faux) = 1 - 0.9 = 0.1*. Comme le

nœud « source d'électricité » (Electricité, *E = vrai*) n'a pas de parents, sa CPT spécifie la fréquence d'avoir une source d'alimentation (la probabilité à priori, qui peut provenir d'une étude statistique) dans ce cas elle est égale à *0.5*.

La relation d'indépendance conditionnelle la plus simple encodée dans un réseau Bayésien peut être formulée comme suit : « Toute variable dans le contexte de ses parents, est indépendante des variables non descendantes », où les notions de frères et parents découlent de la topologie du graphe. En utilisant la règle de Bayes, on peut obtenir la probabilité jointe de tous les nœuds de ce graphe :

$$P(A, B, S, E) = P(E|A, B, S).\ P(S|A, B).\ P(B|A).\ P(A) \qquad (3.4)$$

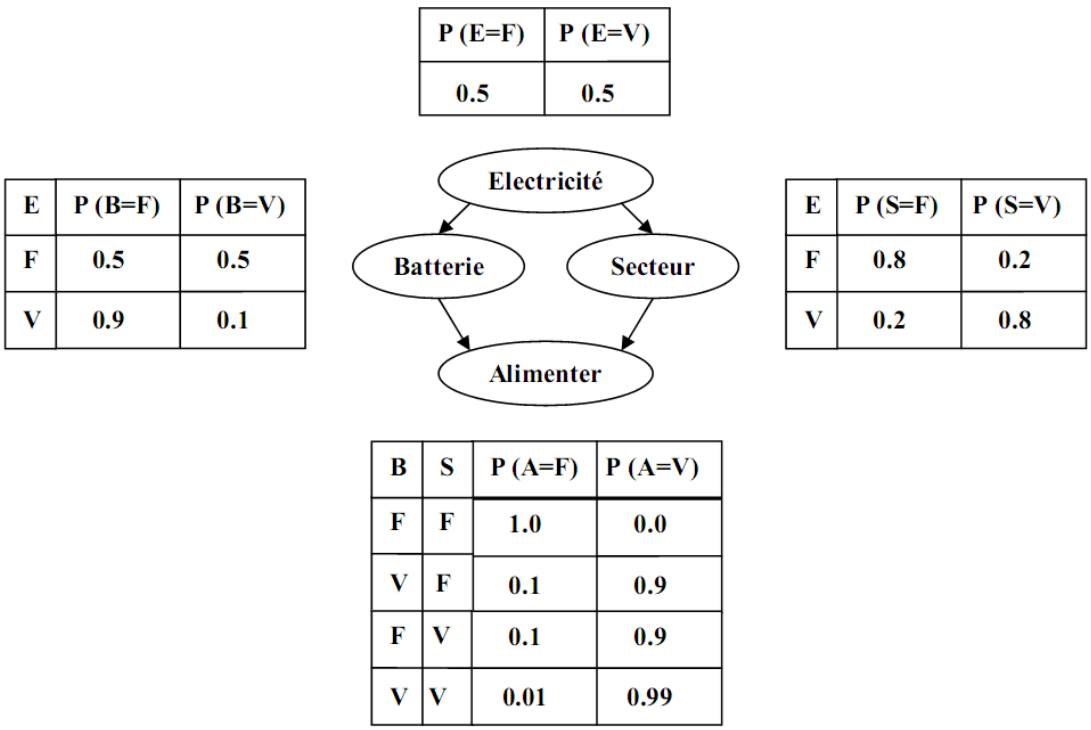


Figure 3. 2. Exemple de représentation d'un système stochastique en réseau Bayésien.

Les relations d'indépendance vont permettre par la suite de simplifier la formule précédente, d'abord on a *B* est indépendant de *S*, moyennant la connaissance de la valeur de leur père *E* qui est déjà présente dans l'équation, ce qui va simplifier le deuxième terme. Ensuite le premier terme peut être simplifié aussi car *A* est indépendant de *E*, sachant les valeurs de ses deux pères *B* et *S*.

$$P(A, B, S, E) = P(A).P(B|A).P(S|A).P(E|B, S) \qquad (3.5)$$

On voit que les relations d'indépendances conditionnelles ont permis de compacter la formule précédente. Dans cet exemple, les économies sont minimes, mais dans le cas général de *n* nœuds binaires, la probabilité jointe complète nécessite un espace de *O(2n)* pour être représentée, tandis que la forme factorisée ne nécessite que $O(n \cdot 2^k)$, ou k est le nombre maximal de parents d'un nœud.

3.3.2. Inférence dans un Réseau Bayésien

Pour montrer comment effectuer une inférence dans un réseau Bayésien reprenons l'exemple déjà cité. On suppose qu'on observe l'évènement « le micro portable est alimenté ». Il y a deux explications possibles pour ça : soit sur batterie, soit le secteur est branché. Quel est la plus probable? Pour cela on peut utiliser la règle de Bayes pour calculer la probabilité postérieure pour chaque explication :

$$P\big(B = V \,\big|\, A = V\big) = \frac{P(B=V,A=V)}{p(A=V)} = \frac{\sum_{e,s} P(E=e,B=V,S=s,A=V)}{P(A=V)} = \frac{0.2781}{0.6471} = 0.43 \qquad (3.6)$$

$$P\big(S = V \,\big|\, A = V\big) = \frac{p(S=V,A=V)}{p(A=V)} = \frac{\sum_{e,b} p(E=e,B=b,S=V,A=V)}{p(A=V)} = \frac{0.4581}{0.6471} = 0.708 \qquad (3.7)$$

Avec : $P(A = V) = \sum_{e,b,s} P(E = e, B = b, S = s, A = V) = 0.6471$ qui représente une constante de normalisation, égale à la vraisemblance des données.

On remarque donc qu'il est plus probable que le portable soit alimenté sur secteur. Dans cet exemple, on a comme évidence un effet (portable alimenté) et on a inféré sur les causes les plus probables. Ceci s'appelle effectuer un diagnostic, ou alors raisonner du bas vers le haut (Bottom Up Reasoning) car l'inférence va des effets vers les causes. Ceci est une tâche très commune dans les systèmes experts. Les raisonnements causaux, du haut vers le bas (Top Down Reasoning) peuvent être aussi effectués par les réseaux Bayésiens. Par exemple, on peut calculer la probabilité que le portable soit alimenté sachant qu'il y a une source d'alimentation. Pour cette raison, les réseaux Bayésiens sont aussi appelés des *modèles génératifs*, car ils spécifient comment les causes génèrent des effets.

3.3.3. Apprentissage dans les Réseaux Bayésiens

Pour décrire un réseau Bayésien on doit spécifier sa topologie (la structure du graphe) ainsi que toutes les tables de distribution des probabilités conditionnelles. Il est possible

d’avoir le graphe et les distributions à partir des données expérimentales. Cependant, il est possible d’avoir des valeurs qu’on ne peut pas observer pendant les expériences ainsi certains nœuds restent cachés, ou que des données manquent. Dans ce cas, l’apprentissage des paramètres devient plus compliqué. Sachant qu’il existe plusieurs travaux qui le traite [42] [43]. Dans la suite on ne va pas traiter l’apprentissage dans sa généralité, mais on va s’intéresser seulement à celui qui concerne le problème de la classification. Mais avant cela, on va donner d’abord une définition de la notion de classification.

3.4. Concept de Classification

La classification est une tâche d'apprentissage automatique dont le but est la détermination d'une fonction qui permet d'affecter des instances à des classes qui représentent des ensembles de catégories discrètes, en se basant sur les valeurs de plusieurs *attributs*. Formellement, ces instances représentent les vecteurs de valeurs de ces attributs et les catégories discrètes sont appelées *les variables, labels ou étiquettes de classes*. La construction de la fonction de classification est réalisée pendant une phase d’*apprentissage* à l’aide d’instances déjà classifiées. Ce type d'apprentissage s'appelle l'*apprentissage supervisé* puisque l'étude est basée sur des données classifiées.

3.4.1. Méthodes de Classification

Beaucoup d'approches de classification tentent à construire explicitement une fonction à partir de l'ensemble commun des valeurs d’attributs pour obtenir les labels des classes.
Parmi ces classificateurs : les arbres de décision, les règles de décision, les réseaux de neurones [44]. Toutes ces méthodes sont comparables en termes d'exactitude de classification, pour bien illustrer le principe de fonctionnement de ces méthodes on va présenter ici l’exemple des arbres de décision. Les arbres de décision sont parmi les techniques d'apprentissage automatique les plus connues. Un arbre de décision se compose de trois éléments de base : Le nœud de décision qui spécifie l'attribut de test, l’arc ou la branche correspondante à l'une des valeurs possibles de l'attribut qui représente aussi l'une des modalités de test de l'attribut et la feuille qui est également appelée le nœud réponse et elle contient la classe à laquelle l'objet appartient. Dans les arbres de décision, deux phases de traitement devraient être assurées :

- La construction de l'arbre : Cette tâche est réalisée en se basant sur un ensemble donné

d'entraînement. Cela consiste à choisir pour chaque nœud de décision l'attribut de test approprié et aussi définir la classe marquée par chaque feuille.

- La classification : Afin de classifier une nouvelle instance, on commence par la racine de l'arbre de décision, ensuite on teste l'attribut spécifié par ce nœud. Le résultat de ce test permet de se déplacer en bas selon la branche relative à la valeur de test de cet attribut. Ce processus sera répété jusqu'à ce qu'une feuille soit atteinte. L'instance sera donc classifiée dans la même classe que celle qui caractérise la feuille obtenue. Plusieurs algorithmes ont été développés afin d'assurer la construction des arbres de décision et leurs utilisations pour la classification des tâches. Les algorithmes *ID3* Et *C4.5* développés par [33] sont probablement les plus utilisés. On peut aussi mentionner l'algorithme CART de [17]. La majorité de ces algorithmes emploient une stratégie descendante, c- à- d. de la racine vers les feuilles.

Pour assurer cette procédure, les paramètres génériques suivants sont nécessaires :

- La mesure de sélection de l'attribut: Tenant compte de la puissance séparée de chaque attribut sur les classes pour choisir le meilleur comme racine de l'arbre de décision. En d'autres termes, cette mesure devra considérer les capacités de chaque attribut pour déterminer les classes objets d'entraînement.
- La stratégie de partitionnement: Ayant comme objectif la division de l'ensemble courant d'entraînement, en tenant compte de l'attribut de test choisi.
- Le critère d'arrêt: Qui représente les conditions d'arrêt de la croissance d'une partie de l'arbre de décision (ou même l'arbre complet). En d'autres termes, il détermine si un sous- ensemble d'entraînement sera encore divisé. Dans le reste de cette section on va s'intéresser à une autre méthode de classification qui utilise un procédé un peu différent de celui des méthodes citées précédemment, à savoir la classification avec les réseaux Bayésiens.

3.4.2. Classification avec les Réseaux Bayésiens

Les réseaux Bayésiens adoptent une approche un peu différente par rapport à d'autres méthodes de classification puisqu'ils fournissent des informations utiles sur la structure du problème lui-même. Dans cette approche, on approxime la distribution jointe de probabilité de la classe et des attributs $P(C, A_1, \ldots A_k)$, *Avec C* la variable aléatoire décrivant la classe, $A_1 \ldots A_k$ sont les variables aléatoires décrivant les attributs. Ainsi, l'apprentissage avec la

classification bayésienne se résume d'abord à l'estimation de cette distribution jointe de probabilité. Après la construction d'une telle estimation, on classifie les nouvelles instances en examinant la probabilité conditionnelle de *C* sachant les valeurs particulières des attributs pour obtenir la classe la plus probable.

L'approche standard de la classification bayésienne utilise la règle de *chaîne* pour décomposer cette distribution conjointe de probabilité :

$$P(C, A_1,\ldots..A_k)=P(C)\,P(A_1,\ldots A_k \mid C) \qquad (3.8)$$

P(C) représente la probabilité antérieure (à priori) de la classe. Celle-ci peut être directement estimée à partir des données d'entraînements, ou à partir d'un échantillon plus grand de la population. Par exemple, on peut obtenir des statistiques sur le nombre d'occurrences de cancer des poumons dans une population plus générale de cette maladie. $P(A_1,\ldots A_k \mid C)$ est la distribution des valeurs d'attributs sachant la valeur de la variable classe. L'estimation de ce terme est généralement plus complexe, et elle sera traitée par la suite.

Une fois que nous avons une estimation de *P(C)* et de $P(A_1,\ldots A_k \mid C)$ on peut utiliser la règle de Bayes pour obtenir la probabilité conditionnelle de la classe sachant les valeurs des attributs :

$$P(C,A_1,\ldots..A_k)= \alpha\, P(C)P(A_1,\ldots A_k \mid C) \qquad (3.9)$$

Avec α un facteur de normalisation qui assure que la probabilité conditionnelle de toutes les valeurs possibles de la variable classe se résume à (3.8) (Dans la pratique, on n'a pas besoin d'évaluer explicitement ce facteur parce qu'il est constant pour un exemple donné.). En utilisant (3.9) on peut classifier les nouvelles instances en combinant la probabilité à priori de chaque classe avec la probabilité des valeurs d'attributs sachant la valeur de cette classe. La classification bayésienne n'effectue pas un apprentissage explicite d'une règle de décision puisque l'apprentissage se réduit à une estimation de probabilités. Par conséquence elle a quelques différences par rapport à d'autres approches de classification telle l'exactitude asymptotique.

a. Exactitude Asymptotique de la Classification Bayésienne

Une propriété de base dont on a besoin souvent dans ce type de problème (classification) est l'*exactitude asymptotique*; le système de classification devrait donner les

meilleurs résultats possibles si on lui fournit un nombre suffisant d'instances pendant la phase d'apprentissage, tout en ignorant les limitations du calcul.

Il est démontrable que l'induction d'un classificateur bayésien peut être asymptotiquement optimale (minimiser les erreurs de classification étant donné un ensemble d'entraînement suffisamment grand). Si la méthode d'estimation de $P(A_1,\ldots A_k \mid C)$ est consistante, elle va converger a une distribution conditionnelle sous-jacente étant donné un échantillon d'entraînement suffisamment grand. Ainsi, les propriétés asymptotiques dépendent du choix des méthodes d'estimation de $P(A_1,\ldots A_k \mid C)$. Il faut noter que contrairement à quelques méthodes d'apprentissage, avec la classification bayésienne il est possible que la classe des hypothèses considérées, contient un classificateur optimal, mais on ne peut pas l'apprendre même avec un volume infini de données. Ceci peut se produire si le modèle probabiliste qui correspond à cette règle de classification optimale ne fournit pas la meilleure approximation de la distribution de la probabilité observée. Cette garantie asymptotique suggère que si la connaissance du domaine mène à croire qu'un modèle particulier (classe d'hypothèse) pour $P(A_1,\ldots A_k \mid C)$ va permettre une bonne approximation de la vraie distribution, alors on doit attendre que le classificateur bayésien ait de bonnes performances.

b. Avantages de la Classification Bayésienne

La sémantique probabiliste de la classification bayésienne produit les avantages suivants par rapport à d'autres méthodes. En premier lieu, la classification bayésienne peut être combinée avec d'autres méthodes pour traiter les fonctions de perte asymétriques. Par exemple, dans la sélection du cancer, un mauvais diagnostic d'une tumeur malveillante est plus coûteux qu'un mauvais diagnostic d'une tumeur bénigne, ainsi la détection du cancer pendant une phase avancée peut améliorer les chances de traiter ce dernier. Pour traiter de telles situations, on peut compter sur la théorie de décision pour fournir des méthodes d'évaluation de probabilités combinée avec l'utilité ou (le coût) des différentes décisions [45].

En second lieu, les méthodes probabilistes permettent de traiter les valeurs absentes dans la classification en utilisant la moyenne des valeurs possibles que l'attribut peut prendre.

L'apprentissage peut être mené avec des considérations similaires concernons ces valeurs absentes, malgré que cela a des conséquences sur le coût du calcul [13]. En fin, la sémantique probabiliste fournit une manière claire d'utiliser la connaissance antérieure du domaine, et la connaissance recueillie à partir d'autres ressources (par exemple, les différentes

données d'entraînement) dans le processus de classification. Cette connaissance peut être employée de différentes manières. Par exemple, la connaissance antérieure peut déterminer le type de modèle qu'on utilise pour estimer $P(A_1,\ldots A_k \mid C)$.

Pour la reconnaissance de la parole, par exemple, les attributs sont des mesures du signal parlé, et le modèle probabiliste est un HMM [46] qui se compose généralement des modèles de phonèmes. Ce modèle fortement structuré est incité par notre connaissance antérieure sur la parole. Il faut noter que le choix du modèle reflète généralement notre connaissance au sujet du processus qui a produit les observations.

La connaissance antérieure peut être également employée d'autres façons. Par exemple, elle peut être employée pour déterminer notre évaluation antérieure des probabilités. Ceci mène à changer notre évaluation par des valeurs spécifiques. Si les données d'entraînement pour un paramètre particulier du modèle sont clairsemées, alors l'estimation finale dépend fortement de l'antérieur, et s'il y a des données d'entraînement suffisantes, alors l'estimation finale n'est généralement pas sensible à l'antérieur. En plus, la sémantique probabiliste et les outils de représentation (tels que les réseaux probabilistes [20]) permettent de combiner l'apprentissage avec les suppositions et les connaissances de modélisation du domaine. C'est-à-dire, qu'on peut fixer à l'avance une partie du modèle et d'apprendre les autres parties.

3.5. Détection d'Intrusions

On rappelle que la détection d'intrusions peut être définie comme le processus de l'identification d'un comportement malveillant qui vise un réseau et ses ressources. Les systèmes de détection d'intrusions sont généralement classifiés en ceux utilisant une approche comportementale (anomaly-based) ou bien ceux qui utilisent une approche par scénarios (missue-based). Le principe de fonctionnement des techniques comportementales est basé sur l'utilisation des modèles, ou des profils concernant le comportement normal des utilisateurs, des applications et du trafic réseau. Les déviations établies par rapport à ces modèles seront interprétées en tant qu'attaques. Les systèmes de détection d'anomalies ont l'avantage d'identifier des attaques précédemment inconnues. En définissant un état prévu et normal, ainsi n'importe quel comportement anormal peut être détecté, qu'il fait partie du modèle des menaces ou non. Cependant, cet avantage est acquitté par un grand nombre des faux positifs (c.-à-d., classer un évènement en tant que malveillant alors qu'il est légitime). Ceci peut rendre

le système inutilisable par la désensibilisation de l'administrateur système par un grand nombre d'alertes incorrectes.

Les systèmes qui utilisent des techniques basées sur des scénarios suivent une approche qui est complémentaire de la précédente. Ces approches ont un certain nombre de descriptions d'attaques appelées « signatures » qui correspondent à un flux d'observation des données d'audit pour des évidences d'attaques modelées. Les données d'audit peuvent être recueillies à partir du réseau, du système d'exploitation, ou bien des fichiers log des applications. Les systèmes basés sur les signatures ont l'avantage de produire généralement peu de faux positives.

Généralement, Les approches classiques de détection d'intrusions posent un problème de décision [47], car avec ces systèmes, la classification d'un évènement comme normal ou bien anormal est faite d'une manière simpliste avec une collection de modèles qui évaluent les différents attributs de cet évènement. Ces modèles renvoient un score d'anomalies ou bien une valeur de probabilité qui reflète la normalité de cet évènement selon les profils courants. Cependant, le système est confronté à la tâche de l'agrégation des différentes valeurs renvoyées par les modèles dans un résultat simple et final. La difficulté est le fait que cette agrégation n'est pas facile à calculer, particulièrement quand les rendements individuels des modèles diffèrent d'une manière significative.

Dans la plupart des systèmes courants, le problème est résolu en calculant la somme des sorties et en la comparant avec un seuil statique. L'inconvénient de cette approche est le fait que ce seuil doit être assez petit pour détecter les évènements malveillants qui se manifestent seulement dans un seul dispositif (c.-à-d., un seul attribut qui produit une valeur élevée indiquant un comportement malveillant). Ceci peut mener aussi à des faux positifs parce que les évènements qui produisent beaucoup de déviations légères du profil vont recevoir des scores agrégés qui excèdent le seuil.

Les réseaux Bayésiens représentent une solution meilleure pour répondre à ce problème en remplaçant le processus simple de décision basé seuil. Au lieu de calculer la somme des différentes valeurs renvoyées par les modèles et de comparer le résultat à un seuil, le procédé de décision bayésien va permettre de classifier les évènements en entrée et d'agréger les différentes sorties des modèles d'une manière plus significative. Ce processus va permettre aussi d'introduire les informations supplémentaires disponibles pour les décisions de

la détection.

3.6 Réseaux Bayésiens pour la Détection d'Intrusions

En pratique, un réseau bayésien peut être envisagé au même titre que d'autres modèles : réseaux neuronaux, systèmes experts, arbres de décision, modèles d'analyses de données, arbre de défaillances, modèles logiques, etc. Différents critères, comme la facilité, le coût, et le délai de mise en œuvre d'une solution interviennent lors du choix d'une méthode. Parmi les aspects qui rendent les réseaux bayésiens préférables à d'autres modèles, dans de nombreux cas, on peut citer :

- ✓ *Acquisition des connaissances*: La possibilité du rassemblement et du fusionnement des connaissances de diverses natures dans un même modèle. Dans plusieurs domaines d'application, malgré la présence des sources d'information, elles sont souvent insuffisantes individuellement pour donner une représentation précise et réaliste du système analysé.
- ✓ *Représentation des connaissances*: La représentation graphique explicite, intuitive et compréhensible d'un réseau bayésien, rend facile sa validation et surtout son utilisation même par des non spécialistes. Généralement, un décideur est beaucoup plus incliné à utiliser un modèle dont il comprend le fonctionnement qu'à faire confiance à une boîte noire.
- ✓ *Utilisation des connaissances*: L'effort de la construction d'un réseau Bayésien peut être rentabilisé par la polyvalence de ce dernier, puisqu'un même modèle peut être utilisé pour l'évaluation, la prévision, le diagnostic, et l'optimisation des décisions. A ces fins, notre système utilise l'approche des réseaux Bayésiens proposée pour la détection d'intrusions. Dans notre cas, nous opèrerons à partir de l'analyse des paquets TCP ainsi que des données des fichiers log système. Ainsi dans cette section on va passer en revue les principes d'opérabilité des IDS, ensuite on va examiner l'ébauche de la classification bayésienne dans le contexte de la détection d'intrusions avec ses avantages et ses limitations.

3.6.1 Apprentissage des paramètres

L'apprentissage des paramètres d'un réseau bayésien se fait à partir de données relatives au problème à modéliser. Toutefois, ces données peuvent être complètes ou incomplètes. Les algorithmes d'apprentissage des paramètres ne sont pas les mêmes dans ces

deux cas. Dans le cas où toutes les variables sont observées (critère : Apprentissage de paramètres données complètes), la méthode la plus simple et la plus utilisée est l'estimation statistique de la probabilité d'un évènement par la fréquence d'apparition de l'évènement dans la base de données. Cette méthode est appelée maximum de vraisemblance (celle utilisée dans notre approche). Une autre méthode dite "estimation bayésienne" est aussi utilisée. Elle suit un principe différent. Elle consiste à trouver les paramètres les plus probables sachant que les données ont été observées, en utilisant des a priori sur les paramètres. Dans le cas où les données ne sont pas complètes (critère : Apprentissage de paramètres données incomplètes), la méthode d'estimation des paramètres couramment utilisée est fondée sur l'algorithme itératif EM (Expectation Maximisation).

a. Apprentissage de structure

Lorsque la structure du réseau bayésien n'est pas fournie a priori par un expert, il est possible d'apprendre cette structure à partir d'une base de données. Toutefois, dans la réalité cette base de données peut présenter un manque d'informations pour certaines mesures. Beaucoup de travaux [24] [12] sont intéressés par ce genre de problèmes et la recherche d'éventuelles solutions. Des approches ont étés proposées pour l'apprentissage de structure pour chacun des cas : données complètes ou incomplètes.

Dans le cas où notre base de données est complète (critère : Apprentissage de structure : données complètes), c'est-à-dire toutes les mesures sont complètes, deux familles d'approches ont été proposées.

La première famille (algorithmes IC, PC etc.) consiste à déterminer dans un premier temps un graphe non orienté en tenant compte des différentes indépendances conditionnelles qui existent entre les variables de ce graphe, puis à orienter ce graphe pour obtenir un réseau bayésien. Ces algorithmes sont peu efficaces dans le cas de problèmes de grande taille puisque la détermination de ces indépendances est exponentielle en fonction du nombre des variables.

La deuxième approche consiste à parcourir tous les graphes possibles, associer un score à chaque graphe, puis choisir le graphe ayant le score le plus élevé. Toutefois, cette méthode n'est pas simple, principalement à cause de la taille super exponentielle de l'espace de recherche en fonction du nombre de variables.

Des idées ont été proposées pour résoudre ce problème. Une première consiste à remplacer l'espace de recherche (espace des réseaux bayésiens) par un espace plus petit,

l'espace des arbres (MWST ou Maximal Weight Spanning Tree). Une deuxième idée consiste à ordonner les nœuds pour limiter la recherche des parents possibles pour chaque nœud (algorithme K2). Une troisième consiste à faire une recherche gloutonne dans l'espace des réseaux bayésiens (algorithme GS). Il existe un autre espace dans lequel nous pouvons apprendre les indépendances de la loi de probabilité: l'espace des représentants des classes d'équivalence de Markov. Cet espace est équivalent à l'espace des réseaux bayésiens sauf qu'il ne contient que des exemplaires des classes d'équivalence de Markov, donc un peu plus petit.

En réalité, plusieurs bases de données présentent des manques de mesures (critère : Apprentissage de structure données incomplètes). Ce manque de données revient à plusieurs raisons. Par exemple, un instrument de mesure tombe en panne pour une période, ce qui se traduit par un manque de données pour toutes les séries de mesures durant cette période, le système entre dans un état particulier imprévu alors il répond aléatoirement par l'une des réponses possibles ou il se bloque, ou encore un responsable de la prise de mesures oublie de le faire pour une cause ou une autre (fatigue, sommeil, mesures variant très rapidement ...), ou que la donnée reportée ne soit pas lisible.

Quand nous voulons modéliser le comportement de notre système à partir de cette base de données, nous aurons un problème causé par ces manques de mesures. La première idée qui nous vient à la tête est de filtrer les données et ne prendre que les exemples complètement observés de la base, sauf que nous ne disposons plus de la même quantité de données pour faire l'apprentissage et nous risquons de ne plus avoir qu'un nombre très petit de données dans la base lorsque le taux de valeurs manquantes est élevé. L'idée la plus pertinente est de compléter les données manquantes en utilisant une technique de substitution par la moyenne ou encore la méthode de maximum de vraisemblance.

Afin de faire l'apprentissage de structure de réseau bayésien, les deux approches proposées dans le cas de données complètes (approche statistique et approche basée sur un score) sont aussi valables dans le cas de données incomplètes mais avec des rectifications.

Les réseaux Bayésiens demeurent un outil puissant dans la modélisation de problèmes complexes et le raisonnement à partir de l'incertain. Nous avons introduit dans ce chapitre la notion de ces modèles graphiques probabilistes ainsi que leur principe de raisonnement. Nous avons précisé également un ensemble de critères qui nous aideront à choisir un bon outil de

développement de ces modèles graphiques. Nous présentons dans la partie suivante l'étude comparative que nous avons effectuée pour choisir cet outil.

b. Comparaison des librairies

Pour la manipulation et la programmation des algorithmes traitant les réseaux Bayésiens, plusieurs librairies ont été initiées. Nous citons la librairie BNJ (Bayesian Network Tools in Java) sur laquelle plusieurs travaux de recherche ont été effectués au sein du laboratoire KDD. Nous pouvons utiliser cette librairie comme référence pour notre travail vue sa richesse en termes d'algorithmes traitant les réseaux Bayésiens. D'autres travaux ont donné le jour à d'autres librairies manipulant les modèles graphiques probabilistes (BNT1, JavaBayes, PNL, ProBT).

Le but de cette partie est, dans un premier temps, d'étudier ces librairies : voir ce que fournit chacune d'elles, les algorithmes implémentés, les types de données supportés ainsi que les facilités proposées pour les nouveaux développeurs. Ensuite, nous allons comparer ces librairies et en choisir une avec laquelle nous pouvons manipuler les réseaux bayésiens pour la détection d'intrusion.

- **BNJ**

BNJ3 (Bayesian network tools in Java) est un ensemble d'outils Java de recherche et de développement des réseaux Bayésiens. Ce projet a été développé au sein du laboratoire KDD de l'université du Kensas [47]. C'est un projet Open source distribué sous la licence GNU (General Public Licence). Sa dernière version 3.3+ a été publiée en Avril 2006. Cette version fournit une interface graphique qui facilite la création, la modification, l'importation et l'exportation des réseaux bayésiens. Elle fournit aussi un ensemble d'algorithmes d'inférence pour les réseaux bayésiens.

Nous détaillons dans la suite le contenu de cette boite à outils dans sa version 3.3+. Pour la définition des réseaux Bayésiens dans l'environnement de BNJ v3.3+, l'utilisateur peut utiliser l'interface graphique de ce système. Il est possible de définir deux types de distribution de probabilité pour les nœuds : distribution tabulaire discrète et distribution continue. Les réseaux Bayésiens créés sont stockés dans des fichiers xml.

Algorithmes d'inférence : BNJ v3.3+ fournit un ensemble d'algorithmes d'inférence pour les réseaux Bayésiens. Ces algorithmes se classent en deux catégories : inférence exacte et inférence approchée.

Les algorithmes d'inférence exacte développés sont "Arbre de Jonction", "Elimination des variables avec optimisation", "Singly-connected network belief propagation" ("Pearl") et "Cutset Conditioning". Les algorithmes d'inférence approchée développés dans cette boite à outils sont des méthodes basées sur les algorithmes d'inférence exacte. En effet, certaines méthodes utilisent la notion d'échantillonnage tels que "Adaptive Importance Sampling (AIS)", "Logic Sampling" et "Forward Sampling", d'autres méthodes appliquent les algorithmes d'inférence exacte sur une sélection d'arcs du graphe à traiter tels que "KruskalPolytree", "BCS" et "PTReduction".

Apprentissage : En parcourant l'ensemble des fichiers sources de cette boite à outils nous ne trouvons pas d'implémentation des algorithmes d'apprentissage ni de paramètres ni de structure.

La librairie BNJ est structurée de façon arborescente facilitant ainsi la navigation dans les différents répertoires. Les fichiers du code source sont bien soignés et présentent des informations utiles pour la compréhension du rôle de chaque fichier. En effet, les différentes méthodes sont décrites par un commentaire, de plus, les variables et les méthodes portent des noms significatifs.

Nous pouvons trouver sur le site de BNJ deux fichiers documentant la version BNJ 2.03a, un destiné aux nouveaux utilisateurs BNJ et un autre destiné aux développeurs qui s'intéressent au code source. Mais la dernière version est fournie sans aucune documentation. Nous trouvons aussi un groupe de discussion. Toutefois, les messages (questions et réponses) sur ce groupe de discussion datent de l'année 2005.

Le soin de la structure et du contenu des fichiers sources s'avère alors le seul refuge des nouveaux développeurs.

- **JavaBayes**

JavaBayes [42] est un ensemble d'outils Java qui permettent de créer et de manipuler des réseaux Bayésiens. Il a été développé par Fabio Gagliardi Cozman en 1998 et il a été distribué sous la licence GNU. Ce système est composé d'un éditeur graphique, un noyau d'inférence et un ensemble d'analyseurs syntaxiques. L'éditeur graphique permet à son

utilisateur de créer et de modifier des réseaux Bayésiens. Les analyseurs syntaxiques permettent d'importer et d'exporter des réseaux Bayésiens dans différents formats. Le moteur d'inférence est responsable de la manipulation des données. Il calcule les probabilités marginales et les espérances. Les algorithmes d'inférence implémentés dans ce système sont "Variable Elimination" et "Bucket Tree Elimination". JavaBayes est capable d'utiliser des modèles avec des ensembles de distributions pour calculer les intervalles des distributions à posteriori ou les intervalles des espérances. Nous remarquons que JavaBayes ne propose pas d'algorithmes pour l'apprentissage de paramètres ou de structure. La librairie JavaBayes est structurée à base de répertoires contenant chacun les fichiers relatifs à la même fonctionnalité (inférence, ..). Les méthodes dans Java-Bayes sont peu commentées et beaucoup de variables ont des noms non significatifs ce qui rend la compréhension du code difficile. Sur le site de JavaBayes, nous trouvons de la documentation sous la forme de tutoriels. Ces tutoriels décrivent les étapes pour l'installation, la compilation et la mise en fonction de la librairie. Comme ils expliquent, à travers deux exemples fournis dans la librairie, quelques fonctionnalités (chargement, enregistrement et modification d'un réseau bayésien).

- **PNL**

PNL (Probabilistic Network Library) est une boite à outils qui permet la manipulation des modèles graphiques (réseaux Bayésiens et chaînes de Markov) [44]. Cette librairie est implémentée en C++ et est distribuée sous la licence Open Source de Intel. Elle supporte les modèles dirigés et non dirigés, les variables discrètes et continues, comme elle fournit une variété d'algorithmes d'inférence et d'apprentissage. Nous détaillons dans la suite ce que fournit cette librairie. Contrairement aux deux librairies précédentes, PNL ne fournit pas d'interface graphique pour la création et la visualisation des graphes.

***Inférence* :** Les programmeurs de PNL se sont intéressés à deux types de modèles graphiques : les modèles statiques et les modèles dynamiques. En effet, la librairie PNL propose un certain nombre d'algorithmes d'inférence et d'apprentissage pour chacun des deux types. Les algorithmes d'inférence pour les modèles graphiques statiques se divisent en deux sous catégories : inférence exacte et inférence approchée. Les algorithmes d'inférence exacte dans PNL sont "Inférence naïve", "Arbre de Jonction" et "Pearl Inférence". Les algorithmes d'inférence approchée dans PNL sont "Gibbs Sampling Inférence" et "LW Sampling

Inférence". "2TBN Inférence", "1 5Slice Inférence", "1 5Slice JTree Inférence", "BK Inférence" et "2TPF Inférence" sont les algorithmes d'inférence pour les modèles graphiques dynamiques dans PNL.

Apprentissage **:** Comme c'est le cas pour les algorithmes d'inférence, PNL propose des algorithmes d'apprentissage pour les modèles graphiques statiques ainsi que pour ceux dynamiques. Certains de ces algorithmes sont relatifs à l'apprentissage des paramètres, d'autres sont relatifs à l'apprentissage de la structure. Nous citons l'algorithme "EMLearningEngine" qui se base sur l'algorithme Expectation-Maximisation pour l'apprentissage des paramètres dans les réseaux Bayésiens dont les nœuds sont discrets ou continus gaussiens et dans les modèles de Markov dont les nœuds sont discrets. Nous citons aussi l'algorithme "BayesLearningEngine" et "BICLearningEngine".

"MlStaticStructLearnHC" et "MWST" (développé pendant mon stage de master) sont deux algorithmes d'apprentissage de la structure d'un réseau Bayésien dont les données d'entrée sont complètes. PNL propose un algorithme d'apprentissage de structure dans le cas de données incomplètes, cet algorithme (Structural EM) se base sur les deux notions EM et Hill-climbing. Nous avons ajouté dans cette catégorie l'algorithme "MWST-EM".

Pour les modèles dynamiques, nous trouvons un algorithme basé sur l'algorithme EM pour l'apprentissage des paramètres et nous trouvons aussi un algorithme basé sur Hill-climbing pour l'apprentissage de structure dans le cas de données complètes. La librairie PNL comprend un grand nombre de classes. Les fichiers de codes sources sont tous mis dans un seul répertoire sans aucune structuration. Le code C++ ne présente pas de commentaires, heureusement les méthodes et les variables portent des noms significatifs.

Le projet PNL comprend, en plus de classes composantes la librairie, des exemples d'utilisation de ces classes (exemple de création d'un réseau bayésien, d'invocation des différents moteurs d'inférence et d'apprentissage ...). Ces exemples ne constituent pas toute la documentation pour la librairie PNL. En effet, ce projet fournit de la documentation sous forme d'un guide d'utilisation et un manuel de références. Ce document est structuré autour de trois parties :

- Une vue d'ensemble pour la librairie.
- Un guide d'utilisation illustrant les différents algorithmes implémentés ainsi que la façon d'interaction entre ces algorithmes via quelques exemples simples.

- Un manuel de références énumérant les différentes classes de la librairie, les méthodes de chaque classe ainsi que leurs rôles et la signification de leurs paramètres. Cette partie présente en plus la structure de la librairie à travers les relations entre les classes (relation d'héritage).

Sur le site de PNL nous trouvons un forum où figure tout l'historique des questions et réponses des anciens membres, organisé par thème. Une fois membre, nous pouvons poser des questions, mais ce n'est pas sûr que nous aurons des réponses, même pas justes, puisque ce ne sont pas des spécialistes en PNL qui répondent systématiquement. Ce forum peut servir dans plusieurs situations mais pas souvent.

- **ProBT**

ProBT [47] est une librairie C++ pour la manipulation des réseaux Bayésiens. Elle a été initiée depuis 10 ans au sein de la société ProBayes7. Les institutions CNRS8, INRIA9, UJF10 et INPG11 tiennent tous les droits de propriété sur le logiciel ProBT. L'exploitation commerciale et industrielle de ce logiciel a été concédée d'une façon exclusive à la Société Probayes. La Société Probayes donne son accord au laboratoire GRAVIR pour la distribution libre (gratuite) du code d'objet de la bibliothèque ProBT à la communauté scientifique pour qu'il puisse être utilisé pour la recherche et des buts éducatifs. La librairie ProBT comprend deux parties : une interface d'application (API) pour la création des réseaux Bayésiens, et un moteur d'inférence bayésien (BIE) permettant d'exécuter tout le calcul de probabilité dans les modes exacts ou approximatifs.

ProBT supporte les deux types de variables discrètes et continues pour la définition des réseaux Bayésiens, comme elle fournit un ensemble d'algorithmes d'inférence :

Pour l'inférence exacte, ProBT utilise "l'Algorithme de Restrictions Successives" (SRA pour Successive Restrictions Algorithm). Pour l'inférence approchée, plusieurs arrangements d'approximation sont utilisés par ProBT comme Monte-Carlo, l'évaluation simultanée et la Maximisation. ProBT propose aussi des algorithmes pour l'apprentissage des paramètres. Dans le cas de données complètes, elle propose un algorithme à base du maximum de vraisemblance et un algorithme basé sur le principe de *EM* dans le cas de données incomplètes.

ProBT ne contenait pas d'implémentations pour l'apprentissage de structure des réseaux Bayésiens. Un travail a été développé (dans le cadre d'un stage de Master, sous la direction de Mr. Philippe LERAY) pour l'implémentation des algorithmes suivants :

- L'algorithme MWST et K2 pour l'apprentissage de structure à partir de données complètes.
- L'algorithme MWST-EM et K2-EM pour l'apprentissage de structure à partir de données incomplètes.

Le code source des différentes classes de la librairie n'étant pas fourni, seuls les fichiers d'entête sont consultables. Le contenu de ces fichiers est bien commenté, les noms des méthodes et des variables est significatif.

En plus, une documentation riche est fournie avec la librairie. Nous trouvons un guide d'utilisation aidant à mieux comprendre le contenu de la librairie, un document sur la programmation bayésienne et un manuel citant les différentes classes ainsi que leurs méthodes et leurs paramètres. Nous trouvons aussi sur le site de ProBT, un groupe de discussion auquel nous pouvons poser des questions ou même profiter des anciennes discussions dans l'archive.

- **Récapitulatif**

Le tableau 3.1 illustre la comparaison entre le contenu de chacune des librairies étudiées lors de ce travail. Cette étude comparative tient compte des différents critères identifiés dans la partie bibliographique auxquels nous avons ajouté les critères relatifs à la structure et la clarté du code et de la documentation de la librairie (critères : Structure de la librairie, Clarté du code, Documentation et Forums).

Nous constatons que la librairie PNL est plus riche et plus évoluée que les deux autres librairies BNJ et JavaBayes. En effet, PNL, malgré l'absence d'une interface graphique qui permet l'affichage des graphes, propose une richesse par rapport à la librairie BNJ au niveau des algorithmes d'apprentissage et le traitement des graphes dynamiques.

Tableau. 3.1 : comparaison entre le contenu de chacune des librairies.

Librairie	JavaBayes	BNJ	PNL	ProBT
Langage de programmation	Java	Java	C++	C++
Licence	GNU	GNU	Open GL	Licence
Interface graphique	+	++	∅	∅
Variables discrètes	+++	+++	+++	+++
Variables continues	∅	++	+++	+++
Graphes statiques	+++	+++	+++	+++
Graphes dynamiques	∅	∅	+++	+++
Inférence exacte	++	+++	+++	+++
Inférence approchée	∅	++	+++	+++
Apprentissage de paramètres : données complètes	∅	∅	+++	+++
Apprentissage de paramètres : données incomplètes	∅	∅	++	++
Apprentissage de structure : données complètes	∅	∅	++	++
Apprentissage de structure : données incomplètes	∅	∅	++	++
Structure de la librairie	+++	+++	++	++
Clarté du code	+	+++	++	+++
Documentation	+	∅	++	+++
Forums	∅	+	++	+++
Note finale	14/45	22/45	35/45	38/45

La richesse de BNJ par rapport à la librairie JavaBayes est au niveau des algorithmes d'inférence approchée et le traitement des variables continues, dans le but d'intégrer ces différents algorithmes dans le soft *Snort* open source pour améliorer le système de diagnostic de celui-ci, qui sera l'objet du dernier chapitre de cette thèse.

3.7. Conclusion

Dans ce chapitre on a vu que les réseaux Bayésiens représentent un outil de modélisation très puissant qui combine la théorie probabiliste et la théorie des graphes. La théorie probabiliste joue le rôle du « liant », avec laquelle les différentes parties sont combinées, assurant que le système dans son ensemble est consistant, et fournit une interface entre les modèles et les données. La théorie des graphes fournit une interface facile avec laquelle les concepteurs peuvent modéliser des ensembles de variables hautement interactives, mais aussi une structure de donnés qui se prête à la conception d'algorithmes efficaces et généraux.

On a présenté aussi l'un des aspects des réseaux Bayésiens à savoir la classification bayésienne qui permet l'apprentissage de la distribution d'instances étant donné les valeurs des différentes classes. Cette approche, qui a été présentée par [46] est examinée par [42], a des propriétés de calcul attrayantes, en même temps, comme les expériences de [42] ont montré, qu'elle présenté des performances compétitives avec d'autres classificateurs. En plus, il est clair que dans quelques situations, il serait utile de modeler des corrélations, entre les différents attributs, la chose qui ne peut pas être réalisée par une structure arborescente. Ceci devient significatif quand il y a un nombre suffisant d'instances d'entraînement pour estimer d'une manière robuste les probabilités conditionnelles.

Comme le cadre de notre travail et l'application des réseaux Bayésiens pour la détection d'intrusions, on a passé en revue cette notion d'approche, et on a examiné l'embauche de la classification bayésienne dans le contexte de la détection d'intrusions avec ses avantages et ses limitations. Pour le reste de ce travail, nous allons appliquer l'apprentissage des réseaux Bayésiens sur l'ensemble de données KDD'99 (de taille très importante) pour la détection d'intrusions, implémenté dans *Snort* open source qui sera l'objet du dernier chapitre.

CHAPITRE 4
Détection d'Intrusion basée sur les Réseaux Bayésiens : Conception & Implémentation

4.1. Introduction

Comme on a détaillé dans la partie théorique, les algorithmes actuels permettent d'envisager l'apprentissage dans les réseaux Bayésiens d'une façon très complète grâce aux différentes librairies initiées dans ce contexte. Même en l'absence totale de la connaissance, on peut chercher à la fois la structure la plus adaptée du réseau, c'est-à-dire les relations de dépendances et d'indépendances entre les différentes variables, et la quantification de ces relations, c'est-à-dire les valeurs de probabilités conditionnelles entre ces variables. On a vu aussi que l'utilisation première d'un réseau bayésien est le calcul de la probabilité d'une hypothèse connaissant certaines observations.

C'est sur cette requête élémentaire que le problème de classification repose. Pour montrer la mise en œuvre de ces deux notions, à savoir l'apprentissage et la classification, dans une application pratique, le système proposé repose sur l'utilisation du réseau bayésien pour la détection d'intrusions à partir d'évènements réseau. Grâce aux algorithmes étudiés dans le chapitre précédent, nous allons démontrer l'exactitude de ces derniers dans le domaine de diagnostic une fois implémenté dans le logiciel *Snort* open source (figure 4.1).

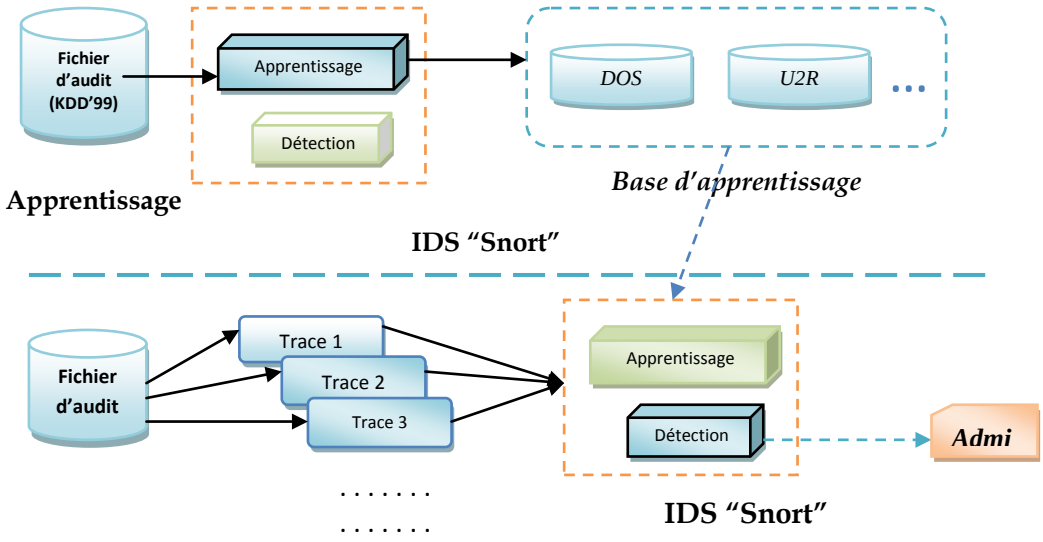


Figure 4.1. Architecture de notre système

Ainsi le reste de ce chapitre sera structuré comme suit; dans la section suivante on va présenter la procédure d'apprentissage, pour présenter par la suite quelques statistiques et manipulations sur les données qu'on va utiliser pour mener cette procédure. Dans les sections 4, 5 donnera les différentes formules et algorithmes utilisés pour l'apprentissage de la structure et des paramètres des réseaux bayésiens. On abordera dans la section 6 la classification par les réseaux obtenus.

4.2. Démarche Attaquant

Le comportement le plus complexe et le plus imprévisible au sein d'un SI « Système d'Information » est le comportement d'un attaquant. L'objectif final d'un attaquant est variable et difficile à déterminer. Ainsi certaines étapes sont nécessaires pour arriver à ses objectifs finaux.

Pour se dissimuler, un attaquant tente de se fondre dans un comportement utilisateur. Le comportement d'un attaquant hérite donc des actions utilisateur. L'un des objectifs intermédiaire indispensable à un attaquant est d'obtenir des droits similaires à ceux d'un administrateur afin de modifier, récupérer ou altérer des données ou paramètres du SI. La figure 4.2 présente l'ensemble des actions possibles d'un attaquant sur le système.

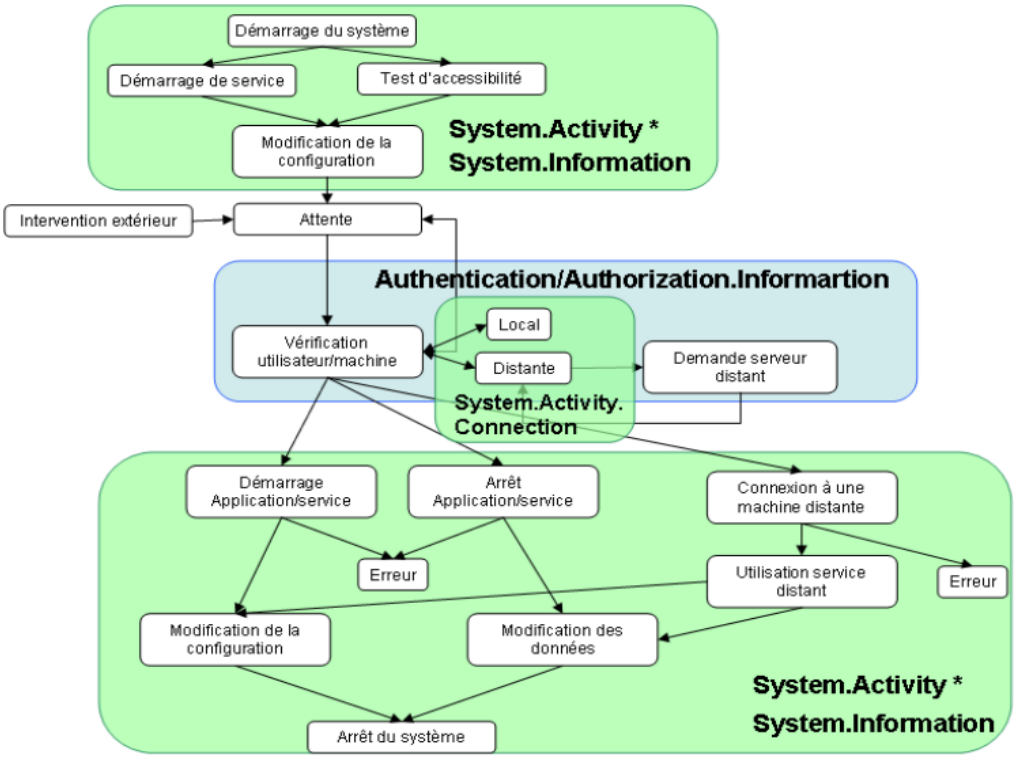


Figure 4.2 : Démarche d'un attaquant [48]

Nous distinguons cinq classes d'objectifs pour un attaquant; le gain de privilège, le gain d'accès, le déni de service, le rebond et l'espionnage. Pour arriver à ses fins, trois ou quatre étapes (selon le scénario utilisé) sont indispensables.

Tout d'abord l'attaquant doit localiser le système à attaquer. Ensuite, il doit récupérer des informations concernant ce système. Une fois la cible localisée et le système analysé (connaissance de l'environnement et de ses vulnérabilités), l'attaquant utilise une faille du système permettant d'atteindre ses objectifs. Un autre processus parallèle existe consistant à utiliser des outils automatiques (virus) qui tentent d'exploiter automatiquement une faille. Une fois le système pénétré, l'attaquant possède un comportement proche de celui de l'utilisateur ou de l'administrateur. Bien que les actions d'un attaquant ne soient pas toujours détectables (nouvelles attaques, absence de sondes), des déviances significatives de comportement lors de l'utilisation d'actions d'utilisateur ou d'administrateur peuvent être décelées. C'est pour cette raison que les actions partagées entre les utilisateurs légitimes (utilisateur classique, administrateur) et les attaquants sont particulièrement importantes pour la détection d'anomalies comportementales.

4.3.Focalisation sur les Points de Passage Obligatoires

Les actions d'un attaquant passent par des goulets d'étranglement appelés Points de Passage Obligatoires (PPO). Afin de sélectionner les indicateurs les plus pertinents nous nous attacherons à définir les PPOs de chaque type de scénarios d'attaques.

Pour être le plus exhaustif possible, nous avons combiné les différents moyens d'attaques décrits dans l'ontologie (voirfigure 4.8) avec la classification des attaques par leur effet de la DARPA [attack-centric taxonomy Lincoln Laboratory1998DARPA].

L'ontologie utilisée dans [49] a défini 25 types d'attaques possibles utilisées comme moyen par les attaquants pour atteindre leur objectif. Cet ensemble d'attaques représente l'arsenal utilisé par un attaquant pour atteindre les effets désirés sur le système. La DARPA a défini une classification des attaques autour de ces effets. Elle préconise de décomposer tout type d'attaques en cinq catégories:

- ☒ ***User to Root*** qui définit un effet d'élévation de privilège depuis un compte connu vers un compte aux privilèges supérieurs,
- ☒ ***Remote to local*** qui définit l'effet d'obtenir un accès à un compte, composant ou service local depuis une connexion distante,

- ***Denial of service*** qui définit l'effet de perte de disponibilité sur un composant ou service du système,
- ***Surveillance/probe*** qui définit l'effet de collecte d'information sur le système ciblé,
- ***System Access/Alter data*** qui définit l'effet de perte d'intégrité et de confidentialité des données du système.

En associant les différents moyens d'attaques et les effets attendus, nous décrivons un ensemble varié de scénarios d'attaques types (voir figure 4.2).

Le trafic réseau enregistré dans l'ensemble de données DARPA 98[annexe] est transformé à des connexions TCP pour former l'ensemble de données de détection d'intrusions de la compétition KDD'99. Spécifiquement, une connexion a été définie comme étant une séquence de paquets TCP commençant et finissant à des moments bien précis entre lesquels le flux de données à partir d'une adresse IP source vers une adresse IP cible sous certains protocoles soit bien déterminé [50]. Chaque connexion est marquée comme étant normale ou comme étant l'une des d'attaques considérées. Sachant que les attaques de type DATA n'ont pas été utilisées dans DARPA 98, les différentes attaques ont été classifiées sous 4 catégories principales :

Dénis de Service (DOS), Utilisateur vers Administrateur (U2R), Distant vers Locale (R2L), Attaques par Sondes (Probe).

Tableau 4.1 : Classifications des actions d'attaques [50]

User To Root (U2R)	***Remote to local (R2L)***	***Denial of Service (DOS)***	***Surveillance/probe***	***System Access/Alter data***
Gain	Backdoor	DDos	InfoGathering	Injection
Injection	Bounce	Dos	Phishing	Overflow
Overflow	Trojan	Trojan	Spyware	Poisoning
Bypass	Virus	Virus		Steal
Trojan	Spoofing	Worm		Bounce
Virus	Injection			Evasion
	BruteForce			Trojan
	Hijacking			Virus
				Physical
				Concealment

Dans cet ensemble de données, 41 attributs sont tirés pour récapituler les informations des connexions. Neuf d'entres eux sont des *« attributs de base »* et les 32 *« attributs additionnels »* restants sont rassemblés dans 3 catégories différentes.
L'étude de l'ensemble des points de passage obligatoires est disponible en annexe.

4.4. Approche utilisée :

Dans ce modèle, les attaques connues constituent des «hypothèses», pouvant «expliquer» les faits observés. On considère qu'une attaque donnée se traduit par des «symptômes», pouvant apparaître sous forme d'évènements dans l'audit, mais aussi de données statistiques comme dans le cas de la détection d'anomalies (occupation mémoire, charge CPU, etc.). Étant donné un ensemble de symptômes, l'inférence bayésienne permet de calculer la probabilité de chaque scénario d'attaque connu. Lorsqu'un scénario affiche une probabilité élevée, une alerte est levée.

La règle d'inférence de Bayes [49] peut s'énoncer de manière simplifiée ainsi :

$$P(A/S) = P(A) \times P(S/A) * a \qquad (4.1)$$

Dans cette notation, P(X|Y) désigne la «probabilité de X étant donné Y». Ainsi, si S est un symptôme et A une attaque particulière, P(S|A) représente la probabilité que l'attaque A fasse apparaître le symptôme S. P(A) est la probabilité générale de l'occurrence de l'attaque A. Étant donné que P(A) aussi bien que P(S|A) sont des données empiriques, connues et fournies par la base d'attaques, on peut aisément calculer P(A|S), la probabilité de l'existence de l'attaque A étant donné le symptôme S (**a** représente ici une simple constante de normalisation des probabilités).

Contrairement à cet exemple, il est très rare en pratique que *S* représente directement un symptôme et *A* une attaque. La véritable signification de ces paramètres est respectivement «l'information courante» (c'est-à-dire une hypothèse) et une «nouvelle information» (un élément venant confirmer ou démentir cette hypothèse).

Un détecteur d'intrusions à inférence bayésienne construit donc récursivement un arbre de décision, selon la règle d'inférence :

$$P(A_{n+1}) = P(A_n|S_n)=P(A_n)*P(S_n|A_n)*a \qquad (4.2)$$

Les données de l'algorithme, c'est-à-dire la base des attaques, contient l'ensemble des hypothèses A_i, dont certaines désignent des attaques connues, ainsi que les probabilités $P(A_i)$ et $P(A_i|S_j)$.

L'information initiale, S_0, correspond à un premier symptôme observé, permettant d'affecter une certaine probabilité à chacune des hypothèses possibles.

L'arrivée de chaque nouvel élément S_i modifie ainsi ces probabilités, donnant lieu à un nouveau nœud de l'arbre. Dans le cas d'une intrusion connue, l'algorithme génèrera finalement un nœud accordant une probabilité élevée à une certaine hypothèse A_q, qui est définie comme étant cette attaque.

Cette méthode présente l'avantage de minimiser en principe le risque qu'un attaquant puisse exploiter sa connaissance de la base des attaques pour passer inaperçu. Chaque élément observé dans l'audit est confronté à différentes hypothèses et un scénario d'attaque est défini comme la présence combinée d'un ensemble de symptômes et non comme une séquence particulière d'évènements. L'élaboration d'un scénario non détectable nécessite la construction d'une série d'opérations réalisant l'attaque voulue, mais dans laquelle rien ne confirme réellement aucune des hypothèses, ce qui est très difficile dans le cas d'une base d'attaques non-triviale. Seules les attaques réellement inconnues dans la base ne seront pas détectées. De plus, si la détection de véritables nouvelles attaques est évidemment exclue, le principe d'inférence permet néanmoins de détecter le nombre de variantes d'une attaque donnée, y compris de nouvelles variantes, et se montre performant notamment dans le cas où l'attaquant cherche à "noyer" son attaque dans du bruit, en générant un grand nombre d'opérations anodines.

En pratique, l'efficacité de cette approche dépend naturellement principalement de la qualité de la base d'attaques, c'est-à-dire : exhaustivité des symptômes définis, pertinence des hypothèses formulées et réalisme des probabilités associées à ces hypothèses.

4.5 Procédure d'Apprentissage

On a montré précédemment qu'un réseau bayésien est constitué à la fois d'un graphe (aspect qualitatif) et d'un ensemble de probabilités conditionnelles (aspect quantitatif). L'apprentissage d'un réseau bayésien doit donc répondre aux deux questions suivantes :

- ☒ Comment trouver la structure du réseau bayésien ?
- ☒ Comment estimer les lois de probabilités conditionnelles ?

On va donc séparer le problème d'apprentissage en deux parties :

1) *Apprentissage de la structure* : Sans aucune hypothèse sur la structure du réseau, rechercher celle, qui, une fois munie des meilleurs paramètres, rende compte le mieux possible des données observées.
2) *Apprentissage des paramètres* : La structure d'un réseau (c'est-à-dire le graphe sous- jacent) étant donnée, rechercher le meilleur jeu de paramètre (les différentes probabilités conditionnelles utilisées dans le graphe) pour rendre compte des données observées.

Comme pour tout problème d'apprentissage, différentes techniques sont utilisées pour mener cette opération selon la disponibilité de données concernant le problème à traiter, ou d'expert dans ce domaine, ainsi ces techniques peuvent se partager en deux grandes familles : Apprentissage à partir des données et acquisition des connaissances d'un expert du domaine.

Dans le domaine de la détection d'intrusions, le nombre de modèles de causalités reliant un certain nombre d'attributs est fini, on peut ainsi envisager de se passer d'expert. On peut alors construire un modèle uniquement à partir des données, en recherchant simplement parmi tous les modèles possibles celui qui représente le mieux la réalité.

4.5.1 Données d'Apprentissage

L'apprentissage présenté par la suite utilise l'ensemble de données proposé dans KDD'99 [50] présenté dans l'annexe, ce corpus est généralement utilisé comme benchmark pour les problèmes de détection d'intrusions. Les laboratoires MIT ont installé un environnement pour acquérir des lignes de données concernant des connexions TCP. Chaque connexion est décrite par 41 dispositifs discrets et continus (par exemple la durée de connexion, le type de protocole, etc.) et marquée comme étant normale, ou bien une attaque, avec un seul type d'attaque par ligne (par exemple Smurf, Perl, etc.).

Dans les expériences qu'on va mener (Apprentissage et évaluation), on manipulera 10% de l'ensemble de données KDD'99 qui correspond à 494019 connexions d'entraînement et à 311029 connexions de test (Tableau 4.2). Pour les variables continues il y a différentes manières pour les traiter afin de construire un réseau bayésien. Parmi les méthodes les plus utilisées : la distribution gaussienne pour les classificateurs bayésiens grâce aux algorithmes de la boite à outils BNJ (décrite au chapitre 3).

Tableau 4.2 : Ensemble d'entrainement

Ensemble d'Entraînement *(494019)*	Ensemble de Test	
	Attaques d'Entraînement *(292300)*	Toutes les Attaques *(311029)*
99.87 %	*97.12 %*	*91.27 %*

L'idée de base de cette méthode repose sur le fait que si on a un ensemble d'individu $\{x_1,\ldots. x_i,\ldots.x_n\}$ et x un nouvel individu, alors pour estimer la densité au point x; on centre un noyau K par exemple gaussien (le plus utilisé) autour de chaque x_i et la contribution de chaque x_i à la densité au point x sera égale à sa densité à ce point. Ainsi la densité estimée au point x sera égale à la somme des contributions de tous les x_i :

$$P(x) = \frac{1}{nh}\sum_{i=1}^{n} k\left(\frac{x-x_i}{h}\right) \qquad (4.3)$$

Avec $k(x) = \frac{1}{\sqrt{2\pi}} e^{-1/2x^2}$ (noyau gaussien)

Le paramètre h sera choisi d'une manière à minimiser l'erreur asymptotique carrée intégrée moyenne AMISE (Asymptotic Mean Integrated Squared Error). Il a été montré que la valeur optimale de h pour un noyau gaussien est égale à :

$$h_{opt} = 3.5 \times n^{-1/3} \qquad (4.4)$$

a. Attributs De base

Les attributs de base peuvent être tirés à partir des en-têtes des paquets sans inspecter le contenu de ces derniers. « *Libpcap* » [47] est utilisé comme analyseur et capteur de paquets sur notre réseau pour déterminer ces attributs de base qui sont :

- ☒ Durée de connexion
- ☒ Type de protocole tel *le TCP, l'UDP, l'ICMP*
- ☒ Type de service tel *le FTP, le HTTP, le TELNET*
- ☒ Flag de statut, qui récapitule la connexion
- ☒ Totale des octets envoyés vers l'hôte destination
- ☒ Totale des octets envoyés à partir de l'hôte source

- Le fait que les adresses de destination et de source soient identiques ou non
- Nombre des faux fragments
- Nombre des paquets urgents

Noter que les types des protocoles et des services sont estimés immédiatement après qu'une connexion soit accomplie.

b. Attributs Additionnels

Comme il a été décrit dans [50], les attributs additionnels tirés à partir de l'ensemble de données KDD'99 sont :

- *Attributs basés Contenus :* La connaissance du domaine est utilisée pour évaluer la charge utile des paquets TCP. Exemples des dispositifs basés contenus sont le nombre de logins non réussis et le fait que l'accès administrateur soit réussi ou non.
- *Attributs basés Temps :* En raison de la nature temporelle des attaques réseau, il est important d'inspecter les paquets dans certain intervalle. Ces attributs sont conçus pour capturer les propriétés dont l'intervalle d'achèvement dépasse une fenêtre temporelle de 2 secondes. Le nombre de connexions au même hôte est un exemple des dispositifs basés temps.
- *Attributs basés hôtes :* Au lieu d'utiliser le facteur temps, ces attributs utilisent une fenêtre d'historique estimée à partir du nombre des connexions, dans ce cas 100 connexions.

Afin d'illustrer les différentes notions concernant les arbres de décision et les réseaux Bayésiens naïfs, nous allons considérer un exemple de base d'apprentissage donné dans le tableau 3, composé de quelques lignes de la base KDD'99. Chaque « connexion », pour des raisons de simplicité, est uniquement décrite par trois attributs discrets (au lieu des 41 attributs que contient la base) qui sont protocol_type, service et flag. Les domaines de ces attributs sont :

$D_{protocol\_type}$ ={tcp, udp} ;
$D_{service}$ = {http, domain_u, auth, private, time}
D_{flag} = {SF (Fin de la syncronisation), REJ (rejet), S0 (premier message de synchronisation), RST0 (la « connexion » mais la source est a abondonée)}

A1 duration	durée de la connexion (nb de secondes)
A2 protocol_type	type du protocole, ex. tcp, udp, icmp…..
A3 service	service réseau (destination) ex. http, telnet
A4 flag	statut de la « connexion » (normal ou erreur)
A5 scr_bytes	nb de données (en octets) de la source vers la destination
A6 dst_bytes	nb de données (en octets) de la destination vers la source
A7 Land	1 si la « connexion » est de/vers le même hôte/port ; 1 sinon
A8 wrong_fragment	nb de fragments « erronés »
A9 urgent	nb de paquets urgents
A10 hot	nb d'indicateur « hot »
A11 num_failed_login	nb d'éssais login rates
A12 logged _in	1 si sucés du login ; 0 sinon
A13 num_compromised	nb de conditions de « compromis »
A14 root_shell	1 si la racine shell est obtenue ; o sinon
A15 su_attempted	1 s'il y'a tentative de la commande « racine » ; 0 sinon
A16 num_root	nbre d'accès a la racine
A17 num_file_creations	nbre de création d'opérations de fichiers
A18 num_shells	nbre de shell prompts
A19 num_access_files	nbre d'opérations sur les fichiers de control d'accès
A20 num_outbound_cmds	nbre de commandes outbound dans une session ftp
A21 is_hot_login	1 si le login appartient a la liste « hot » ; 0 sinon
A22 is_guest_login	1 si le login est login « invite » ; 0 sinon
A23 count	nb de connex. Pour le même hôte
A24 srv_count	nb de connex. Pour le même service
A25 serror_rate	% de connex. Pour le même hôte ayant l'erreur « SYN »
A26 srv_ serror_rate	% de connex. Pour le même service ayant l'erreur « SYN »
A27 rerror_rate	% de connex. Pour le même hôte ayant l'erreur « REJ »
A28 srv_rerror_rate	% de connex. Pour le même service ayant l'erreur « REJ »
A29 same_srv_	% de connex. Pour le même hôte utilisant le même service
A30 diff_srv_rate	% de connex. Pour le même hôte utilisant différents service
A31 srv_diff_host_rate	% de connex. Pour le même service utilisant différents hôte
A32 dst_host_count	nbre de connexion pour le même hôte
A33 dst_host_srv_count	nbre de connex. Pour le même hôte utilisant le même service
A34 dst_host_same_srv_rate	% connex. Pour le même hôte utilisant le même service
A35 dst_host_diff_srv_rate	% connex. Pour le même hôte utilisant différents service
A36 dst_host_same_srv_port_rate	% connex. Pour le même hôte ayant le port source
A37 dst_host_srv_diff_host_rate	% connex. Pour le même hôte et le même service
A38 dst_host_serror_rate	% connex. Pour le même hôte ayant l'erreur « SYN »
A39 dst_host_srv_serror_rate	% connex. Pour le même hôte utilisant le même service « SYN »
A40 dst_host_rerror_rate	% connex. Pour le même hôte ayant l'erreur « REJ »
A41 dst_host_srv_error_rate	% connex. Pour le même hôte utilisant le même service « REJ »

Figure 4.3 : Liste des attributs

4.5.2 Apprentissage de la Structure

La structure du graphe des réseaux Bayésiens détermine les relations de causalité entre les évènements. Notre structure est construite automatiquement par des algorithmes d'apprentissage dans la librairie BNJ vu au chapitre 3 (voir l'arbre de décision figure 4.4).

L'apprentissage de cette structure est réalisé à partir de la base de données KDD'99 dans le détail est en annexe. Toutefois, dans la réalité cette base de données peut présenter un manque d'informations pour certaines mesures. Beaucoup de travaux [24] [12] sont intéressés par ce genre de problèmes et la recherche d'éventuelles solutions. Des approches ont étés proposées pour l'apprentissage de structure pour chacun des cas : données complètes ou incomplètes.

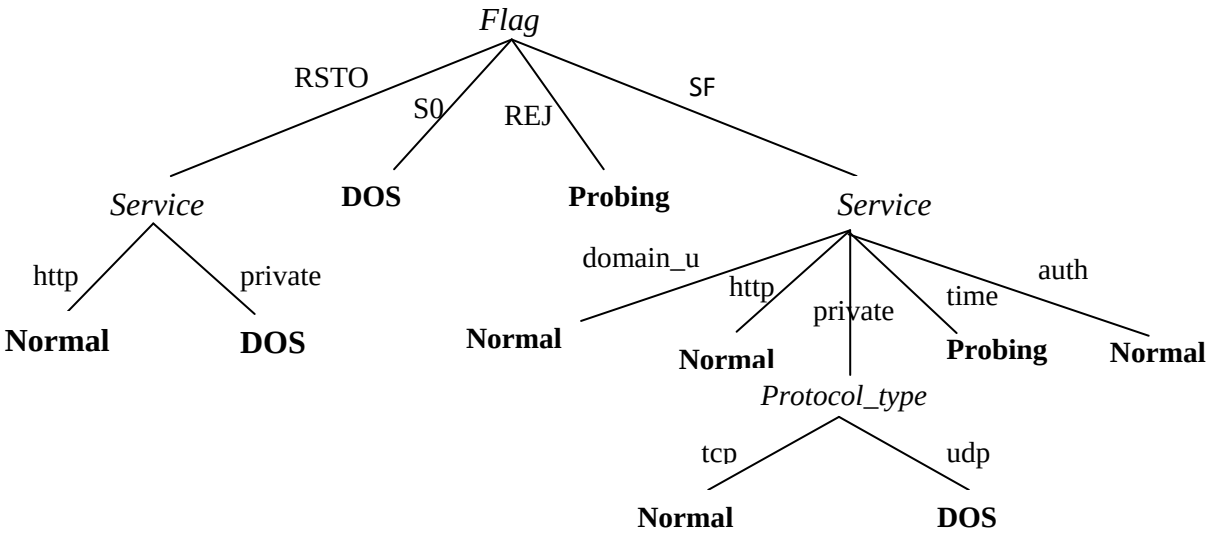


Figure 4.4. Arbre de décision de notre système

Dans le cas où notre base de données est complète (critère : Apprentissage de structure : données complètes), c'est-à-dire toutes les mesures sont complètes, deux familles à approches ont été proposées.

Pour construire la structure (Figure 4.5) à partir des attributs de l'ensemble de données KDD'99 [50], on doit appliquer tous les concepts présentés précédemment, en commençant par calculer les poids des arrêtes qui relient chaque couple de ces attributs (informations mutuelles). Dans notre cas nous allons essayer de calculer cette valeur entre les attributs *« Type de protocole »* et *« Type de service »*, on doit parcourir l'ensemble de données afin de calculer les probabilités pour toutes les combinaisons des valeurs de ces deux attributs avec les différentes valeurs que la variable classe peut prendre (le fait que la connexion représente une attaque ou non). Ainsi si on considère que x_i est la variable «

Type de protocole » et x_j la variable « *Type de service »* alors :

- $x_i \in \{TCP, UDP, ICMP\}$
- $x_j \in \{FTP, HTTP, TELNET, etc.\}$
- $c \in \{normal, perl, rootkit, satan, smurf, etc.\}$

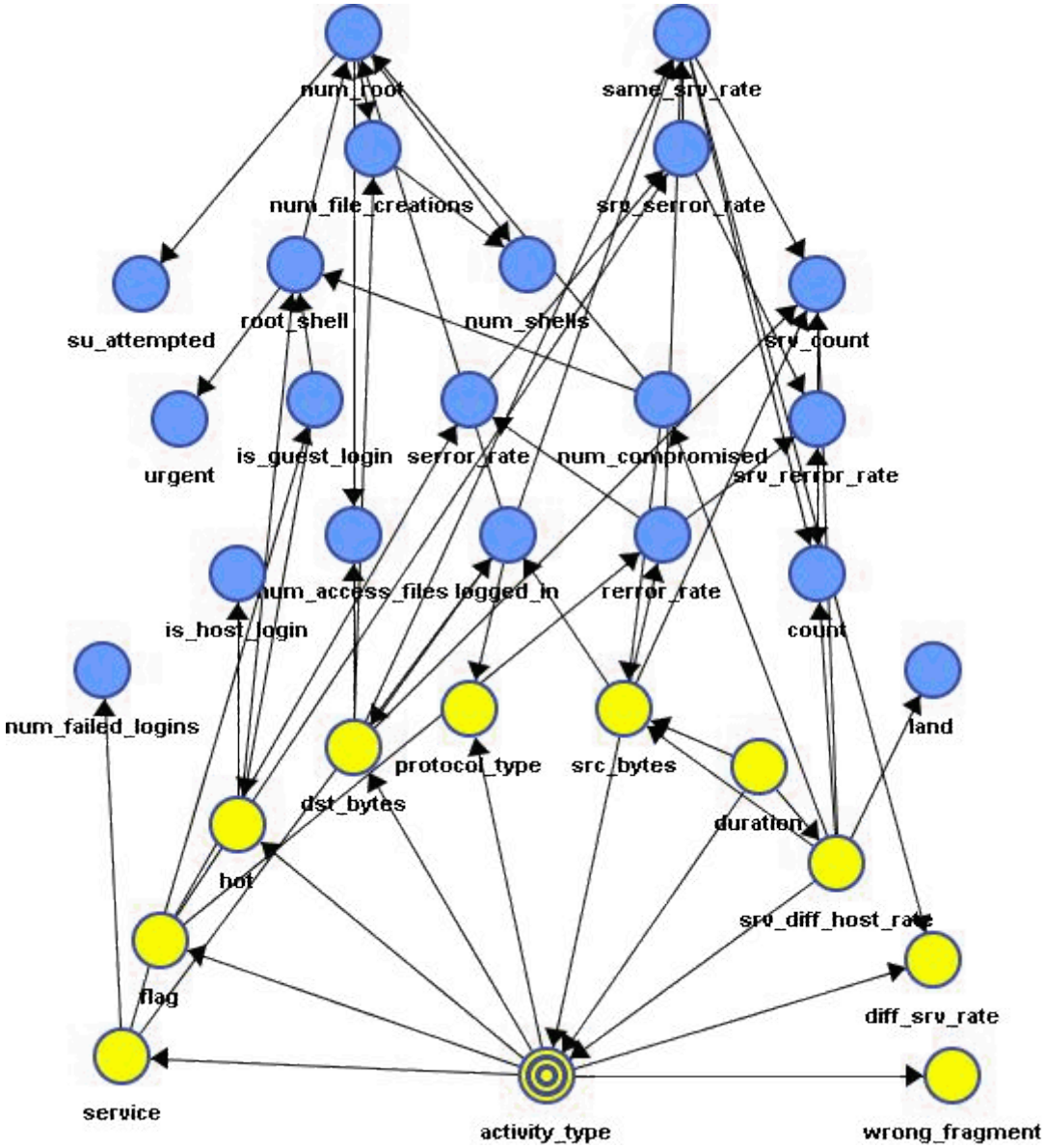


Figure 4.5 : Structure obtenue à partir de l'ensemble des attributs

La librairie BNJ [47] permet automatiquement de déterminer les probabilités des deux attributs « type protocoles et services » par l'intermédiaire de la commande ***CPT*** voir annexe pour plus de détails.

Le programme suivant déduit de la règle de *Snort* open source montre clairement la connexion à la base de données et la partie apprentissage des données KDD'99.

```
String driver = “org.gjt.mm.mysql.Driver”;
String url = “jdbc:mysql://localhost/mydb”;
String login = “mylogon”, passwd = “opensesame”;
public class MyLearner extends ScoreBasedLearner {
public MyLearner(Data d) { super(d); }
public MyLearner(Data d, LearnerScore s) { super(d,s); }
public MyLearner(Data d, LearnerScore s1, LearnerScore s2)
{ super(d,s1,s2); }
List l = new LinkedList();
l.add(“node1”); l.add(“node2”);
BBNCPF cpf = node.getCPF();
BBNCPF cpf = new BBNCPF(l);
(a = true, b = true, c = false):
Hashtable t = new Hashtable();
t.put(“a”, “true”); t.put(“b”, “true”); t.put(“c”, “false”);
double value = c.query(t);
```

Extrait du programme d’apprentissage de notre système

Une fois la connexion faite à la base de données des signatures d’attaques, le fichier log du logiciel Snort sera décomposé et le journal d’alarmes des attaques est généré.

```
[**] [1:1201:7] ATTACK-RESPONSES 403 Forbidden [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/20-14:05:02.375081 193.194.83.196:80 -> 166.153.147.168:38225
TCP TTL:63 TOS:0x0 ID:19781 IpLen:20 DgmLen:602 DF
***AP*** Seq: 0x70C9F8FF Ack: 0x131A130 Win: 0x16D0 TcpLen: 20
[**] [1:553:6] POLICY FTP anonymous login attempt [**]
[Classification: Misc activity] [Priority: 3]
11/20-14:05:44.591309 189.138.117.34:16578 -> 189.195.45.196:21
TCP TTL:124 TOS:0x0 ID:43273 IpLen:20 DgmLen:56 DF
***AP*** Seq: 0x4AE6FEE Ack: 0x734E03DA Win: 0x21FE TcpLen: 20
```

Extrait du journal des alarmes générées

4.6 Apprentissage des Paramètres

L'apprentissage des paramètres d'un réseau bayésien se fait à partir de données relatives au problème à modéliser. Toutefois, ces données peuvent être complètes ou incomplètes. Les algorithmes d'apprentissage des paramètres ne sont pas les mêmes dans ces deux cas. Dans le cas où toutes les variables sont observées (critère : Apprentissage de paramètres données complètes), la méthode la plus simple et la plus utilisée est l'estimation statistique de la probabilité d'un évènement par la fréquence d'apparition de l'évènement dans la base de données. Cette méthode est appelée maximum de vraisemblance.

La structure du réseau étant connue, il reste ainsi à déterminer les probabilités conditionnelles de chaque nœud du réseau, la question qui se pose est : où peut on trouver ces probabilités ? Il est en effet assez peu réaliste de penser qu'un expert pourra fournir de façon numérique l'ensemble des paramètres nécessaires à l'inférence dans un graphe. Il est ainsi intéressant de déterminer ces paramètres à partir d'une base d'instances. L'estimation de distributions de probabilités dans le cadre des réseaux bayésiens est un sujet très vaste et complexe. Dans le cas où toutes les variables sont observées, la méthode la plus simple et la plus utilisée est l'*estimation statistique* qui consiste à estimer la probabilité d'un évènement par la fréquence d'apparition de cet évènement dans la base de données. Cette approche, appelée « *Maximum de vraisemblance (MV)* », nous donne alors :

$$P(X_i = x_k | pa(X_i) = x_j) = \hat{\theta}_{i,j,k}^{MV} = \frac{N_{i,j,k}}{\sum_k N_{i,j,k}} \qquad (4.4)$$

Où $N_{i,j,k}$ est le nombre d'évènements dans la base de données pour lesquels la variable X_i est dans l'état x_k et ses parents sont dans la configuration x_j.

Une autre méthode dite "estimation bayésienne" est aussi utilisée. Elle suit un principe différent. Elle consiste à trouver les paramètres les plus probables sachant que les données ont été observées, en utilisant des a priori sur les paramètres. Dans le cas où les données ne sont pas complètes (critère : Apprentissage de paramètres données incomplètes), la méthode d'estimation des paramètres couramment utilisée est fondée sur l'algorithme itératif EM (Expectation Maximisation) dont nous détaillons le principe dans l'annexe. L'annexe 4 présente une étude plus détaillée pour l'apprentissage de structure de réseau bayésien à partir de données complètes et de données incomplètes.

Tableau 4.3 : Ensemble d'apprentissage

Protocol_type	*Service*	*Flag*	*C*
Tcp	http	SF	Normal
Tcp	http	RSTO	Normal
Tcp	http	REJ	Probing
Tcp	time	SF	Probing
Tcp	time	S0	DOS
Tcp	auth	SF	Normal
Tcp	auth	S0	DOS
Tcp	private	SF	Normal
Tcp	private	SF	Normal
Tcp	private	REJ	Probing
Tcp	private	RSTO	DOS
udp	private	S0	DOS
udp	domain_u	SF	Normal
Tcp	private	SF	DOS
Tcp	http	RSTO	Normal
Tcp	private	RSTO	DOS
Udp	http	SF	Normal
udp	private	SF	DOS
udp	domain_u	SF	Normal

Par conséquent, en présence d'un ensemble d'apprentissage, la seule investigation à faire est de calculer les probabilités conditionnelles puisque la structure du réseau est unique. Ce calcul peut être résumé comme suit :

- Les probabilités conditionnelles pour les attributs discrets sont calculées à partir des fréquences en comptant le nombre d'apparitions de chaque valeur d'attribut avec chacune des valeurs que le nœud parent peut prendre;
- Les attributs continus sont généralement traités en supposant qu'ils suivent une distribution de probabilité gaussienne (c'est-à-dire normale). Donc, pour chaque valeur de classe c_i et chaque attribut continu, nous devons calculer la moyenne μ et l'écart type σ donnée par le tableau 4.4 (voir annexe) qui vont nous servir pour le calcul de la fonction de densité de probabilité pour chaque valeur de classe c_i et chaque valeur a_k de A_k

$$f(a_k) = \frac{1}{\sqrt{2\pi}\ \sigma}\ e^{-\frac{(a_k-\mu)^2}{2\sigma^2}} \qquad (4.5)$$

Tableau 4.4 : Extrait du vecteur de normalité

Attributs numériques	Moyenne	Écart type
duration	216.66	1359.22
src_bytes	1157.056	34226.301
dst_bytes	3384.668	37578.391
wrong_fragment	0	0
urgent	0	0.01
hot	0.045	0.858
num_failed_logins	0	0.021
num_compromised	0.029	4.047
num_root	0.056	4.53
num_file_creations	0.005	0.203
num_shells	0	0.021
num_access_files	0.005	0.081
num_outbound_cmds	0	0
count	8.163	17.712

Le calcul des probabilités conditionnelles et se basant a priori sur les fréquences peut s'avérer entaché d'erreur si la valeur d'un attribut n'apparaît pas avec toutes les classes dans l'ensemble d'apprentissage. En effet, ceci peut entraîner des probabilités conditionnelles nulles qui vont réduire à zéro les probabilités de certaines classes. La technique standard pour éviter ce problème s'appelle estimateur de Laplace et elle consiste à ajouter 1 à tous les numérateurs et de compenser ces ajouts dans les dominateurs. En utilisant cet estimateur, le calcul des probabilités conditionnelles est donné dans le tableau 4.5, le calcul de ces probabilité est généré automatique grâce à la commande BBNPDF : CPT de la librairie BNJ (voir annexe) pour la fonction de distribution de probabilité.

Tableau 4.5 : Calcul des probabilités conditionnelles

P (c)			***P(protocol_type|C)***				***P(service|C)***				***P(flag|C)***			
N.	DOS	Prob		N.	DOS	Prob.		N.	DOS	Prob		N.	DOS	Prob
10/22	8/22	4/22	Tcp	8/11	6/9	4/5	http	5/14	1/12	2/8	SF	8/13	3/11	2/7
			Udp	3/11	3/9	1/5	domain_u	3/14	1/12	1/8	REJ	1/13	1/11	3/7
							auth	2/14	2/12	1/8	S0	1/13	4/11	1/7
							private	3/14	6/12	2/8	RSTO	3/13	3/11	1/7
							time	1/14	2/12	2/8				

4.7 Classification Bayésienne

D'un point de vu intuitif, l'*inférence* dans un réseau de causalités consiste à propager une ou plusieurs informations certaines au sein de ce réseau, pour en déduire comment sont modifiées les croyances concernant les autres nœuds.

Le problème d'inférence dans un réseau bayésien est uniquement un problème de calcul. Il n'y a aucun problème théorique; en effet, la distribution de probabilité étant entièrement définie, on peut (en principe) tout calculer. Pour les réseaux Bayésiens on parle plutôt d'un problème de *classification*, en considérant le nœud racine pour être une variable cachée représentant la classe à laquelle chaque objet dans l'ensemble de test devrait appartenir et les autres nœuds fils représentent les différents attributs spécifiant cet objet. Par conséquent, une fois la structure du réseau déterminée et quantifiée en calculant les différentes tables de probabilités conditionnelles de chacun des nœuds de cette structure, il est ainsi possible de classifier n'importe quel nouvel objet étant donné les valeurs de ses attributs en utilisant la règle de bayes exprimée par :

$$P(c_i|A) = \frac{P(A|c_i).P(c_i)}{P(A)} \qquad (4.6)$$

Avec c_i une valeur possible de la classe session et *A* est l'information globale portée par les valeurs des nœuds attributs. L'évidence *A* peut être distribuée sur un ensemble de valeurs $a_1, a_2, \ldots\ldots\ldots a_n$ relativement aux attributs $A_1, A_2, \ldots\ldots\ldots A_n$ respectivement.

Chaque attribut dépend au plus d'un autre attribut et de la variable de classe C, leur probabilité combinée est obtenue comme suit :

$$P(c_i|A) = \frac{P(a_1|pa_1, c_i).P(a_2|pa_2, c_i) \ldots\ldots\ldots P(a_n|pa_n, c_i).P(c_i)}{P(A)} \qquad (4.7)$$

Avec Pa_i l'attribut présenté par le nœud parent de celui représentant l'attribut a_i dans le réseau. Il est à noter qu'il n'est pas nécessaire de calculer explicitement le dénominateur $P(A)$ parce qu'il représente une constante de normalisation.

4.8 Conclusion

Un réseau bayésien est un moyen de représenter la connaissance d'un système. Une telle représentation n'est bien entendu pas une fin en soi; elle s'effectue, selon les contextes, dans le but de :

- Prévoir le comportement du système
- Diagnostiquer les causes d'un phénomène observé dans le système.
- Contrôler le comportement du système.
- Simuler le comportement du système.
- Analyser des données relatives au système.
- Prendre des décisions concernant le système.

Dans ce chapitre on a présenté les différentes étapes à suivre pour mener la construction d'un réseau bayésien pour la détection d'intrusions, en partant de la phase de l'apprentissage de la structure et des paramètres de ce, dernier, vers la phase de l'inférence, en exploitant un ensemble de données très élaboré utilisé pour la conception et la réalisation de tels systèmes. Dans le chapitre suivant on va présenter les différents résultats des expériences qu'on a menées pour tester les performances du système obtenu en implémentant les différents algorithmes sur le logiciel *Snort* open source. Dans le chapitre suivant on va présenter les différents résultats des expériences qu'on a menées pour tester les performances du système obtenu.

CHAPITRE 5
Implémentation de notre approche Tests & Résultats

5.1. Introduction

L'expérimentation est un élément indispensable pour mesurer la qualité de tout système. Dans la détection d'intrusion, deux types d'expérimentations sont généralement menées. En effet, nous distinguons les jeux de tests expérimentaux qui se rapprochent le plus possible des conditions d'utilisation réelle, des jeux de tests unitaires ("cas d'écoles") qui représentent généralement des données choisies permettant d'illustrer les approches utilisées. Les Systèmes de Détection d'Intrusion possédant une vision locale du système informatique utilisent ces différents jeux de tests pour valider leur approche. Des jeux de tests comme ceux fournis par la DARPA sont adaptés pour tester des systèmes de détection locale.

Dans les expériences qu'on a menées, on a manipulé 10% de l'ensemble complet des données KDD'99 [50], ce qui correspond à 494019 connexions d'entraînement et 311029 connexions de test, avec 18729 connexions qui représentent des attaques nouvelles qui n'apparaissent pas dans l'ensemble d'entraînement. Les différentes expérimentations réalisées supposent que les états des connexions sont certainement connus ce qui n'est pas toujours le cas dans le trafic TCP/IP réel. La stratégie des expérimentations présentées par la suite est basée sur les points suivants:

Deux niveaux de granularités d'attaques: On peut se focaliser sur trois cas, relativement aux différentes attaques afin de manipuler:

1. Toutes les classes d'attaques qui apparaissent dans l'ensemble des données KDD'99 en plus de la classe normale.
2. Les quatre catégories d'attaques (i.e. Dos, R2L, U2R, Probing) (voir annexe).

Dans les deux cas on doit procéder à un apprentissage de la nouvelle structure ainsi que les tables de probabilités relatives à celle-ci, à partir du nouvel ensemble de données d'entraînement obtenu.

Rassemblement après Classification : Le premier cas étudié avec cette stratégie est celui où on s'intéresse à grouper les résultats de la classification en quatre catégories

d'attaques (Dos, R2L, U2R, Probing) (figure 5.1). Dans ce cas l'ensemble des données d'entraînement demeure inchangé. Chaque connexion classifiée comme étant l'une des 38 attaques, à l'aide de cette structure apprise à partir de l'ensemble originaire des données d'entraînement est affectée à l'une des quatre catégories d'attaques à laquelle elle appartient après classification.

Tableau 5.1 Ensembles des Tests réalisés

Ensemble d'Entraînement *(494019)*	**Ensemble de Test**	
	Attaques d'Entraînement *(292300)*	**Toutes les Attaques** *(311029)*
99.87 %	*97.12 %*	*91.27 %*

User To Root (U2R)	***Remote to local (R2L)***	***Denial of Service (DOS)***	***Surveillance/probe***
Buffer_overflow	Ftp_write	Apache2	Ipsweep
httptunnel	Guess_password	Back	Mscan
Loadmodule	Imap	Land	Nmap
Xterm	Multihop	Mailbomb	Portsweep
Perl	Named	Neptune	Saint
Rootkit	Phf	pod	Satan
	Dict	Processtable	
	Snmpguess	Smurf	
	sqlattack	Teardrop	
	xlock	Udpstorm	
	xsnoop		
	Guest		
	Spy		

Figure 5.1 : Classification des attaques

L'ensemble des tests et résultats trouvés ont été évalués par le logiciel open source *Snort* après implémentation des différents algorithmes de la librairie ***BNJ*** étudiés en chapitre 3 et 4 sur celui-ci et en comparant la méthode de diagnostic classique de *Snort*

pour la détection d'intrusion avec celle proposée par notre approche.

En fait, alors qu'avec *Snort* on parle généralement de règles, on parlera plutôt ici de scenarios, la règle ne devenant qu'un cas particulier de scenario.

5.2. Jeu de tests : banc de tests

Nous présentons dans cette section les différentes expériences réalisées pour évaluer les performances de cette approche pour le filtrage de plusieurs signatures d'attaques (Intérieurs, Extérieurs) (Figure 5.2). Nous développons un module à part qui implémente les divers algorithmes d'apprentissage et de détection d'attaques. L'implémentation de ces derniers dans *Snort* open source nous permettra de tester la méthode proposée.

Durant notre expérimentation, nous testons le filtrage sur des trafics simulés et des trafics réels, pendant 3 mois de tests sur le serveur de l'université de Blida (USDB).

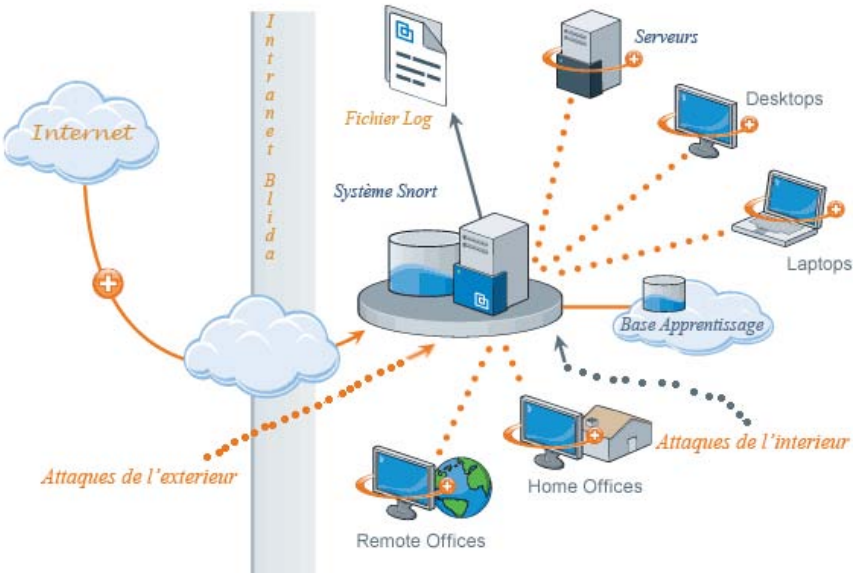


Figure 5.2 : Architecture de notre système

Le but du trafic simulé du log1 est de valider le bon fonctionnement des méthodes de filtrage dans *Snort*. Nous essayons de détecter des phrases que nous insérons dans des segments TCP et des paquets IP envoyées en désordre. Le log2 représente également un trafic simulé et comporte un grand nombre de fragments et de longues sessions TCP réalisé grâce au logiciel de simulation d'attaque DoS (log1 et log 2). Nous testons via ce procédé l'influence de ce type de trafic sur nos algorithmes. En particulier, nous évaluons la gestion des listes de traces. Enfin log3 et log4 comportent des trafics réels dont les caractéristiques sont représentées au Tableau 5.2.

Tableau 5.2 : Caractéristiques du trafic

	Type	***Taille (Mo)***	***TCP (%)***	***UDP (%)***	***ICMP (%)***	***Autres (%)***	***Paquets Fragmentés***	***Flux TCP***
Log 1	Sim	2.10^{-3}	80	10	10	0	2	1
Log 2	Sim	1.2	50	25	25	0	2000	1000
Log 3	Réel	10^{2}	90.7	6.7	3.1	01	0	1026
Log 4	Réel	10^{3}	75.3	17.8	6.7	0.2	46	18460

5.3. Snort Open source

Les systèmes de détection d'intrusion en temps réel basés sur un ensemble de règles, tels que *Snort [51]*, sont efficaces pour reconnaitre la signature d'une attaque lorsque celle-ci peut se détecter avec un unique paquet. Cependant, pour des scenarios d'attaques plus complexes impliquant plusieurs paquets, notamment dans le cas de certaines attaques de type déni de service (DOS), l'expressivité des langages utilisés pour les signatures s'avère insuffisante. Justement, le but de notre recherche est d'étendre le langage des signatures de *Snort* de façon à pouvoir exprimer des scénarios d'attaques complexes, comme le cas de l'attaque DOS. L'enrichissement proposé permet d'exprimer simplement ce qui autrement doit être fait par programmation, par exemple a l'aide des préprocesseurs par le biais de la structure réalisée auparavant grâce à la libraire BNJ.

5.3.1 Présentation

Le système de détection d'intrusions *Snort* [51], originellement conçu pour être un renifleur (sniffer) plus évolué que tcpdump [6], s'est avéré petit a petit très efficace comme outil de détection d'intrusions, entant que la spécification des attaques ne concerne qu'un seul paquet. Utilisant, tout comme tcpdump, la librairie de capture de paquets Libpcap [47], il comporte un module de décodage de paquets très évolué permettant de décoder un grand nombre de protocoles de différents niveaux. Au dessus de ce module de décodage se situe un ensemble de préprocesseurs, dont la fonction originale est de régulariser le format des paquets afin de faciliter la tâche de la base de règles.

Le script ci-dessous représente un exemple de signature *Snort*. L'interprétation de cette signature se lit comme suit :

Si un paquet TCP provenant de l'extérieur ($EX- TERNAL NET) pénètre dans notre réseau ($HOME NET), peu importe les ports (any), et que ce paquet a le drapeau ACK activé, de même que les deux bits réservés (flags :A,12), et que le numéro

d'acquiescement est 0 (ack :0), peu importe l'état de la session (stateless), il faut alors signaler un balayage TCP fait avec nmap [6] (msg :"SCAN nmap TCP").

```
signature :
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg :"SCAN nmap TCP" ; stateless ; flags :A,12 ; ack :0 ; reference :arachnids,28 ; classtype :attempted-recon ; sid :628 ; rev :3 ;)

sortie :
02/18-08 :37 :08.585026 [**] [1 :628 :3] SCAN nmap TCP [**] [Classification : Attempted Information Leak] [Priority : 2] TCP 207.46.144.222 :44979 -> 193.194.83.166 :1
```

Extrait de la signature d'attaque dans Snort

Le reste de la signature ne sert pas pour la détection proprement dite, mais constitue plutôt de l'information ajoutée pour documenter l'attaque. Juste en dessous, on voit ce qui est affiché lors de la détection de l'attaque.

5.3.2 Structure de Snort proposée

Avec sa syntaxe et sa structure relativement simples, *Snort* est très efficace pour détecter les attaques se limitant à un seul paquet (environ 2000 à ce jour). Pour les attaques comportant plusieurs paquets, *Snort* est un peu moins efficace. En fait, *Snort* est capable de reconnaitre certaines attaques se déroulant sur plusieurs paquets, tel que le balayage de ports, mais cette détection ne passe pas par sa base de règles. Pour détecter de telles attaques, certains programmeurs ont ajouté des modules qui agissent au même niveau que les préprocesseurs, et qui sont en fait considérés comme tels. Cependant, cette affirmation n'est pas tout à fait exacte : il ne s'agit pas de préprocesseurs au sens pur du terme, car ceux-ci n'ont pas pour rôle de reformater les paquets pour permettre une meilleure détection. Plutôt, ils ne font que détecter par programmation les attaques ne pouvant s'exprimer sur la base d'un seul paquet. C'est pourquoi, sur la figure 5.3, on voit que ces préprocesseurs doivent interagir avec les modules d'affichage.

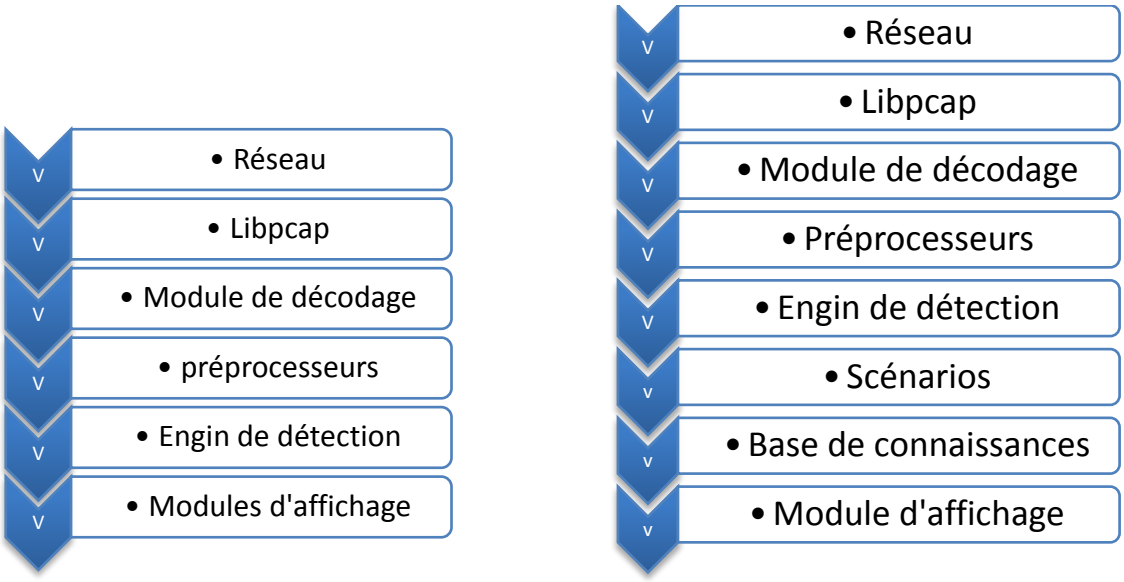


Figure 5.3 : Structure de Snort (à gauche), et celle de l'outil proposé (à droite).

En plus de ces préprocesseurs responsables de reconnaitre les attaques s'étendant sur plus d'un paquet, il y a également un certain nombre de préprocesseurs responsables de conserver en mémoire une certaine partie du contexte. Par exemple, plusieurs attaques ne doivent être considérées comme telles que si elles surviennent dans le contexte d'une session TCP active. Exploiter le fait que *Snort* fonctionnait paquet par paquet pour générer automatiquement des paquets correspondant aux signatures d'attaque, engorgeant ainsi le système de détection d'intrusions de faux-positifs.

Ce que nous proposons ici est d'élargir la syntaxe du langage de signatures que ***Snort*** utilise de façon à pouvoir représenter les attaques s'étendant sur plusieurs paquets et tenir compte du contexte dans lequel une attaque donnée est identifiée par l'aspect d'apprentissage réalisé par la librairie ***BNJ*** (figure 5.4).

Pour représenter le contexte, nous proposons l'utilisation d'une base de connaissances. Cette base de connaissances serait tenue à jour par un nouvel engin de détection de scénarios qui travaille à partir d'évènements fournis par l'engin de détection actuel, pour but de diminuer davantage le nombre de faux-positifs (but de notre recherche).

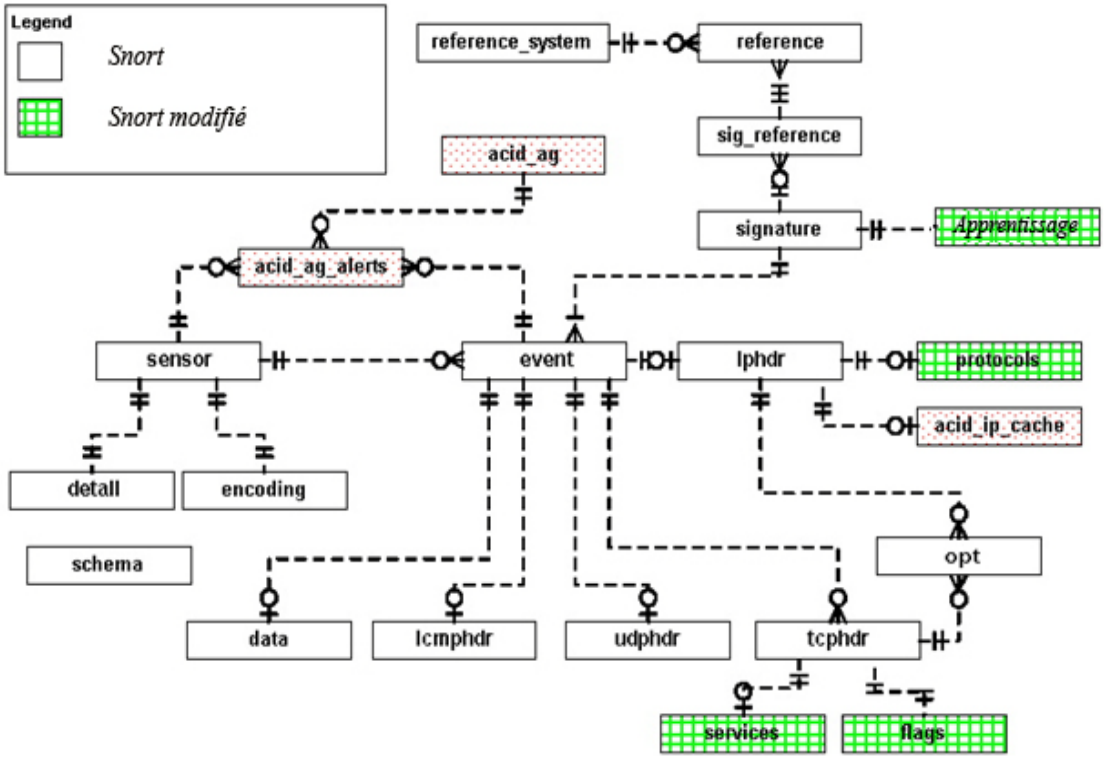


Figure 5.4 : La Structure proposée pour Snort

En plus de la connaissance des connections TCP actives, celle des systèmes d'exploitation. La structure actuelle des règles de *Snort* permet, à l'aide d'un système de références, de savoir quels sont les systèmes d'exploitation vulnérables à une attaque donnée. On pourrait alors vérifier, avant de signaler une attaque, que le système d'exploitation de la victime fait bel et bien partie de ceux concernés. Bien que la connaissance des systèmes d'exploitation soit donnée en entrée par l'administrateur du système, on pourrait aussi imaginer établir un ensemble de règles permettant de générer automatiquement cette connaissance en écoutant sur le réseau. Il s'agirait de rendre passives certaines techniques de prises d'empreintes digitales [4]. Cette façon de faire présente un intérêt particulier dans les environnements hétérogènes où chaque usager est responsable d'administrer son propre système. Dans un second lieu, la connaissance ainsi acquise sur le réseau pourrait être consultée directement par l'administrateur réseau en cas de besoin.

5.4. Résultats des expériences

Cette section présente les résultats expérimentaux, qui permettent d'évaluer l'approche proposée pour la détection des attaques élémentaires. Les résultats de l'ensemble des expériences ont été effectuées au niveau du serveur principal de l'université

de BLIDA, vu le trafic dense auquel est confronté a plein temps de l'extérieur ou de l'intérieur (voir figure 5.1), le système *Snort* une fois modifié, a été installé et mis en marche pour une durée de 3 mois, pour les tests ainsi que les différentes observation effectuées sur ce serveur pour les attaques réel et simuler grâce à des outils d'attaques englobant l'ensemble des classes d'attaques vu auparavant (exp : logiciel DoS ver5.5).

Lors des tests, nous avons constaté 60 évènements usuels collectées pendant cette même période, trois types de scénarios d'attaques (figure) ont été détectés. Ces scénarios sont respectivement un scénario de déni de service (DoS), un scénario de "Brute Force" et un scénario de contamination par un cheval de Troie. Ces scénarios sont constitués à la fois d'évènements usuels (ex : paquets reçus, mails envoyés) et d'évènements anormaux (signature de DoS, Virus détecté).

Tableau 5.3 : Résultats des expériences

	Snort		***Snort Modifié***	
	Alerte	***Temps (s)***	***Alerte***	***Temps (s)***
Log 1	2	0.1	5	0.1
Log 2	0	2.4	1	1.11
Log 3	169	9.17	178	7.62
Log 4	2974	89.76	3017	83.9

5.4.1 Modélisation de signatures de scenarios d'attaques

Les algorithmes d'apprentissage vu au chapitre 4 ont permis de construire le réseau Bayésien ci-dessous grâce à la librairie BNJ. La variable cible, activity_type, est directement connectée aux variables contribuant le plus à la connaissance de cette cible, comme par exemple les variables service et protocol_type.

La modélisation des activités utilisateurs (Figure 5.5) est basée sur le jeu de tests de la phase d'apprentissage. L'ensemble des données ont été traitées une seule fois et a permis la création d'un réseau Bayésien composé de 30 nœuds d'alertes, de 48 liens représentés chacun pas un nœud de lien et un nœud temporel (soit 31 nœuds de lien). Par l'analyse de notre modèle de référence un analyste différentiera rapidement les différentes signatures d'attaques (U2R, U2L, Dos, probing).

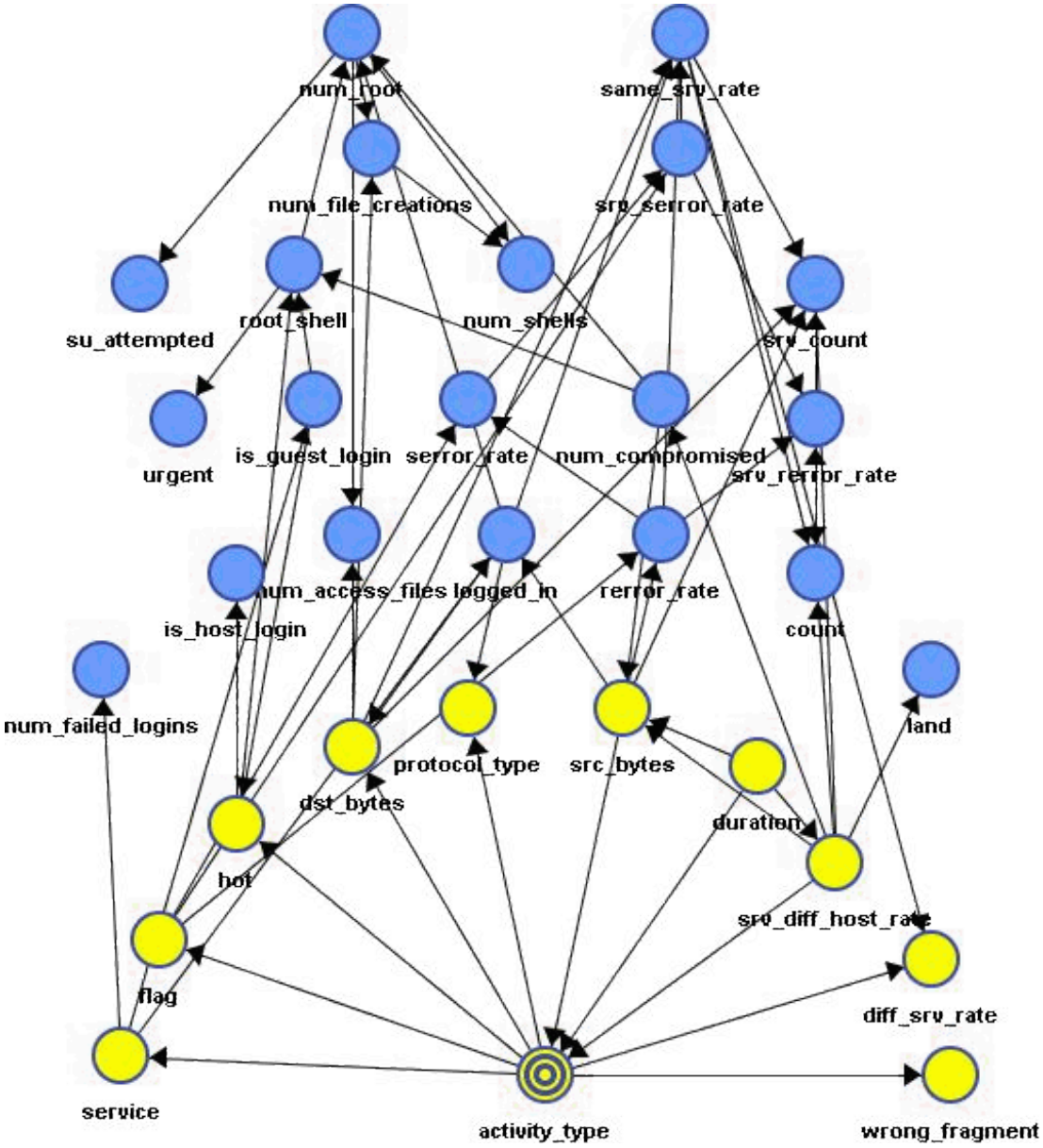


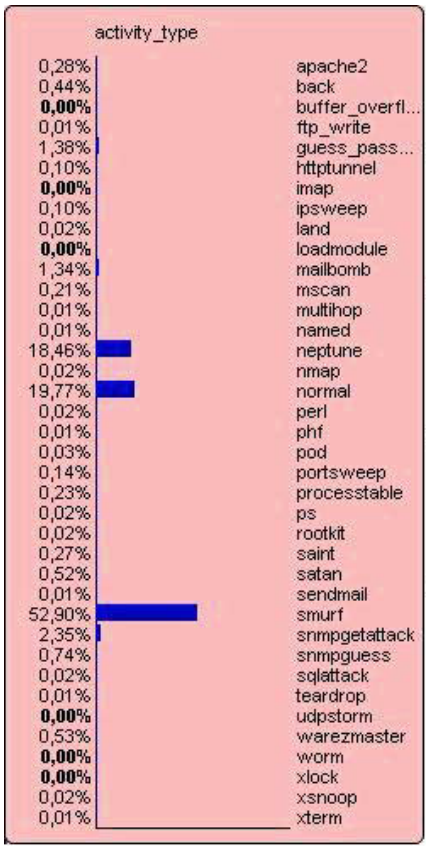
Figure 5.5 : Structure appropriée pour notre système d'intrusion « Snort »

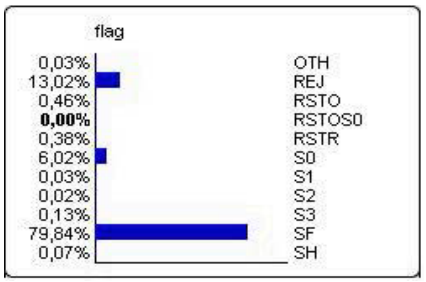
Quelques attaques ont été testées pour voir le lien les symptômes d'intrusion avec les attributs appris auparavant par la table de probabilité conditionnelle dans la section qui suit :

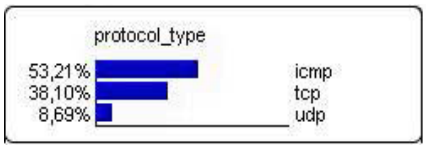
a. Première Observation (Tableau 5.4)

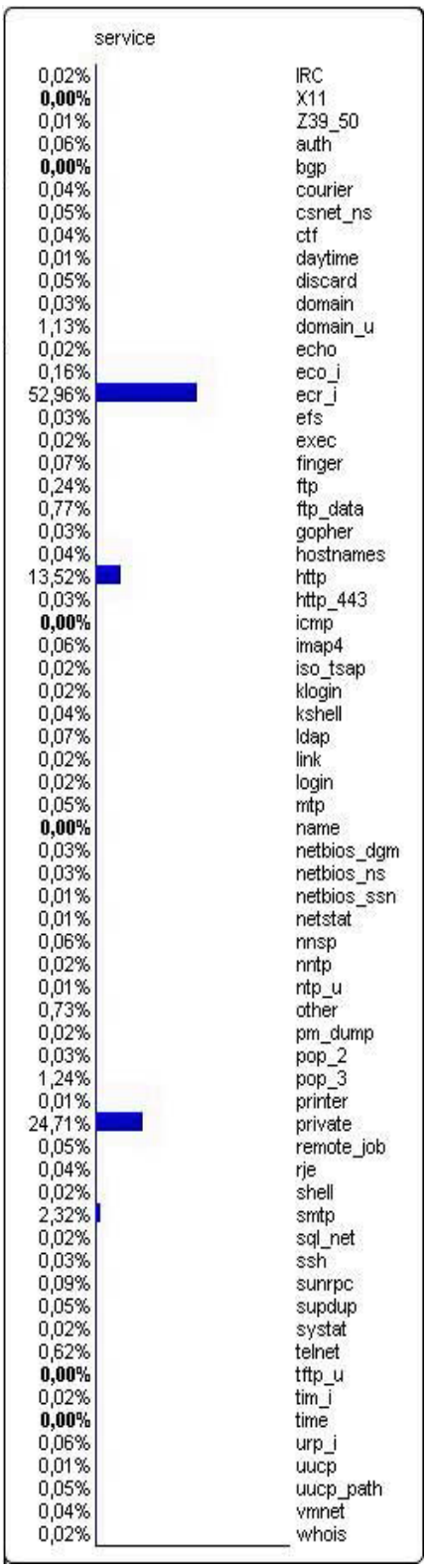
Comme on peut le constater sur la figure ci-dessous, l'activité la plus probable correspond à une attaque smurf (52.90%), un service à ecr_i (ECHO_REPLY, 52.96%) et un protocole de type icmp (53.21%).

Tableau 5.4 : Statistique sur la détection de l'attaque Smurf





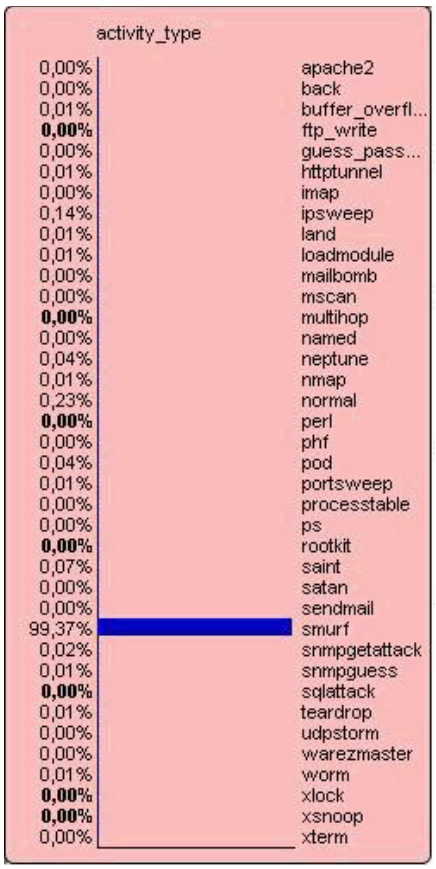


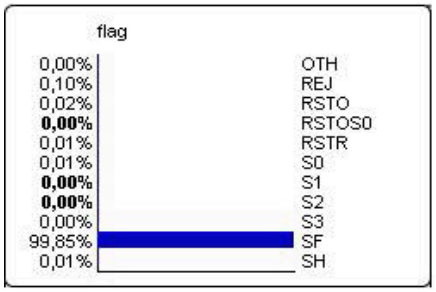


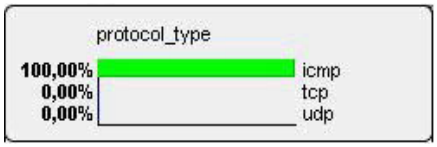
b. Deuxième Observation (Tableau 5.5) :

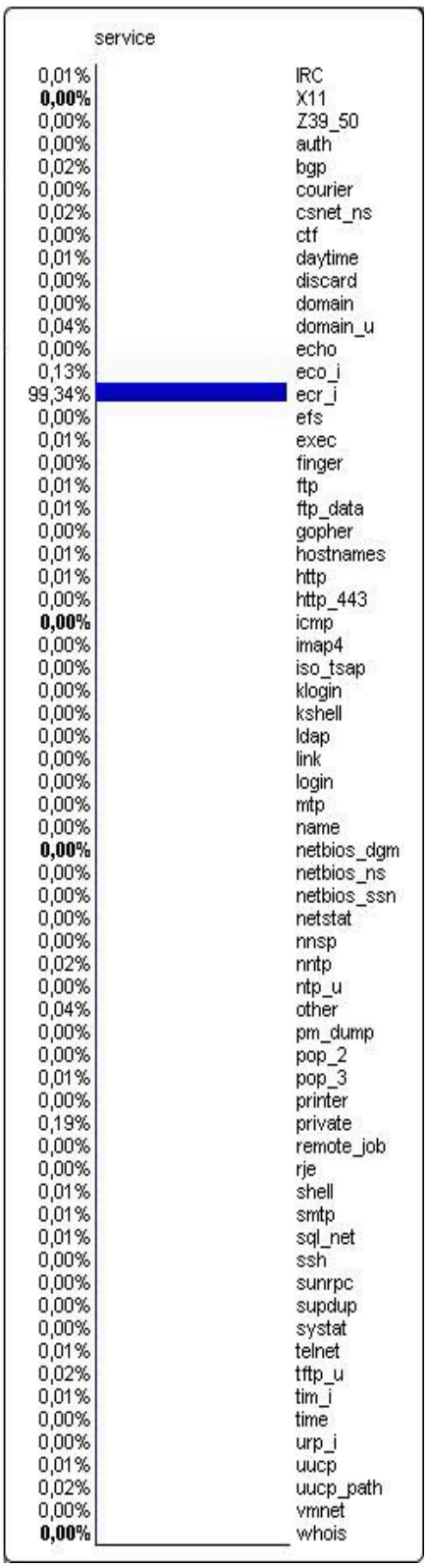
Les moniteurs ci-dessous montrent que dans le cas d'un protocole ICMP, la probabilité d'avoir une attaque Smurf augmente de 52.90% à 99.37%.

Tableau 5.5 : Attaque ICMP augmente le taux de détection de Smurf





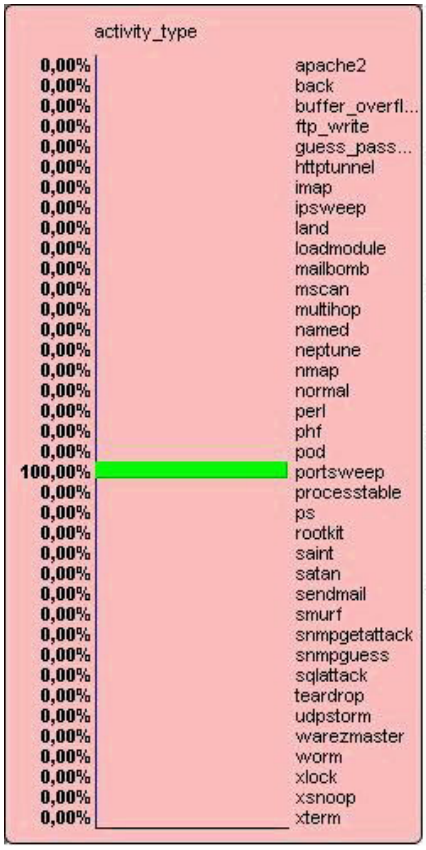


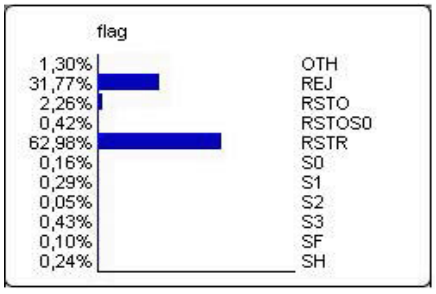


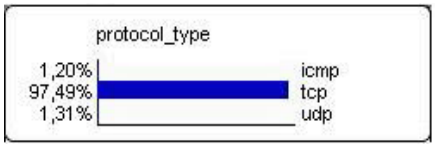
c. *Troisième Observation (Tableau 5.6) :*

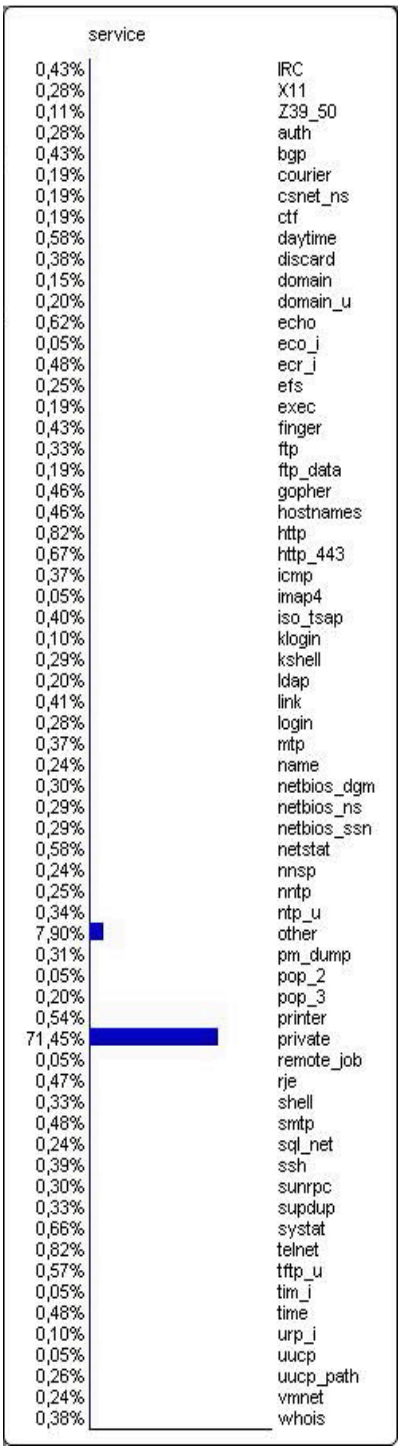
Les moniteurs ci-dessous permettent par exemple de caractériser l'attaque *portsweep*, qui représente un processus de reconnaissance. On constate ainsi que les probabilités du protocole *TCP* et du service privé augmentent de 38.10% à 97.49% et de 24.71% à 71.45% respectivement.

Tableau 5.6 : Attaque Portsweep détectée









5.4.2 Détails des résultats

Cette section présentera quelques captures d'écran lors des tests réels et simulés des détections d'anomalie pendant une durée de 3 mois. Ces observations récupérées sur le serveur de l'université de BLIDA, vont nous permettre de voir de près l'application développée.

a. Tests Réels :

La figure ci-dessous (*Figure 5.6*) montre un ensemble de 4080 attaques réelles classifiées par protocole (TCP, UDP, ICMP), la sonde locale nous a permis de détecter 34 adresses IP source et 483 adresses destination, avec un nombre de ports source de 366 (TCP « 352 », UDP « 14 ») et le nombre de ports destination est de 269 (TCP « 263 », UDP « 7 »).

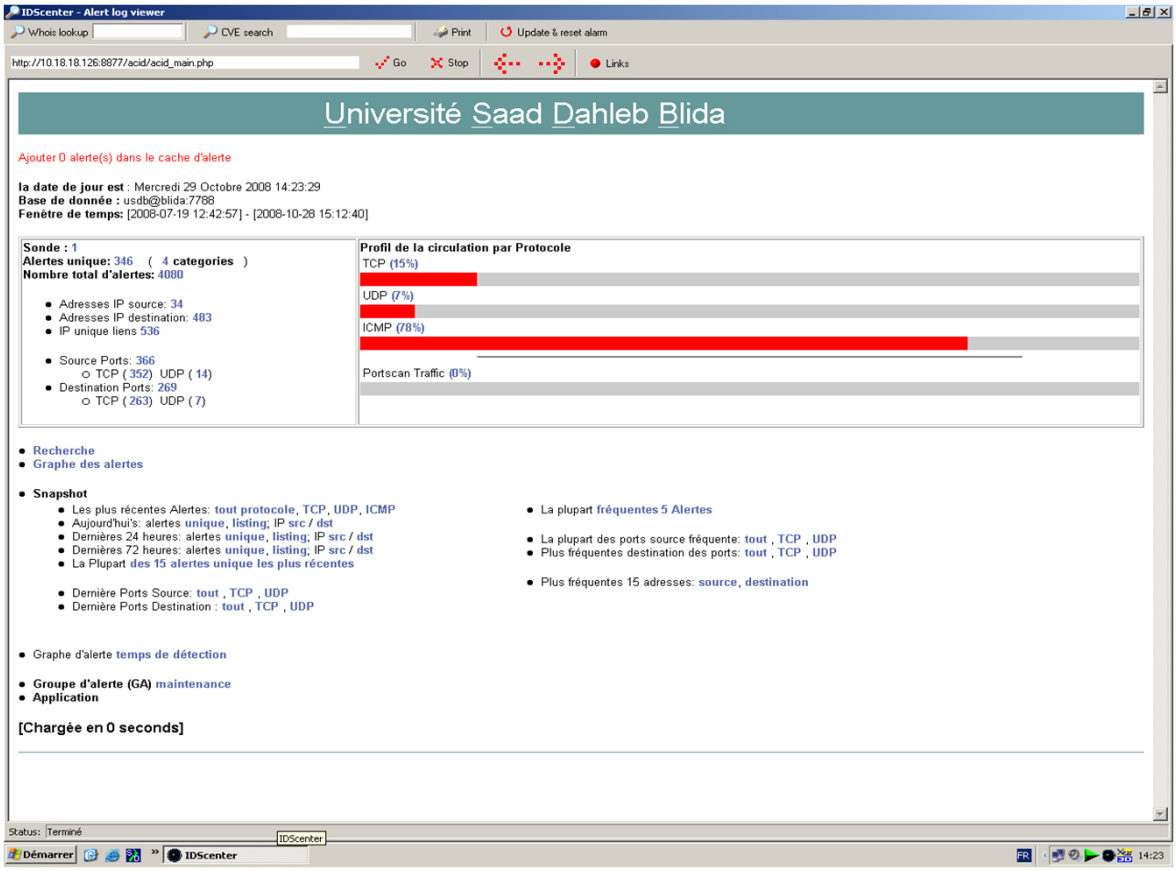


Figure 5.6 : Différentes attaques répertoriées par protocoles à travers le réseau USDB

La liste des attaques observées lors de ces tests est représentée dans la figure 5.7, dévoilant le portscan, qui représente une attaque probing ainsi que d'autre attaques. Parmi ces attaques, on peut voir même les attaques les plus fréquentes. Justement la figure 5.7 montre les cinq attaques les plus fréquentes pendant cette même durée de temps lors de la phase de détection d'attaques.

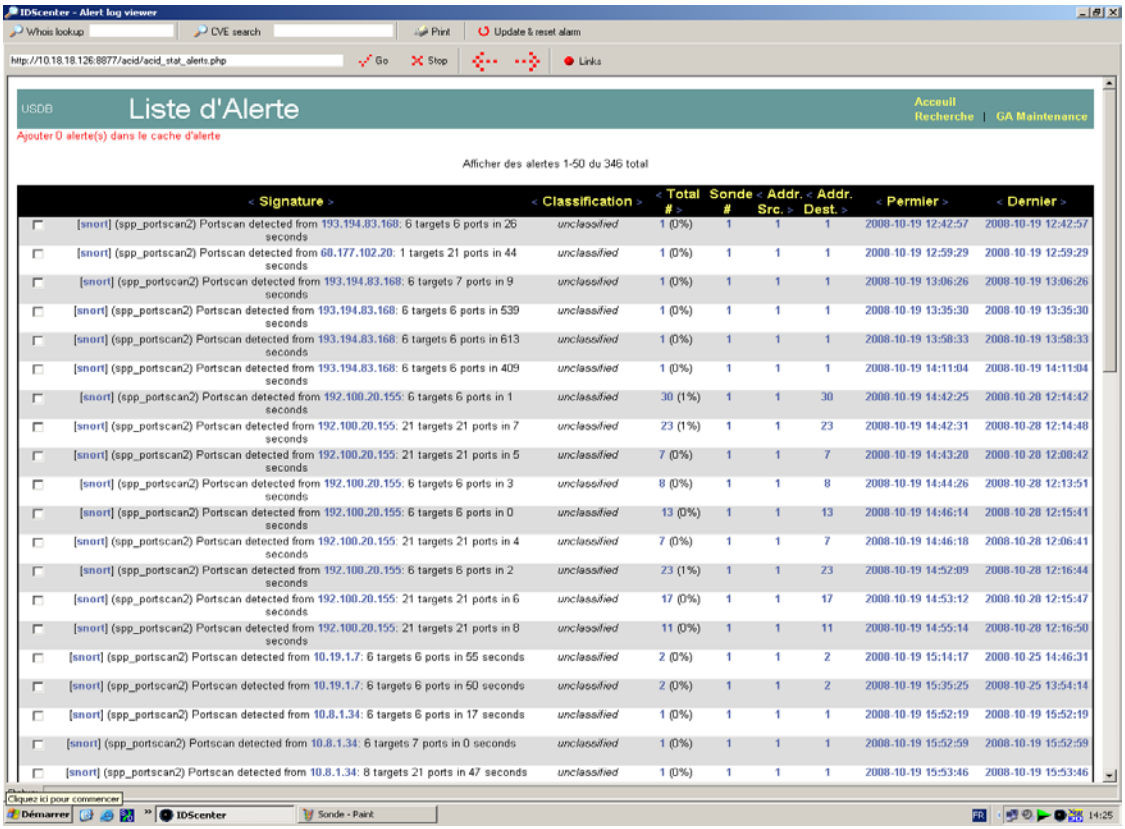


Figure 5.7 : Liste des attaques relevées pendant 3 mois de test

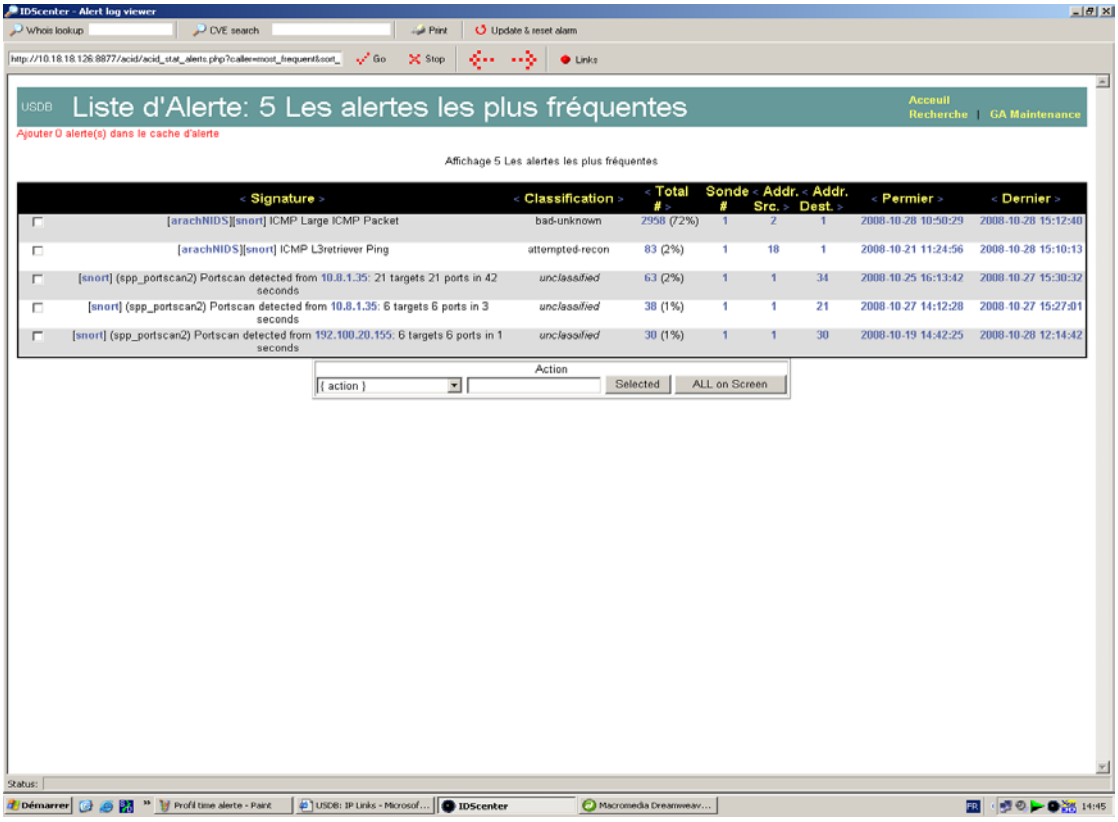


Figure 5.8 : liste des alertes les plus fréquentes

Les différentes attaques internes TCP détectées par notre sonde avec la spécification des adresses sources, la figure 5.9 montre clairement l'affichage des 15 derniers alertes TCP.

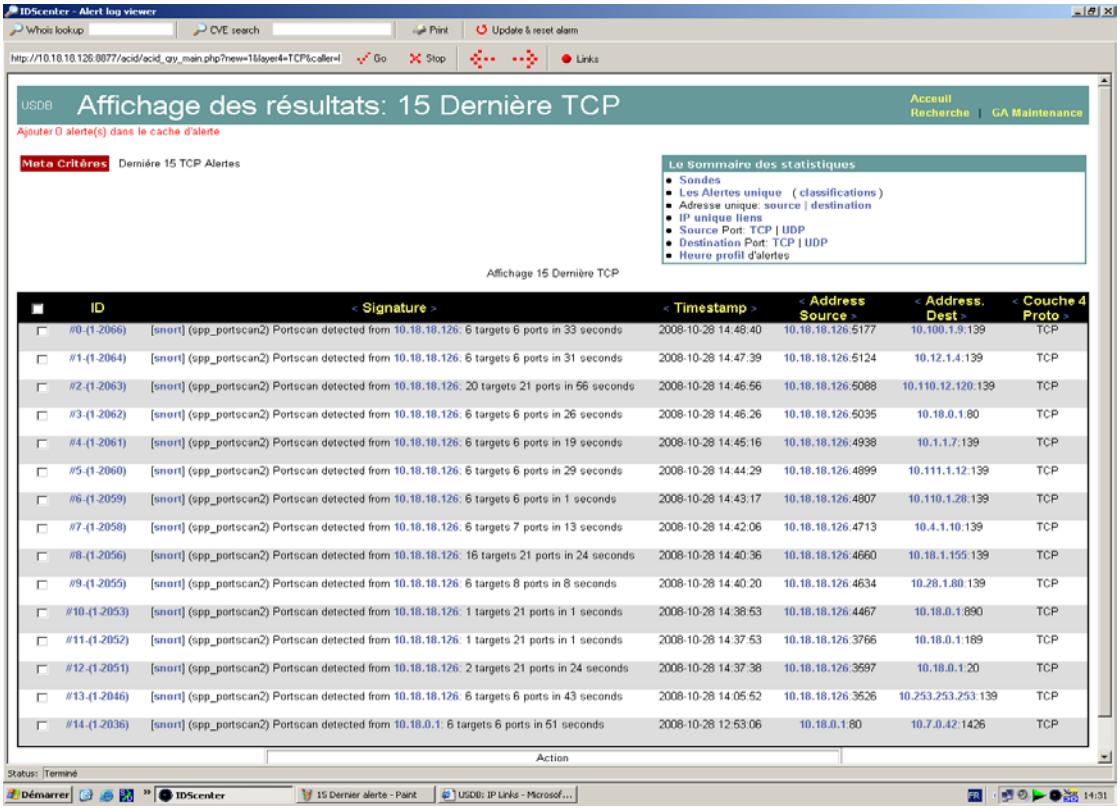


Figure 5.9 : Les 15 dernières alertes détectées

b. Simulation d'attaques :

Dans la partie simulation, nous avons procédé à une multitude de test d'attaques, cette simulation est réalisée grâce à des outils de hacking dans le but de valider les résultats et surtout l'attaque DOS distribué qui représente une fatalité pour le détecteur d'intrusion *Snort*, les modifications apportées à ce soft et l'apprentissage de cette technique nous ont permis de détecter cette faille.

Le logiciel d'attaque DDos (Figure 6.1) présente un bon outil, très puissant et très utilisé par la communauté des hackers, une attaque de dénis de service distribuée générée par celui-ci permet de saturer, bloquer et même redémarrer le système.

Le logiciel DDOS v5.5 nous a permis de tester plusieurs simulations d'attaques, puisque celui-ci regroupe l'ensemble des attaques observées lors de l'apprentissage. Une fois que l'attaque DOS distribuée est réalisée sur notre serveur de test, nous avons pu détecter cette anomalie par notre approche (Figure 6.2).

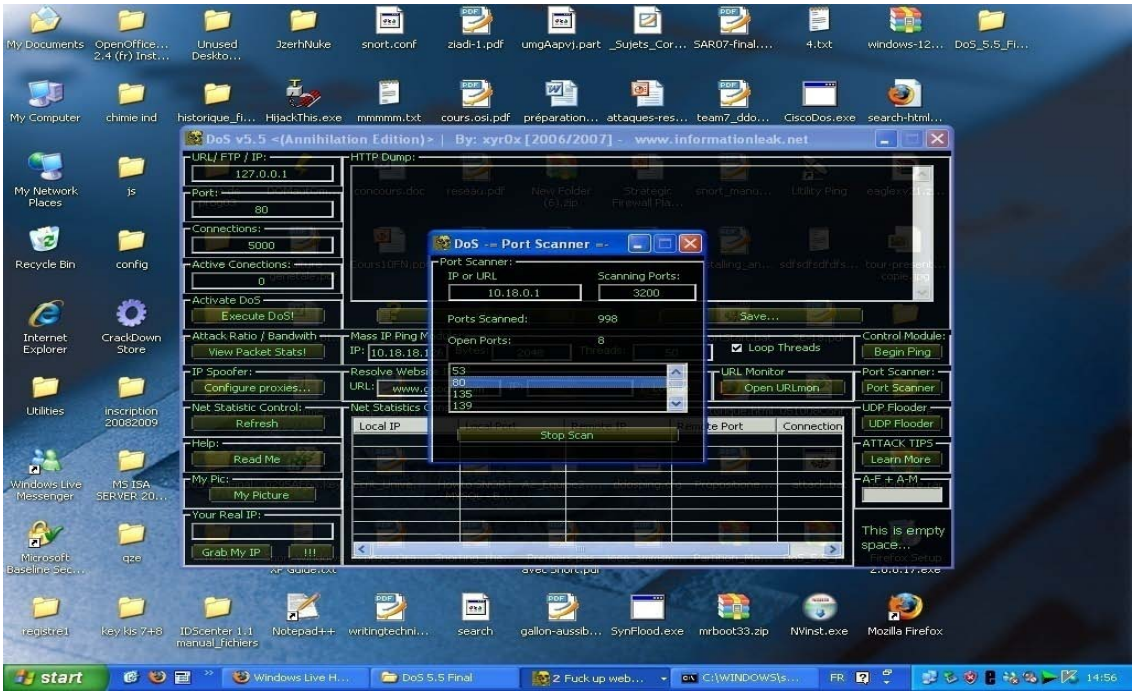


Figure 6.1 : Logiciel de simulation d'attaques DDOS v5.5

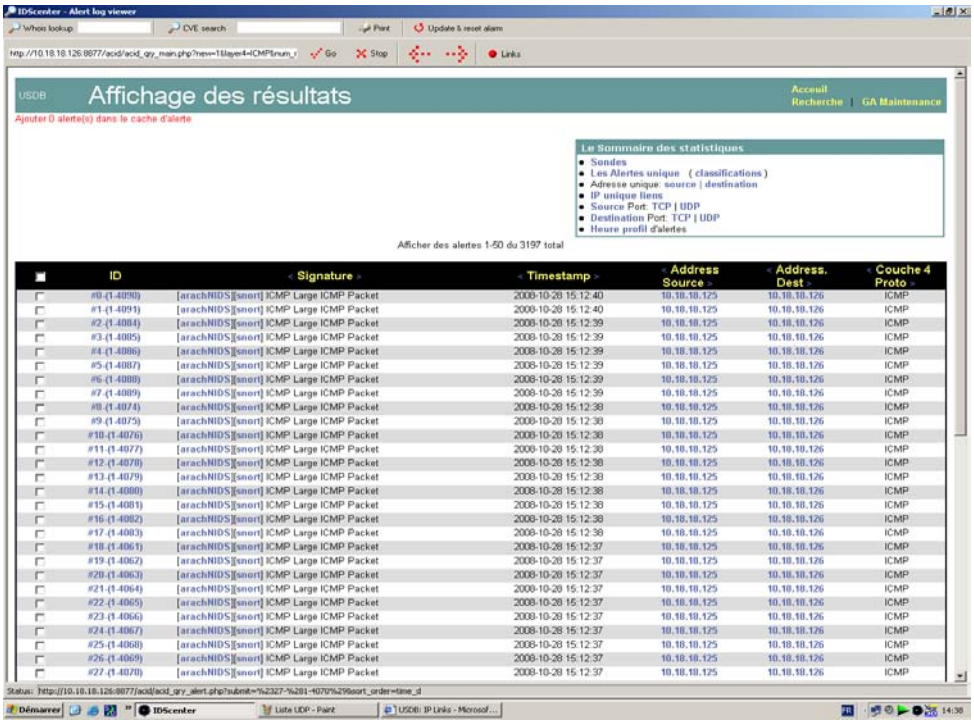


Figure 6.2 Détection de l'attaque DDOS lors de la simulation.

La signature d'attaque mentionnée ici représente un PortScan effectué par le logiciel d'attaque DoS de la machine 10.18.18.125.

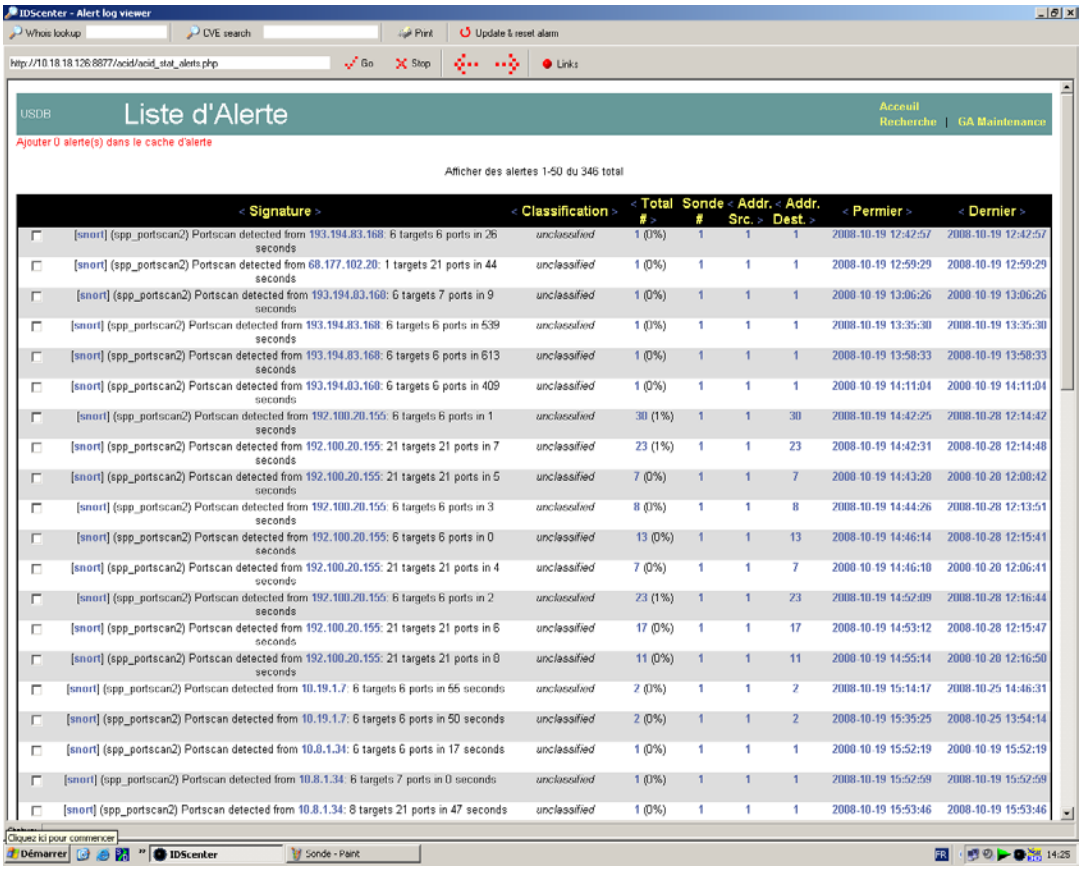


Figure 6 .3 : Liste des alertes engendrées par des attaques réelles

5.4.3 Résultats globaux de la détection d'anomalies

Une fois le modèle réalisé et entrainé, les données de test sont été injectées dans notre moteur de détection d'anomalies (*Snort*). Les premiers résultats relatifs aux taux de détection d'attaques sont présentés dans la Tableau 5.7. Cette figure distingue le taux de détection de chaque scénario en fonction de différentes probabilités.

Les attaques détectées :

Partant du fait que les données d'apprentissage contiennent des attaques qui remontent déjà à 8 ans, on s'attendait peu à trouver ces anciennes attaques dans un trafic récent d'autant plus que les données a priori normales du réseau ont été collectées durant une période de fonctionnement, a priori, normal. Mais les attaques détectées (tableau 5.7) montrent que certaines de ces attaques existent encore. Ci-après certains exemples d'attaques détectées:

- ***Probes***

  La majorité des attaques détectées sont des Probes. En effet, il ne se passe pas une journée sans que des scans se produisent sur le site de BLIDA. Voici des exemples de scans détectés dans les données a priori normales du réseau :

• Scans horizontaux : la majorité de ces scans recherchent des services particuliers sur le réseau. A titre d'exemple, un scan cherchant le service SSH est lancé par la machine dont l'adresse IP est 210.196.250.152 et qui parcourt de manière non séquentielle toutes les machines du réseau 193.194.83. Ce scan s'est produit le 25 octobre 2008 de 19:56:46 à 19:56:49. Un autre exemple de scan détecté concerne la recherche du service FTP et a eu lieu le 27 octobre 2008 entre 16:43:56 et 16:44:33. Ce scan a été lancé depuis l'adresse IP 85.179.219.250 vers les machines du réseau 193.193.83 et ce plusieurs fois et de manière non séquentielle. L'attaque Ipsweep, pourtant très ancienne et très simple, a été observée le 28 octobre 2008 à partir de 08:56:16 et lancée depuis 61.104.160.250. Un autre Ipsweep s'est produit à partir 09:38:43. Il a été lancé depuis 213.201.45.11 et concerne toutes les machines des réseaux 193.193.83 et de manière non séquentielle.

• Scans verticaux (Portsweep) : Ces attaques scannent les ports ouverts sur une destination données. Cette attaque a été détectée plusieurs fois dans les données a priori normales de notre réseau. D'autres scans sont en même temps horizontaux et verticaux dans la mesure où ils recherchent un service donné sur toutes les machines d'un réseau donné puis cherche un autre service sur ce réseau, et ainsi de suite.

- ***DoS***

Le nombre de DoS est relativement insignifiant eu égard aux grands volumes de trafic générés généralement par les attaques DoS. Dans les données a priori normales du réseau, quotidiennement et plusieurs fois par jours, de manière simultanée et synchronisée, la machine dont l'adresse IP est 193.194.83.179 est submergée (figure 6.3) de requêtes ping de certaines machines (4.78.240.100, 216.112.33.26, 63.216.14.130, 66.77.65.79, 38.96.241.130, 209.9.8.81, 205.234.160.30...).

Il s'agit d'un ICMP PING NMAP qui est normalement un scan mais il produit ici l'effet d'un DoS distribué puisque à chaque fois c'est une rafale de requêtes icmp echo envoyées depuis une dizaine de sources et qui s'abattent au même moment sur la même destination 193.194.83.179. Cette dernière ne donne aucune suite à ces requêtes. D'autres attaques DoS détectées semblent dues à la différence des débits réseaux dans les données KDD'99.

- ***R2L et U2R***

Puisque les données a priori normales de Blida ne contiennent pas de trafic relatifs aux services où l'on trouve la majorité des attaques R2L et U2R (dans KDD'99, ces attaques concernent la plupart du temps des sessions telnet, ftp, smtp, login, etc.), on s'attendait à ce que ces attaques ne soient pas détectées dans les données a priori normales du réseau. A titre d'exemple, l'attaque snmpget a été détectée dans les données a priori normales du réseau.

Tableau 5.7 : Affichage des résultats

	Snort					**Snort modifié**				
KDD'99	Normal	DoS	R2L	U2R	Probe	Normal	DoS	R2L	U2R	Probe
DoS (223298)	36,58%	**63,11%**	0,00%	0,00%	0,31%	16,55%	**83,03%**	0,00%	0,00%	0,42%
R2L (5993)	79,99%	1,01%	**18,43%**	0,098%	0,47%	74,79%	0,28%	**24,63%**	0,08%	0,32%
U2R (39)	17,45%	10,23%	41,23%	**28,56%**	2,56%	20,03%	12,56%	29,23%	**34,43%**	3,75%
Probe (2377)	12,1%	2,34%	0,00%	0,00%	**85,56%**	3,67%	3,12%	0,00%	0,00%	**93,21%**

5.5. Conclusion

Les expériences menées dans cette section, nous ont permis d'évaluer les systèmes développés selon le taux de détection des attaques, l'identification et la classification des attaques détectées. Les premières expériences, concernant la détection des attaques élémentaires, ont montré la conformité des résultats de la classification par les réseaux bayésiens avec les données d'entraînement.

L'implémentation ainsi que les tests de notre approche réalisés sur le logiciel Snort open source, nous a permis d'observer de près le phénomène de la détection d'attaques qui correspond à 494019 connexions d'entraînement «données d'apprentissages » ainsi, les résultats obtenus dans ces expériences présentent un taux élevé de la détection des attaques à partir de l'ensemble de test, ce taux est meilleur que celui détecté avec l'analyseur classique de *Snort* open source dans les mêmes conditions expérimentales.

Les résultats des différents classes (DOS, R2L, U2R et Probe) montrent clairement que le taux de détection est amélioré avec cette technique de détection, la probabilité des attributs sur lesquels les attaques reposent devront être bien estimés pour que le nombre de faux positifs puisse être diminué.

CONCLUSION ET PERPECTIVES

Avec ses nombreux avantages, l'Internet a également créé plusieurs manières pour compromettre la sécurité et la stabilité des différents systèmes qui le constituent. Pour cela, des politiques et des outils ont été développés pour fournir des mécanismes de défense de plus en plus efficaces. Les solutions de défense statiques, qui sont analogues aux barrières autour des propriétés, peuvent fournir un niveau de sécurité raisonnable. Ils sont prévus pour empêcher les attaques de se produire. Parmi les exemples de ces solutions le maintient des logiciels tels les systèmes d'exploitation à jour, et le déploiement des pare-feu aux points d'accès.

Aucun système n'est parfait. Les pirates informatiques sont toujours en avance pour trouver les failles de sécurité. Pour cela des mécanismes de défense dynamiques tels que les Systèmes de Détection d'Intrusions (IDS) devraient être combinés avec les solutions préventives. Quand une attaque a lieu, elle marque sa trace dans les données d'audit de l'hôte cible et/ou dans le trafic réseau. Le but des IDS est de surveiller le réseau et les systèmes connectés pour trouver ces traces d'attaques. Ils sont ainsi une partie essentielle pour la formulation d'une politique de défense complète. Deux approches principales sont utilisées pour rechercher la trace des intrusions : l'approche par scénarios et l'approche comportementale. L'approche par scénarios utilise la connaissance a priori des attaques pour rechercher les traces de ces dernières. Ces systèmes sont, par définition, très précis sur les attaques connues incluses dans leur base de données de signatures. D'autre part, puisque les signatures sont associées à un comportement spécifique, il est facile de déterminer le type d'attaque. Cependant, leurs capacités de détection sont limitées aux attaques qui apparaissent dans la base de signatures. Comme à chaque fois des nouvelles attaques sont découvertes, la base de données de signatures exige une mise à jour continue pour inclure les nouvelles signatures d'attaques.

Les systèmes comportementaux adoptent une approche opposée, il faut savoir ce qui est normal, ensuite trouver les déviations par rapport au comportement normal. Ces déviations sont considérées comme intrusions possibles. Ainsi les attaques, y compris les nouvelles, sont détectées tant que le comportement de l'attaque dévie suffisamment du comportement normal. Cependant, si l'attaque est semblable au comportement normal, elle ne peut être détectée. Et comme les utilisateurs changent souvent leurs comportements, le comportement normal devrait être redéfini pour suivre ces changements.

Selon les ressources qu'ils surveillent, les IDS sont divisés en deux catégories. Les systèmes basés hôte surveillent les ressources d'un hôte en recherchant les traces d'une intrusion alors que les systèmes basés réseau essaient de trouver les signatures d'une intrusion dans les données du réseau.

La tendance courante dans la détection d'intrusions est de combiner les informations basées hôte et ceux basées réseau pour développer des systèmes hybrides et ne compter pas donc sur une seule méthode. Les IDS ont trois problèmes communs : la vitesse, l'exactitude et l'adaptabilité. Le problème de la vitesse résulte de la quantité extensive des données que les systèmes doivent surveiller afin de percevoir la situation entière. En mesurant les performances d'un IDS, les taux des faux positifs et celui des faux négatifs sont utilisés pour récapituler les différentes caractéristiques de l'exactitude de la détection. Les faux positifs peuvent être définis comme des alarmes qui sont déclenchées à partir d'activités légitimes. Les faux négatifs sont les attaques, qui ne sont pas détectées par le système. Un IDS est plus précis s'il détecte plus d'attaques et donne peu de fausses alarmes.

Dans le cas des systèmes basés signatures, les experts de sécurité doivent examiner les nouvelles attaques pour ajouter les signatures de détection correspondantes. Dans le cas des systèmes utilisant l'approche comportementale, les experts humains sont nécessaires pour définir au moins le comportement normal. Ceci nous mène au problème d'adaptabilité. Les aptitudes des IDS courants à s'adapter sont très li mitées. Ceci les rend peu efficaces pour la détection des attaques nouvelles ou inconnues ou bien pour s'adapter aux environnements évolutifs, en d'autres termes l'intervention humaine est toujours exigée.

Les méthodes d'apprentissage automatique fournissent une solution potentielle pour les problèmes d'adaptation et d'exactitude dans le domaine de la détection d'intrusions. Dans ce domaine, l'apprentissage signifie l'extraction des modèles du comportement normal ou d'attaques. Selon la méthode d'apprentissage utilisée les algorithmes d'apprentissage sont supervisés ou non. Dans l'apprentissage supervisé, l'algorithme d'apprentissage est appliqué sur un ensemble d'instances classifiées (ou marquées) et utilisé pour traiter les nouvelles instances non classifiées. D'un autre coté, l'apprentissage non supervisé implique la recherche des associations entre les attributs sans se servir des classes ou des labels.

La classification est une tâche de l'apprentissage automatique qui a une phase

d’entraînement où il faut apprendre mathématiquement les modèles à partir de l'ensemble de données introduit. Cet ensemble de données s'appelle également l’ensemble d’entraînement, qui devrait contenir des instances suffisantes et représentatives des modèles recherchés sachant qu’une instance se compose d’un ensemble d’attributs. Les modèles appris sont utilisés pour faire des prédictions à partir d’un nouvel ensemble d’instances de données. Beaucoup d'approches de classification tentent la construction explicite d’une fonction à partir de l'ensemble commun des valeurs d’attributs pour obtenir les labels des instances.

Les réseaux Bayésiens adoptent une approche un peu différente par rapport à ces méthodes puisqu'ils fournissent des informations utiles sur la structure du problème lui même. Les réseaux Bayésiens permettent d’extraire la connaissance contenue dans l’ensemble de données d’entraînement, cette connaissance n’est pas accessible directement car elle est noyée dans des chiffres, pour la rendre interprétable, il faut la transformer en graphe de causalité, mettant en évidence les relations de dépendances/indépendances entre les variables observées, pour qu’elles peuvent être lues et interprétées par la suite, beaucoup plus facilement que les données initiales.

Nous avons présenté dans ce projet une nouvelle approche de détection d’intrusions basée sur la classification Bayésienne. Nous avons montré comment un réseau Bayésien détecte de manière probabiliste une attaque d'un réseau de communication, et permet ainsi la généralisation de systèmes de détection d'intrusion réseaux (SDIR).Les différentes formes que peut prendre cette approche ont été expliquées, puis une approche originale, paramétrée par la politique de sécurité, a été détaillée. Une formalisation de cette approche, a été implémentée et réalisée sur Snort open source. Elle permet de prouver la fiabilité et la pertinence du modèle de détection. En d’autres termes, une intrusion implique la levée d’une alerte; une alerte implique qu’une intrusion a effectivement eu lieu.

La plupart des classifications ou taxonomies des Système de Détection d’Intrusion utilise deux critères discriminant pour différentier ces systèmes : la nature des données traitées (NIDS, HIDS,méta-IDS)et le mode de détection (par signature ou par détection d’anomalies). Nous avons introduit un nouveau critère permettant de différentier ces systèmes à l’aide de la nature d’apprentissage de ces outils de détection. Ce nouveau critère de classification des Systèmes de Détection d’Intrusion qui permet de spécifier comment ces systèmes génèrent leurs profils normaux ou leurs signatures d’attaques constitue notre première contribution. Les systèmes de détection actuels peuvent combiner

à la fois la détection d'attaques connues et la détection de nouvelles attaques. La construction automatique de signatures d'attaques, par exemple, va permettre cette détection duale. Avec l'utilisation des critères traditionnels de mode de détection ou de nature des données traitées, cette dualité n'est pas identifiable. L'ajout de la nature de l'apprentissage comme critère va permettre cette distinction. Ainsi ces systèmes de détection d'intrusion seront considérés comme des systèmes à base de signatures et d'apprentissage automatique.

Notre seconde contribution nous a permis de modéliser la démarche générique d'un attaquant. L'utilisation de l'ontologie et des démarches des utilisateurs légitimes a permis de décrire la stratégie d'un attaquant en modélisant ces différents objectifs et en différentiant ces actions légitimes de ces actions malicieuses. Cette description de la démarche de l'attaquant permet de différencier l'ensemble des étapes d'un attaquant, de la reconnaissance d'une cible à l'utilisation d'une machine compromise. Nous nous différencions des descriptions des démarches d'attaquant existantes en différenciant les étapes d'attaques ou de reconnaissance des actions légitimes détournées.

L'utilisation conjointe de la normalisation des évènements collectés et de la sélection des indicateurs à observer nous a garanti un passage comportemental appliqué à la détection d'intrusion. Notre approche se différencie des approches actuelles par la nature des données analysées. Basé sur l'ontologie [53] décrivant tout évènement survenu sur le réseau. Cette modélisation est réalisée à l'aide des réseaux Bayésiens permettant de concentrer les différentes informations et de proposer un modèle humainement lisible. La plupart des solutions de détection de scénarios d'attaques utilisent des modèles à bases d'automates d'états, de méthodes de DataMining, des réseaux de Pétri ou encore des chaines de Markov. En utilisant les réseaux Bayésiens comme technique de modélisation des comportements déterminés (analyse de protocoles), nous pouvons modéliser les probabilités des séquences d'actions dans un graphe réduit "concentrant" l'ensemble des informations. L'utilisation des réseaux Bayésiens dans la détection d'anomalies à l'échelle de notre approche. Cette étude de comportements constitue notre troisième contribution.

Cette approche probabiliste que nous avons proposée n'est qu'une tentative de recherche, nous avons démontré que la capacité d'apprentissage et déployant cette approche par réseaux Bayésiens sont des outils prometteurs et utiles pour déterminer des évènements suspicieux et les signaux annonciateurs de menaces.

Perspectives

Comme nous l'avons introduit, la détection d'intrusion est devenue un enjeu majeur pour la sécurité des systèmes d'informations. L'utilisation conjointe de systèmes de détection d'intrusion par signature et de systèmes de détection d'anomalies est primordiale pour, d'une part détecter rapidement des scénarios d'attaques connues et des utilisations anormales du système (traces d'intrusion) et d'autre part réduire les faux positifs des deux méthodes de détection. En recoupant les informations reçues par chaque méthode de détection, il sera possible de valider ou d'infirmer la présence d'attaques ou de suspicions d'intrusion. Le modèle présenté est parfaitement adapté aux organismes fortement structurés.

Les nouvelles architectures (pervasives, ahhoc) portent des contraintes supplémentaires telles que la non connaissance des voisins et l'impossibilité de mettre en place une surveillance centrale. Notre modèle à base de réseau Bayésien pourrait être réutilisé dans ces architectures de coopération en modifiant les informations traitées. Nous pouvons envisager que chaque entité d'un réseau possède deux modèles de comportement. Un modèle décrivant les activités normal d'échanges avec d'autres entités et un modèle global partagé avec les voisins décrivant les comportements normaux des échanges dans le réseau. Par la mise en place d'une telle structure, chaque entité aurait la possibilité d'identifier si les échanges avec une entité voisine comme correspondent à une activité normale vis-à-vis de sa propre connaissance mais également vis-à-vis de la connaissance globale partagée dans le réseau.

APPENDICES

A. Liste des symboles et des abréviations

A : Adjoint matrix, complex conjugate matrix

ACL : liste de contrôle d'accès « Access Control List »

AIS : Adaptive Importance Sampling

API : Application protocol interface

BNJ : Bayesian Network tools in Java

BNT : Bayes Net boite à outils (Toolbox)

CIDF : Common Intrusion Detection Framework

C : component index in a mixture(algorithmic)

CEM : Component-wise EM algorithm

CPD : La distribution des probabilités conditionnelles

CPT : Table conditionnelle de probabilités

DAG : Graphe acyclique dirigé

DCE : Distributed Computer Environment

DDOS : Denis de Service Distribué (Distributed Denis of Service)

DF : Data Flag

DOS : Denis de Service

EM : Expectation Maximization

EVDO : Réseau sans fil a grand vitesse (Evolution-Data Optimized)

FTP : Protocole de Transfert de Fichiers (File Transfer Protocol)

GMM : Modèle de mélange gaussien (Gaussian Mixture Model)

HIDS : Détection d'intrusion en niveau hôte (Host Intrusion Detection System)

HMM : Hidden Markov Models

HTTP : HyperText Transfer Protocol

ICMP : Internet Control Message Protocol, Les paquets de type « echo request » et « echo reply »

IDS : Système de détection d'intrusion

IDWG : Intrusion Detection Working Group

IETF : Interner Engineering Task force

IIS : Système de détection d'intrusion

IRC : Internet Relay Chat

K : (algorithmic) index of a class

K : Number of classes

ML : Maximum de vraisemblance (Maximum Likelihood)

MWST : Maximal weight spaning tree

n : index of a sample(algorithmic)

N : Nombre d'échantillons (number of samples)

NIDS : Détection d'intrusion en niveau Réseau (Network Intrusion Detection System)

p(=) : Fonction de densité de probabilité (probability density function)

P(=) : Probability (mass) function

PDF : Fonction de densité de probabilité (Probability Density Function)

PING : Packet InterNet Groper, permettant de tester l'état actif d'une station d'un réseau

PNL : Probabilistic Network Library

SI : Système d'information

SRA : Algorithme de Restriction Successive (Sucesive Restrictions Algorithm)

TCP : Protocole de contrôle de transmissions (Transmission Control Protocol)

Telnet : TErminal NETwork ou TELecommunication NETwork

TS :TimeStamp

TTL : Temps de vie (Time-To-Live)

V : Number of parameters of a mixture distribution component

VPN : Réseau privé virtuel (Virtual Private Network)

Wimax : Connexions à haut-débit par voie hertzienne

WS : Window Scale

X : Ensemble de vecteurs

x : Un échantillon, vecteur de données

B. Lexique

Ce lexique précise le sens que nous donnons à de nombreux termes utilisés dans ce mémoire. Il est largement inspiré des définitions proposées par l'IDWG (Intrusion Detection Working Group) de l'IETF.

Administrateur : personne chargée de mettre en place la politique de sécurité, et par conséquent, de déployer et configurer les IDS.

Alerte : message formaté qui décrit un évènement relatif à une action qui compromet la sécurité d'un système ou d'un réseau. Les alertes sont produites par un analyseur. Nous faisons l'hypothèse que le format utilisé est l'IDMEF.

Analyseur : outil logiciel qui met en ouvre l'approche choisie pour la détection (comportementale ou par scénarios). Il génère des alertes lorsqu'il détecte une intrusion à partir des évènements remontés par les capteurs ou à partir d'alertes générées par d'autres analyseurs.

Approche comportementale : ensemble des techniques utilisées par les IDS qui basent leur processus de détection sur l'hypothèse que toute déviation significative du comportement observé d'une entité par rapport à son modèle de comportement normal constitue une intrusion potentielle.

Approche par scénarios : ensemble des techniques utilisées par les IDS qui détectent les intrusions en recherchant dans les activités courantes celles qui sont caractéristiques de scénarios d'attaques connus (comparaison avec une base de signatures d'attaques). Aussi appelée approche par signatures.

Attaque : synonyme d'intrusion.

Capteur : logiciel qui génère les évènements en filtrant et formatant les données brutes intéressantes provenant dune unique source d'information (paquets du réseau, logs du système ou logs applicatifs).

Défacement : Un défacement, défaçage ou défiguration (defacing en anglais) est un anglicisme désignant la modification non sollicitée de la présentation d'un site Web, suite au piratage de ce site. Il s'agit donc d'une forme de détournement de site Web par un pirate.

Détection d'intrusions : processus logiciel de recherche des intrusions qui s'appuie sur la surveillance des activités des entités dans les systèmes et les réseaux. Nous réservons le

terme de détection d'intrusions à l'analyse logicielle et automatique par opposition à l'analyse manuelle de logs effectuée par un opérateur humain.

Evénement : message formaté renvoyé par un capteur. C'est l'unité élémentaire utilisée pour représenter une étape d'un scénario d'attaque connu.

Exploit : terme utilisé pour désigner un programme d'attaque.

Faux positif : alerte émise en présence dune action légitime rapportée à tort comme étant une intrusion par un système de détection d'intrusions (fausse alerte).

Faux négatif : absence d'alerte en présence dune action qui constitue bien une intrusion mais qui na pas été détectée comme telle par un système de détection d'intrusions.

IDS : acronyme de "Intrusion Detection System". Voir système de détection d'intrusions.

Intrusion : action (ou tentative d'action) qui a pour conséquence de compromettre l'intégrité, la confidentialité ou la disponibilité d'une ressource (violation de la politique de sécurité).

Manager : composant d'un IDS permettant à l'opérateur de gérer les autres composants. Ses fonctions comportent généralement la configuration des capteurs et analyseurs, la notification des alertes à l'opérateur et éventuellement la réaction.

Opérateur : personne chargée de l'utilisation du manager associé à l'IDS. Elle propose ou décide de la réaction à apporter en cas d'alerte. C'est parfois la même personne que l'administrateur.

Politique de sécurité : spécification des règles à respecter dans le 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.

Portsweep : il s'agit de scanner plusieurs hôtes, à la recherche d'un port d'écoute spécifique

Réaction : mesures passives ou actives qui peuvent être prises en réponse à la détection d'une attaque, pour la stopper ou pour corriger ses effets.

Scénario : suite des étapes dune intrusion.

Signature : règle utilisée par certains analyseurs pour identifier parmi les activités surveillées celles qui sont caractéristiques dune intrusion.

Sonde : regroupement (logique ou fonctionnel) d'un capteur et d'un analyseur.

Source de données : dispositif générant de l'information sur les activités des entités du système d'information (ex : système d'audit d'un système d'exploitation, sniffer réseau, ...).

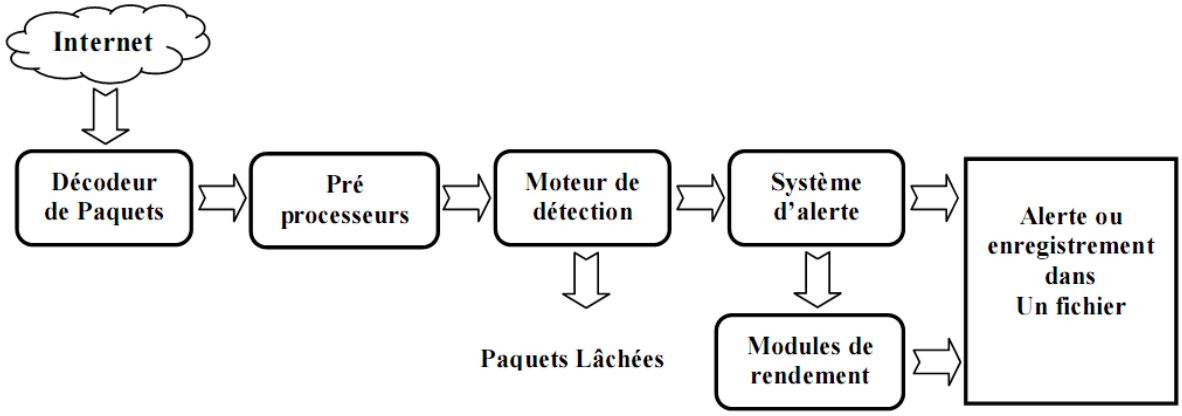
Système de détection d'intrusions : ensemble complet composé de capteur(s), d'analyseur(s) et de manager(s).

C. Installation de Snort

C.1 Snort

Snort est le *N-IDS* le plus populaire de nos jours, son principe de fonctionnement repose sur l'approche par scénarios. *Snort* est un produit Open Source *(http://www.snort.org)* ce qui lui permet d'avoir une communauté très importante d'utilisateurs et de contributeurs.

Snort est logiquement divisé à des composants multiples. Ces composants fonctionnent ensemble pour détecter des attaques particulières et pour générer un rendu dans un format requis par le système de détection. La *figure 1.6* montre comment ces composants sont arrangés.



Composants de l'IDS Snort

Dans ce qui va suivre on va montrer le rôle de chaque module :

- *Décodeur De Paquets*: Il prend les paquets des différentes interfaces réseau et les prépare pour être prétraités ou pour être envoyés au moteur de détection. Les interfaces peuvent être : *Ethernet*, *SLIP*, *PPP*, etc.
- *Pré-processeurs* : Sont des composants de connexion qui peuvent être employés avec *Snort* pour arranger les paquets de données avant de les envoyer vers le moteur de détection, pour découvrir si ces paquets sont utilisés par un intrus. Quelques préprocesseurs effectuent également la détection en cherchant les anomalies dans les en-têtes des paquets et en générant des alertes si c'est le cas. Les

préprocesseurs sont utilisés pour se protéger aussi contre les différentes techniques employées par les intrus pour duper les *IDS,* puisqu'ils permettent de défragmenter les paquets, décoder les *URI HTTP*, rassembler les flux *TCP* et ainsi de suite. Ces fonctions sont une partie très importante dans un bon système de détection d'intrusions.

- *Moteur de détection*: C'est la partie la plus importante de *Snort*. Son rôle est de détecter l'existence de n'importe quelle activité d'intrusion dans un paquet. Le moteur de détection utilise les règles *Snort* à cette fin. Les règles sont lues à partir de structures ou de chaînes de données internes où elles sont comparées avec tous les paquets. Si un paquet ressemble à n'importe quelle règle, une action appropriée est pris; autrement le paquet est lâché. Ces actions consistent à stocker le paquet sous le format approprié ou produire des alertes. Le moteur de détection est la partie critique en temps de *Snort*, dépendant de la puissance de la machine, la vitesse des bus internes est du nombre des règles définies, il peut prendre des durées différentes pour répondre aux différents paquets. Si le trafic réseau est important, *Snort* peux laisser tomber quelques paquets et ne peux pas obtenir de véritable réponse en temps réel.
- *Système d'alerte et d'enregistrement*: Dépendant de ce que le moteur de détection trouve à l'intérieur d'un paquet, le paquet peut être employé pour enregistrer l'activité ou pour produire une alerte. Les enregistrements sont maintenus dans les fichiers textes simples, des fichiers de style *tcpdump* ou sous une autre forme.
- *Les modules de rendement*: Ces modules peuvent faire différentes opérations selon la façon dont l'utilisateur veut traiter le rendement produit par le système d'alerte et d'enregistrement de *Snort*. Selon la configuration établie, les modules de rendement peuvent :

    - Envoyer des trappes *SNMP* ou des messages au *syslog*;
    - Stocker des comptes rendus dans une base de données comme *MySQL*, Oracle ou bien simplement les stockés dans des fichiers avec une extension donnée.
    - Produire des fichiers Extensible Markup Langage *(XML)*.
    - Modifier la configuration des routeurs et des pare-feux.

Envoyer des messages bloquants du serveur *(SMB)* vers les machines fonctionnant sous Microsoft Windows.

C.*2 Introduction*

Ce document va tenter d'expliquer les différentes étapes pour mettre en place le détecteur d'intrusions SNORT à partir des sources. Un détecteur d'intrusions s'appelle aussi "IDS" pour Intrusion Detection System. SNORT est un système de détection d'intrusions réseau en OpenSource, capable d'effectuer l'analyse du trafic en temps réel. On l'utilise en général pour détecter une variété d'attaques et de scans tels que des débordements de tampons, des scans de ports furtifs, des attaques CGI, des scans SMB, des tentatives d'identification d'OS, et bien plus.

Avant de commencer l'installation de SNORT, vous devez avoir installé :

PACKAGES REMARQUES

- ☒ MySQL La base de données MySQL
- ☒ MySQL−client La partie cliente de mysql (connexion BD)
- ☒ php−mysql le module php de mysql
- ☒ Apache Le serveur web Apache
- ☒ mod_php Le module php pour Apache
- ☒ libpcap/libpcap0−devel Librairie utilisée par SNORT pour capturer les paquets (rpm téléchargeable sur rpmfind.net)
- ☒ gcc indispensable pour compiler les sources de SNORT

Si vous n'avez pas encore installé le trio Apache/PHP/MySQL.

Les étapes pour l'installation de SNORT sont les suivantes :

- ☐Installation de l'outil SNORT
- ☐Installation des règles SNORT
- ☐Liaison Mysql et SNORT
- ☐Mise en place de ACID (Interface php pour visualiser les logs SNORT)

C.3 Installation de SNORT

Téléchargez la dernière release de SNORT à l'adresse suivante : http://www.SNORT.org/dl. La compilation de ce programme reste traditionnelle :

COMMANDES	**REMARQUES**
cd /usr/local/snort	...

tar −xvzf SNORT−1.9.*.tar.gz	Décompacte l'application
./configure −−with−mysql=/usr/lib/mysql	Retirez l'argument −−with−mysql si vous ne souhaitez pas rediriger les logs SNORT vers une base de données mysql *
make	Compilation
make install	Installation

Pour l'argument −−with−mysql, vous pouvez l'adapter si vous utilisez une base de données autre que MySQL :

- --with-odbc=$PATH_ODBC : pour une base de données Microsoft SQL server
- --with-postgresql=$PATH_POSTGRE : pour une base PostegreSQL
- --with-oracle=$ORACLE_HOME : pour une base de données Oracle.

C.4 Installation des règles SNORT

Maintenant, il faut télécharger les règles de SNORT. En effet, SNORT utilise des règles pour détecter les intrusions. Il existe aujourd'hui environ 1200 règles différentes. Ces règles se caractérisent par un ensemble de fichiers (ftp.rules, p2p.rules,telnet.rules etc...). Vous devez téléchargez les sources de ces règles à l'adresse suivante :

http://www.SNORT.org/dl/signatures

Créez le répertoire de configuration SNORT, et installez−y les règles :

COMMANDES	**REMARQUES**
mkdir /etc/snort	Création du répertoire contenant la configuration SNORT
cp /usr/local/snort*/etc/snort.conf /etc/snort	Copie du fichier de config snort dans /etc/snort
cp snortrules.tar.gz /etc/snort	Mise en place des règles dans le répertoire de configuration SNORT
cd /etc/snort	On se place dans le répertoire de configuration SNORT
tar −xvzf snortrules.tar.gz	Décompactage des règles

Les règles SNORT sont alors placées dans le répertoire /etc/snort/rules.

1. ***Installation de l'IDS SNORT***

Maintenant, Il faut éditer le fichier de configuration snort (/etc/snort/snort.conf) et spécifier le réseau sur lequel l'IDS travaille. Il faut pour cela modifier la variable HOME_NET :

var HOME_NET [10.1.1.0/24] # SNORT travaille sur le réseau 10.1.1.0

var HOME_NET (10.1.1.0/24,192.168.1.0/24] # Si votre carte réseau possède 2 alias

Dans le fichier de configuration de SNORT (/etc/snort/snort.conf), vous avez toute une série de include. Il s'agit des règles utilisées par

SNORT pour détecter d'éventuelles intrusions. Il y a des règles de telnet, ICMP, FTP, ... Bref, commentez celles que vous ne voulez pas et décommentez celles qui vous parait utile. Conseil : Décommentez les règles ICMP, car elles ne cessent pas de vous remonter des alarmes très souvent inutiles.

Pour des explications plus détaillées concernant les règles SNORT, allez voir ici.

2. Lancement de SNORT

Deux possibilités s'offrent à nous. Soit vous lancez SNORT tout seul, et dans ce cas, il générera ces logs dans un fichier plat. Soit vous décidez de l'interfacer avec une base de données. Suivant le cas, SNORT ne se lancera pas de la même façon.

Sans Mysql :

/usr/local/snort*/src/snort −c /etc/snort/snort.conf −i eth0 −D

Avec Mysql :

/usr/local/snort*/src/snort −c /etc/snort/snort.conf

Remarque : Si vous souhaitez interfacer SNORT avec une base de données, ne lancez pas SNORT avec l'argument −L qui spécifie l'emplacement des logs.

3. Lier les logs SNORT avec MySQL

Maintenant, nous allons éditer le fichier de configuration de SNORT afin de lui indiquer qu'il faut rediriger les logs dans une base de données (ici

MySQL). Avec vos yeux de lynx, retrouvez la ligne suivante dans le fichier de configuration SNORT /etc/snort/snort.conf :

#output database:log,mysql,user=root password=test dbname=SNORT host=localhost

Décommentez et modifiez cette ligne par :

output database:log,mysql,user=user_snort password=snort_pwd dbname=snort host=localhost

Ici, l'utilisateur MySQL accédant à la base de données s'appelle "user_snort", son password associé est "snort_pwd", le nom de la base MySQL utilisée par snort est "snort" et la machine qui fait tourner la base Mysql est la même que celle où SNORT tourne.

4. Création de la base de données SNORT

Au préalable, assurez−vous d'avoir installé :

PACKAGES **REMARQUES**

MySQL−client−* partie cliente de MySQL

MySQL−devel−*

Astuce : La commande "rpm−qa | grep client" vous permet de vérifier que votre station Linux possède bien ces packages installés.

Suivez alors les instructions suivantes :

COMMANDES	**REMARQUES**
cd /usr/local/snort*/contrib	On se place à l'endroit du fichier contenant les tables SQL de SNORT
mysql −u root −p	Connexion à la base de données en tant qu'administrateur (au passage, si ce n'est pas encore fait, définissez un password pour l'administrateur de la base par la commande 'set password for root@localhost=PASSWORD('totomdp');
create database SNORT;	Création de la base de données SNORT
use mysql;	On se place ici pour créer l'utilisateur MySQL qui gérera la base de données snort

insert into user values('localhost', 'user_snort',
password('snort_pwd'), 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', '', '',
'', '', 'Y', 'Y', 'Y');

Création utilisateur MySQL "user_snort". Attention le nombre de 'Y' dépend de votre version de MySQL. (faites un select * from user; pour voir combien il faut en mettre)

grant ALL PRIVILEGES ON SNORT.* TO user_snort@localhost IDENTIFIED BY 'snort_pwd' WITH GRANT OPTION;

Attribution des droits de la base "snort" à l'utilisateur "user_snort"

flush privileges;	Recharge les tables de droits pour prendre en compte les nouvelles modifications
use snort;	on se place dans la base où l'on veut créer les tables pour SNORT

Source create_mysql Création des tables pour SNORT

Vérifiez que les tables sont bien créées. Allez voir dans /var/lib/mysql/snort et vous y verrez tout un tas de fichiers correspondant au nom des tables de la base de données SNORT (il doit y avoir 3 fichiers par tables).

Lancez SNORT. Désormais, SNORT envoie les informations dans la base de données (astuce : installez PhpMyAdmin, et vérifiez la taille de la base de données SNORT. Si tout fonctionne, vous la voyez augmenter si bien évidemment il y a du trafic !).

5. Installation/Configuration ACID

ACID est une interface PHP qui permet de visualiser les remontées d'alarmes générées par SNORT. Cette partie sous−entend que vous avez une base de données qui récupère les informations envoyées par SNORT. Avant de suivre l'installation de cette application, assurez−vous d'avoir téléchargé :

Adodb : Contient des scripts PHP génériques de gestion de bases de données. L'installer dans la racine d'apache (/var/www/html/adodb par exemple)

PHPlot : librairie de scripts PHP utilisée par ACID pour présenter graphiquement certaines données statistiques (optionnel)

Le téléchargement de ACID se fait ici. Imaginons que la racine de votre serveur web est /var/www/html. Installez ACID dans la racine d'apache :

COMMANDES	**REMARQUES**
cd /var/www/html	Placez−vous dans la racine du serveur web
tar −xvzf acid*	Décompactage de ACID

tar −xvzf adodb*	Décompactage de AdoDB
tar −xvzf phplot*	Décompactage de PHPlot
vi /var/www/html/acid/acid_conf.php	Renseignez les champs suivants :

- $DBlib_path="../adodb";
- $Chartlin_path="../phplot";
- alert_dbname="snort"
- alert_host="localhost"
- alert_user="user_snort"
- alert_password="snort_pwd"

Voilà, maintenant vous pouvez vérifier que ACID est bien configuré (allez voir sur http://localhost/acid). Si vous le souhaitez, l'accès peut se faire via certificat SSL de manière à crypter l'échange entre vous et le détecteur d'intrusions.

C.5 Fonctions de Snort

ConnexionBase	Initialise la connexion à la base de données
DeconnexionBase	Ferme une connexion précédemment ouverte
snortNombreEvenements	Nombre d'alertes reportées par Snort
snortConsulterSondes	Description des senseurs utilisés par Snort
snortConsulterAltertes	Signature, source/destination, date et nom des alertes enregistrées
snortGroupeAlertes	Liste des alertes (quantité, date début et fin, etc.) regroupées par signature
snortGroupesPriorites	Nombre d'alertes regroupées par priorité
snortAlertesPrioritaires	Liste des alertes ayant la plus haute priorité
snortAlertesPrioritairesCompletes	Idem, mais avec les adresses sources et destination
snortNombreIPSource	Nombre d'adresses IP reportées comme source d'alerte
snortNombreIPDestination	Idem, mais les adresses sont celles des destinations
snortNombreTCPPortSource	Nombre de ports TCP/IP sources d'alerte
snortNombreTCPPortDestination	Idem, mais pour les ports visés
snortNombreUDPPortSource	Nombre de ports UDP/IP sources d'alerte
snortNombreUDPPortDestination	Idem, mais pour les ports visés
snortAdresses	Liste de toutes les adresses IP sources d'alerte, avec l'adresse ciblée et les ports (pour TCP, UDP et/ou ICMP)
snortPartAdresseSource	Liste des adresses IP sources, avec leur pourcentage d'occurrences
snortPartAdresseDestination	Idem, pour les adresses IP ciblées
snortPartPortDestination	Liste des ports ciblés, avec leur pourcentage d'occurrences
snortPartPriorites	Listes de alertes par priorité, avec leur pourcentage d'occurrences
snortPartPrioritesHeure	Listes des alertes par priorité, ordonnées pas tranche horaire
snortPaysPopulaires	Répartition géographique des adresses IP sources d'alerte

D. Détails sur l'Ensemble de Données KDD 99

D.1 Introduction :

C'est l'ensemble de données utilisé dans la troisième compétition internationale des outils KDD (Knowledge Discovery and Data Mining) [56], qui a eu lieu en même temps que la Kdd 99, la cinquième conférence internationale sur les KDD. L'objectif de la compétition était d'obtenir le meilleur modèle prédictif pour la détection d'intrusions. Dans ce qui va suivre on va présenter les procédure de la collecte et de récapitulation de ces données, ainsi que le contenu des ces dernières.

D.2 Collecte des Données – Ensemble de Données DARPA 98 :

L'ensemble de données KDD99 est basé sur l'ensemble de données pour la détection d'intrusion DARPA 98. L'évaluateur des systèmes de détection d'intrusions off line DARPA 98 était le premier corpus standard d'évaluation dans ce domaine adopté sous la direction du Laboratoire des Recherches de l'Armée de l'Air Américain et de la DARPA-IPTO (Defense Advanced Research Projects Agency – Information Processing Technologie Office).

Sept semaines de rassemblement de données d'entraînement avec des attaques étiquetées ont eu lieu, utilisées par la suite pour le développement des systèmes. Suivi de deux semaines de rassemblement des attaques d'essais, non étiquetées, utilisées pour une évaluation aveugle des systèmes développés. Six différents Systèmes de Détection d'Intrusion ont été testés en utilisant les données rassemblées.

Un ensemble d'ordinateurs de test produisaient les données en émulant des centaines d'utilisateurs agissant l'un sur l'autre dans des milliers de sites. Avec le trafic de masse, il y avait plus de 300 instances de 38 attaques différentes contre trois machines victimes opérant sous le noyau UNIX *(les systèmes d'exploitation SunOS, Solaris, et Linux)*. Les données de test incluent des attaques créées spécifiquement pour l'évaluation, des attaques récentes, et les attaques des données d'entraînement.

Les détails de l'évaluation DARPA 1998 peuvent être trouvés dans [25], [12], et [55]. Les résultats de l'évaluation ont été analysés en traçant des taux de détection d'attaques contre les taux des fausses alarmes en utilisant des courbes caractéristiques du fonctionnement des récepteurs. Beaucoup de systèmes de détection d'intrusions pouvaient

détecter les attaques utilisées dans les données d'entraînement avec une exactitude élevée (63% à 93%) et peu de fausses alarmes (10 par jour). Cependant, ces systèmes donnent des mauvaises performances avec les nouvelles attaques, particulièrement quand le mécanisme de ces attaques diffère de celui des attaques utilisées dans l'ensemble d'entraînement [12].

Attaques	Categories	Ensemble d'Entraînement	Ensemble de Test
Apache2.	*DOS*	*0*	*794*
Back.	*DOS*	*2203*	*1098*
Buffer overflow.	*U2R*	*30*	*22*
Ftp write.	*R2L*	*8*	*3*
Guess passwd.	*R2L*	*53*	*4367*
Httptunnel.	*R2L*	*0*	*158*
Imap.	*R2L*	*12*	*1*
Ipsweep.	*PROBE*	*1247*	*306*
Land.	*DOS*	*21*	*9*
Loadmodule.	*U2R*	*9*	*2*
Mailbomb.	*DOS*	*0*	*5000*
Mscan.	*PROBE*	*0*	*1053*
Multihop.	*R2L*	*7*	*18*
Named.	*R2L*	*0*	*17*
Neptune.	*DOS*	*107201*	*58001*
Nmap.	*PROBE*	*231*	*84*
Normal.	*NORMAL*	*97277*	*60593*
Perl.	*U2R*	*3*	*2*
Phf.	*R2L*	*4*	*2*
Pod.	*DOS*	*264*	*87*
Portsweep.	*PROBE*	*1040*	*354*
Processtable.	*DOS*	*0*	*759*
Ps.	*U2R*	*0*	*16*
Rootkit.	*U2R*	*10*	*13*
Saint.	*PROBE*	*0*	*736*
Satan.	*PROBE*	*1589*	*1633*
Sendmail.	*R2L*	*0*	*17*
Smurf.	*DOS*	*280790*	*164091*
Snmpgetattack.	*R2L*	*0*	*7741*
Snmpguess.	*R2L*	*0*	*2406*

Spy	*R2L*	*2*	*0*
Sqlattack.	*U2R*	*0*	*2*
Teardrop.	*DOS*	*979*	*12*
Udpstorm.	*DOS*	*0*	*2*
Warezclient.	*R2L*	*1020*	*0*
Warezmaster.	*R2L*	*20*	*1602*
Worm.	*R2L*	*0*	*2*
Xlock.	*R2L*	*0*	*9*
Xsnoop.	*R2L*	*0*	*4*
Xterm.	*U2R*	*0*	*13*

Tab. D.1. Statistiques sur le Nombre des Labels pour l'ensemble de Données KDD 99

Nom	Description	Type
Duration	*Langueur (nombre des secondes) de la connexion*	*Continu*
Protocol_type	*Type de protocole, ex. tcp, udp, etc.*	*Discret*
Service	*Service réseau de destination, ex. http, telnet, etc.*	*Discret*
Src_bytes	*Nombre des octets de la source vers la destination*	*Continu*
Dst_bytes	*Nombre des octets de la destination vers la source*	*Continu*
Flag	*Statut normal ou bien erreur de la connexion*	*Discret*
Land	*1 si la connexion est de/vers le même hôte /port; 0 sinon*	*Discret*
Wrong_fragment	*Nombre de faux fragments*	*Continu*
Urgent	*Nombre de paquets urgents*	*Continu*

Tab. D. 2. Attributs de Bases de l'Ensemble KDD 99

Nom	Description	Type
Hot	*Nombre d'indicateurs "hot"*	*Continu*
Num_failed_logins	*Nombre des tentives login échouée*	*Continu*
Logged_in	*1 si le login a réussi, 0 sinon*	*Discret*
Num_compromised	*Nombre des conditions compromises*	*Continu*
Root_shell	*1 si le Shell administrateur est obtenu; 0 sinon*	*Discret*
Su_attempted	*1 si la commande « su root » a été tentée; 0 sinon*	*Discret*
Num_root	*Nombre des accès administrateur*	*Continu*
Num_file_creations	*Nombre des operations de creation des fichiers*	*Continu*
Num_shells	*Nombre des Invites Shell*	*Continu*
Num_access_files	*Nombre des opérations d'accès aux fichiers contrôlés*	*Continu*
Num_outbound_cmd	*Nombre des commandes outbound dans une session FTP*	*Continu*
Is_hot_login	*1 si le login appartient à la liste des hôtes; 0 sinon*	*Discret*
Is_guest_login	*1 si le login est un login invité; 0 sinon*	*Discret*
Count	*Nombre des connexions vers le même hôte que la connexion courante pendant les deux secondes passées. Note: les connections suivantes réfèrent à ces même connexions hôte.*	*Continu*
Serror_rate	*% de connexions qui ont des erreurs "SYN"'*	*Continu*
Rerror_rate	*% de connexions qui ont des erreurs "REJ"'*	*Continu*
Same_srv_rate	*% de connexions au même service*	*Continu*
Diff_srv_rate	*% de connexions aux différents services*	*Continu*
Srv_count	*Nombre de connexions au même service que le service courrant pendant les deux secondes passées Note: Les dispositifs suivants réfèrent à ces même connexions de service.*	*Continu*
Srv_serror_rate	*% de connexions qui ont des erreurs "SYN"*	*Continu*
Srv_rerror_rate	*% de connexions qui ont des erreurs "REJ"*	*Continu*
Srv_diff_host_rate	*% de connexions vers des hôtes différentes.*	*Continu*

Tab. D. 3. Attributs Additionnels de l'Ensemble KDD 99

E. Apprentissage et Algorithme EM

Nous présentons dans le paragraphe 3.2.1 le modèle de mélange de gaussiennes utilisé pour modéliser les paramètres de chaque requête.

Nous détaillons dans le paragraphe 3.2.2 l'algorithme EM. Nous donnons dans le paragraphe 3.2.3 les contraintes d'application de l'algorithme EM et enfin, dans le paragraphe 3.2.4, nous détaillons la méthode minimisation d'entropie permettant de trouver le nombre optimal de k, c'est-à-dire le nombre de clusters correspondant au nombre de fonctions élémentaires dans la fonction pdf.

Le modèle de mélange Gaussiennes : *GMM (Gaussian Mixture Model)*

Nous modélisons le comportement de chacun des vecteurs par une fonction pdf suivant le modèle de mélange de distributions. Pour la loi statistique nous avons choisi d'utiliser la loi normale (voir paragraphe 3.1). La fonction élémentaire $\phi(\overline{y}, \overline{\mu}_k, \overline{\Sigma}_k)$, exprimée dans l'équation 3.1, est alors remplacée par la fonction de distribution d'une loi normale $\phi(\overline{y}, \overline{\mu}_k, \overline{\Sigma}_k)$, l'expression de la fonction pdf du vecteur aléatoire $\overline{y}$ devient alors :

$$f(\bar{y},\bar{\Psi}) = \sum_{k=1}^{K} \omega_k \phi(\bar{y},\bar{\mu_k},\overline{\Sigma}_k)$$

Eq 3.2

Avec :

$\phi(\overline{y}, \overline{\mu}_k, \overline{\Sigma}_k)$: la fonction de distribution d'une loi normale

$\overline{\mu}_k$: représente la moyenne.

$\overline{\Sigma}_k$: représente la matrice de covariance

K : le nombre de nuage

$[\omega_1, \omega_2, \ldots . \omega_k, \overline{\mu}_1, \overline{\mu}_2, \ldots . \overline{\mu}_k, \overline{\Sigma}_1, \overline{\Sigma}_2, \ldots . \overline{\Sigma}_k]$: est le vecteur d'inconnus.

Lorsque toutes les fonctions pdf élémentaires décrivent une distribution d'une loi normale, le modèle général de mélange de distribution est appelé : modèle de mélange Gaussiennes ou GMM (pour Gaussiennes Mixture model).

Algorithme *EM*

L'algorithme *EM,* utilisé pour optimiser le modèle de mélange de distribution, est basé sur l'approche du maximum de vraisemblance afin d'approximer la fonction pdf d'ordre k d'un vecteur aléatoire $\overline{y}$. Il suppose connu k, le nombre de fonctions élémentaires pour la fonction pdf. L'algorithme EM est détaillé dans [16]. Nous détaillons les étapes *E*-step et *M*-step de chaque itération de l'algorithme *EM.*

Le but de la méthode du maximum de vraisemblance *(ML) est trouver ne estimation* Ψ* pour Ψ qui maximise la vraisemblance du vecteur aléatoire $\overline{y}$ pour un ensemble d'observation $\overline{y}$=[$\overline{y}$1, $\overline{y}$2,……$\overline{y}_{\mathrm{p}}$] $^{\mathrm{T}}$.

Nous supposons que les différentes réalisations $\overline{y}$1, $\overline{y}$2,……$\overline{y}_{\mathrm{p}}$ du vecteur $\overline{y}$ sont indépendantes.

La fonction de vraisemblance logarithmique pour Ψ est la suivante :

$$\log L_C\ (\overline{\Psi}) = \sum_{j=1}^{p} \log \sum_{K=1}^{k} \omega_{\mathrm{k}}^{i}\ \mathrm{p}\ \left(\overline{\mathrm{y}}_j\ |\mathcal{K}\right) \qquad \text{Eq 3.3}$$

Une estimation de Ψ selon le critère du maximum de vraisemblance est obtenue en trouvant une solution à l'équation suivante (3.4) qui correspond à un maximum local de (3.3) :

$$\log L_C\ (\overline{\Psi}) \qquad \text{Eq 3.4}$$

Sachant qu'il est difficile d'optimiser Ψ directement, des variable cachées z_{jk} (non observées) sont alors introduites dans l'ensembles des observations tel que z_{jk} est égale à 1 si $\overline{\mathrm{y}}_j$ est issu du k$^{\mathrm{eme}}$ composant du modèle de mélange de gaussiennes, z_{jk} est égale à 0 dans le cas contraire (j = 1,…p ; k = 1,…k).

Soit $\overline{\mathrm{Z}}_j$ = [z_{j1}, $\mathrm{z}_{\mathrm{j2,.....}}$, z_{jk}]$^{\mathrm{T}}$ le vecteur des variables cachées pour l'observation $\overline{\mathrm{y}}_j$ nous définissons le vecteur de données complètes $\overline{\mathrm{X}}_j$ (non observé) par (3.5) :

$$\overline{X}=[\ \overline{x}1,\ \overline{x}2,\ldots\ldots\overline{x}_{\mathrm{k}}\ ]^{\mathrm{T}} \qquad \text{Eq 3.5}$$

Pour cette spécification, la vraisemblance logarithmique pour le vecteur de données complètes est donnée par (3.6) :

$$Q\left(\overline{\Psi},\overline{\Psi}^{i}\right) = E_{\overline{\Psi}^{i}}\left(\log L_{C}\left(\overline{\Psi}\right)|\overline{\mathrm{y}}\right) \qquad \text{Eq3.6}$$

L'algorithme *EM*, défini dans [16], s'adapte bien lorsque la maximisation de la vraisemblance des données complètes (3.6) est plus facile que la maximisation de la vraisemblance des données incomplètes (3.3). L'algorithme *EM* propose donc de trouver itérativement une estimation de Ψ qui donne un maximum local (s'il existe) pour l'équation (3.6). Il consiste à fournir une estimation pour la distribution qui permet indirectement d'obtenir une estimation de Ψ. En d'autres termes, maximiser l'équation (3.3) revient à maximiser l'équation (3.6) (lorsque les observations sont indépendantes)

L'algorithme propose deux étapes dans chaque itération : l'étape *E*-step de l'itération (i + 1) consiste à calculer (3.7)

$$Q\left(\overline{\Psi},\overline{\Psi}^{i}\right) = E_{\overline{\Psi}^{i}}\left(\log L_{C}\left(\overline{\Psi}\right)|\overline{\mathrm{y}}\right) \qquad \text{Eq3.7}$$

Ou $Q\left(\overline{\Psi},\overline{\Psi}^{i}\right)$ est l'espérance de la vraisemblance logarithmique des données complétes ($\log L_{C}\left(\overline{\Psi}\right)$) définie dans (3.6), étant données les observations $\overline{\mathrm{y}}$ et l'estimation courante $\overline{\Psi}^{i}$ de $\overline{\Psi}$.

Etant donné que $\log L_{C}\left(\overline{\Psi}\right)$ est une fonction linéaire des variables z_{jk} non observables, l'étape E est exécutée en remplaçant chaque z_{jk} par son espérance étant données l'observation $\overline{\mathrm{y}}_{\mathrm{j}}$ et l'estimation courante $\overline{\Psi}^{i}$ de $\overline{\Psi}$.

L'équation (3.7) est remplacée comme suit par l'équation (3.8)

$$T_{K}\left(\overline{\mathrm{y}}_{j},\overline{\Psi}^{i}\right) = E_{\overline{\Psi}^{i}}\left(z_{jk}\left|\overline{\mathrm{y}}_{j}\right.\right) = \frac{\omega_{\mathrm{k}}^{i}\,\phi\left(\overline{\mathrm{y}}_{j},\overline{\mu}_{\mathrm{k}}^{i},\overline{\Sigma}_{\mathrm{k}}^{i}\right)}{\sum_{\acute{K}=1}^{k}\omega_{\acute{K}}^{i}\,\phi\left(\overline{\mathrm{y}}_{j},\overline{\mu}_{\mathrm{k}}^{i},\overline{\Sigma}_{\acute{K}}^{i}\right)} \qquad \text{Eq 3.8}$$

Nous constatons que $T_{K}\left(\overline{\mathrm{y}}_{j},\overline{\Psi}^{i}\right)$ exprime l'estimation courante de la probabilité a priori que la $\mathrm{j}^{\mathrm{eme}}$ réalisation $\overline{\mathrm{y}}_{j}$ provienne du $\mathrm{k}^{\mathrm{eme}}$ nuage ($\mathrm{p}\left(\overline{\mathrm{y}}_{j}\,|\mathcal{K}\right)$).

L'équation (3.8) peut être donc réécrite sachant que :

$$P\left(\mathcal{K}\left|\overline{y}_j\right.\right) = \frac{\omega_k^i \ \phi\left(\overline{y}_j \ \overline{\mu}_k^i, \overline{\Sigma}_k^i\right)}{\sum_{\acute{K}=1}^{k} \omega_{\acute{K}}^i \phi\left(\overline{y}_j \ , \overline{\mu}_k^i, \overline{\Sigma}_{\acute{K}}^i\right)} = \frac{\omega_k^i \ p\left(\overline{y}_j \ |\mathcal{K}\right)}{p\left(\overline{y}_j\right)} = \frac{p(\mathcal{K})\, p\left(\overline{y}_j \ |\mathcal{K}\right)}{p\left(\overline{y}_j\right)} \qquad \text{Eq 3.9}$$

L'équation (3.9) n'est autre que l'expression du théorème de Bayes tel que l'estimation de la probabilité de chaque composant p(k) est obtenue par l'estimation courante du coefficient de pondération ω_k^i .

En combinant l'équation (3.7) et l'équation (3.8), nous obtenons l'expression de l'étape E-step suivante :

$$Q\left(\overline{\Psi}, \overline{\Psi}^i\right) = \frac{\sum_{\acute{K}=1}^{k} \log \sum_{\acute{K}=1}^{k} \omega_k^i \ \phi\left(\overline{y}_j \ \overline{\mu}_k^i, \overline{\Sigma}_k^i\right)}{p} \qquad \text{Eq 3.10}$$

Equivalente à :

$$L\left(\Psi^i\right) = Q\left(\overline{\Psi}, \overline{\Psi}^i\right) = \frac{\sum_{j=1}^{p} \log \sum_{\acute{K}=1}^{k} \omega_k^i \ p\left(\overline{y}_j \ |\mathcal{K}\right)}{p} = \frac{\sum_{j=1}^{p} \log \left(p\left(\overline{y}_j\right)\right)}{p} \quad \text{Eq 3.11}$$

L'objectif de l'étape M-step de l'itération (i + 1) est de choisir la valeur $\overline{\Psi}^{i+1}$ de $\overline{\Psi}$ qui maximise $Q\left(\overline{\Psi}, \overline{\Psi}^i\right)$ il s'agit, en d'autres termes, d'ajuster les valeurs des coefficients de pondération (ω_k^{i+1}), des moyennes ($\overline{\mu}_k^{i+1}$) et des matrices de covariance ($\overline{\sum}_k^{i+1}$) comme suit :

$$\omega_k^{i+1} = \frac{\sum_{j=1}^{p} T_K\left(\overline{y}_j \ , \overline{\Psi}^i\right)}{p} \quad \text{Eq 3.12}$$

$$\overline{\mu}_k^{i+1} = \frac{\sum_{j=1}^{p} T_K\left(\overline{y}_j \ , \overline{\Psi}^i\right) \overline{y}_j}{\sum_{j=1}^{p} T_K\left(\overline{y}_j \ , \overline{\Psi}^i\right)} \qquad \text{Eq 3.13}$$

$$\overline{\sum}_k^{i+1} = \frac{\sum_{j=1}^{p} T_K\left(\overline{y}_j \ , \overline{\Psi}^i\right)\left(\overline{y}_j - \overline{\mu}_k^{i+1}\right)\left(\overline{y}_j - \overline{\mu}_k^{i+1}\right)^T}{\sum_{j=1}^{p} T_K\left(\overline{y}_j \ , \overline{\Psi}^i\right)} \qquad \text{Eq 3.14}$$

Equivalente à :

$$\omega_k^{i+1} = \frac{\sum_{j=1}^{p} T_K\left(\overline{y}_j \ , \overline{\Psi}^i\right)}{p} \quad \text{Eq3.15}$$

$$\overline{\mu}_{\mathrm{k}}^{i+1} = \frac{\sum_{j=1}^{p} \mathrm{P}\left(\mathcal{K}\middle|\overline{\mathrm{y}}_j\right)\overline{\mathrm{y}}_j}{\sum_{j=1}^{p} \mathrm{P}\left(\mathcal{K}\middle|\overline{\mathrm{y}}_j\right)} \quad \text{Eq 3.16}$$

$$\overline{\sum}_{\mathrm{k}}^{i+1} = \frac{\sum_{j=1}^{p} \mathrm{P}\left(\mathcal{K}\middle|\overline{\mathrm{y}}_j\right)\left(\overline{\mathrm{y}}_j - \overline{\mu}_{\mathrm{k}}^{i+1}\right)\left(\overline{\mathrm{y}}_j - \overline{\mu}_{\mathrm{k}}^{i+1}\right)^{\mathrm{T}}}{\sum_{j=1}^{p} \mathrm{P}\left(\mathcal{K}\middle|\overline{\mathrm{y}}_j\right)} \quad \text{Eq 3.17}$$

Une caractéristique intéressante de l'algorithme EM est de garantir le fait que la vraisemblance logarithmique du modèle de mélange de gaussiennes $L\left(\Psi^i\right)$ ne peut jamais baisser après chaque itération de l'EM. Nous en déduisons que L $(\overline{\Psi}^{i+1})$) ≥ (L $(\overline{\Psi}^{i})$ ce qui implique la convergence de $L\left(\Psi^i\right)$ vers L*, si $L\left(\Psi^i\right)$ est borné.

Les étapes E-step et M-step sont répétées plusieurs fois jusqu'à arriver à une variation de l'estimation de la vraisemblance logarithmique (ou des estimations des paramètres) assez petite indiquant la convergence vers L*.

Nous donnons les différentes étapes de l'algorithme :

1. Initialiser $\overline{\Psi}^{*}$ avec des valeurs aléatoires pour $\omega_{1,}^{0}\omega_{2,\ldots,}^{0}\omega_{k}^{0}\,\overline{\mu}_{1,}^{0}\overline{\mu}_{2,\ldots}^{0}\overline{\mu}_{k,}^{0}\,\overline{\sum}_{1,}^{0}\overline{\sum}_{2\ldots,}^{0}\overline{\sum}_{\mathrm{k}}^{0}$
2. Initialiser i = 0 et calculer $L\left(\Psi^i\right)$ en évaluant l'équation (3.11), ce qui correspond à l'exécution d'une étape E-step
3. pour i = i + 1, calculer la nouvelle valeur de Ψ^i en exécutant l'étape M-step (équations 3.15, 3.16, 3.17) et de $L\left(\Psi^i\right)$ en exécutant l'étape E-step (équations 3.11).
4. si L $L\left(\Psi^i\right) - L\left(\overline{\Psi}^{\mathrm{i}-1}\right) > \delta$ (pas encore convergé), répéter l'étape (3)
5. mettre à jour les paramètres $\overline{\Psi}^{*}$ du modèle courant avec ceux obtenus pour le modèle convergé : $\overline{\Psi}^{*} = \overline{\Psi}^{i}$
   Où δ est une constante de convergence.

Algorithme EM adapté

1. Initialiser le compteur c = 0
2. c = c + 1

3. Initialiser $\overline{\Psi}^{0}$ avec des valeurs aléatoires pour $\omega_{1,}^{0}\omega_{2,\ldots,}^{0}\omega_{k}^{0}\,\overline{\mu}_{1,}^{0}\overline{\mu}_{2,\ldots}^{0}\overline{\mu}_{k,}^{0}\,\overline{\sum}_{1,}^{0}\overline{\sum}_{2\ldots,}^{0}\overline{\sum}_{\mathrm{k}}^{0}$

4. Initialiser i = 0 et calculer $L\left(\Psi^i\right)$ en évaluant l'équation (4.11), ce qui correspond à l'exécution d'une étape E-step

5. pour i = i + 1, calculer la nouvelle valeur de $\overline{\Psi}^{i}$ en exécutant l'étape *M-step* (équations 4.15, 4.16, 4.17) et de $L(\Psi^{i})$ en exécutant l'étape *E-step* (équations 4.11).

6. Si $L(\Psi^{i}) - L(\overline{\Psi}^{i-1}) > \delta$ (pas encore convergé), répéter l'étape (5)

7. Si $\left(\left(|\det(\overline{\sum}_1^{i}\right| < \varepsilon\right) \| \left(|\det(\overline{\sum}_2^{i}\right| < \varepsilon\right) \| \cdots\cdots \| \left(|\det(\overline{\sum}_{\mathrm{k}}^{i}\right| < \varepsilon\right)\right)$ alors répéter l'étape (2)

8. Si (c = 1) ‖ $L(\Psi^{i}) > l_{opt}$) alors

$$l_{opt} = L(\Psi^{i})$$
$$\Psi_{opt} = \overline{\Psi}^{i}$$

9. Si c $\leq C_{max}$ alors répéter l'étape (2)
10. mettre à jour les paramètres Ψ^{i} du modèle courant avec ceux obtenus pour le modèle convergé : $\overline{\Psi}^{*} = \overline{\Psi}_{\mathrm{opt}}$ Où δ est une constante de convergence.

F. Librairie Bayesian Network tools for Java
(BNJ Ver 3.3)

F.1 Architecture du noyau

- La librairie BNj de decompose en plusieurs parties:
  - ☐ Core classes – edu.ksu.cis.bnj.bbn package
    For expressing graphs, nodes, edges, cpts (CPFs), PDFs (cpt's entries).
  - ☐ Inference – edu.ksu.cis.bnj.bbn.inference package
    Exact and inexact inference
  - ☐ Learning – edu.ksu.cis.bnj.bbn.learning
    Structure learning

1. Core Classes

- ☒ Mainly inherited from OpenJGraph
- ☒ BBNGraph: Bayes Net Graph
- ☒ BBNNode: Bayes Net Node
- ☒ BBNCPT: Conditional Probability Table
  - o Called CPF[unction] because support for continuous values planned
  - o Full CPFs not yet implemented
- ☒ BBNPDF: CPT entries
  - o PDF = Probabilistic Distribution Function; again, for continuous values
- ☒ BBNValue: Abstract class for discrete (BBNDiscreteValue) and continuous values (BBNContinuousValue) → will be phased out in upcoming release
- ☒ BBNConstant: Derived class of BBNPDF for representing constants (superfluous, will be phased out)
- ☒ EvidenceParser → Class to parse evidence values. (should not be called externally)

BBNGraph

- ☒ To create a Bayes net graph:
  BBNGraph g = new BBNGraph();
- ☒ To load a Bayes net:
  BBNGraph g =
  BBNGraph.load("file");
- ☒ To save a graph:
  g.save("file"); – or –
  g.save("file", "format");
  e.g., g.save("mynet.net", "net");
- ☒ To load evidence:
  g.loadEvidence("evidencefile");
- ☒ To save evidence:

BBNNode

- ☒ To create a node:
  BBNNode node = new BBNNode();
  node.setName("name");
  // must then add node to graph
- ☒ To set node values:
  BBNDiscreteValue v = new
  BBNDiscreteValue();
  v.add("v1"); v.add("v2"); // … and
  so on
  node.setValues(v);
- ☒ To get node values:
  BBNValue v = node.getValues();
- ☒ To set or get evidence:

g.saveEvidence("evidencefile");

- ☒ To print graph contents for debugging:
  System.out.println(g.toString());
- ☒ To add or remove a node:
  g.addNode(bbnNode);
  g.removeNode(bbnNode); // related edges will also be deleted
- ☒ To add or remove an edge:
  g.addEdge(node1, node2);
  g.removeEdge(node1, node2);
- ☒ To topologically sort a graph:
  List nodeList = g.topologicalSort();

node.setEvidenceValue(val); // val must be in the BBNValue
Object val = node.getEvidenceValue();

- ☒ To turn node into Decision / Utility node:
  node.setType(BBNNode.DECISION);
  node.setType(BBNNode.UTILITY);
- ☒ To inquire about a node:
  node.isDecision();
  node.isUtility();
  node.isEvidence();
- ☒ To access CPT:
  BBNCPF cpf = node.getCPF();
  // Note: Don't use query from the node directly // as it will be phased out soon

BBNCPF

- ☒ CPFs should not be created individually unless you know what you are doing (e.g. creating ICPTs for SIS or AIS)
- ☒ To create a CPF object yourself, use:
  List l = new LinkedList();
  l.add("node1"); l.add("node2");
  // etc, add the node names involved for the CPT (in string)
  BBNCPF cpf = new BBNCPF(l);
- ☒ Let a and b be the parents of node c
- ☒ Let all nodes be Boolean
- ☒ To query content of CPT entry for (a = true, b = true, c = false):
  Hashtable t = new Hashtable();
  t.put("a", "true"); t.put("b", "true");
  t.put("c", "false");
  double value = c.query(t);

2. Inference Classes

- ☒ Inference: Abstract class that all inference modules should inherit from
- ☒ ExactInference: Abstract class for all exact inference
- ☒ ApproximateInference: Abstract class for approximate inference
- ☒ MCMC: For MCMC based approximate inference (edu.inference.approximate.sampling)
- ☒ Example (Lauritzen-Spiegelhalter, i.e., junction tree):
  BBNGraph g = BBNGraph.load("netfile");
  g.loadEvidence("evidenceFile"); // if needed
  LS ls = new LS(g);
  InferenceResult result = ls.getMarginals();

- ☒ Variable result: printable hash table System.out.println(result.toString());
- ☒ Getting all available attributes:
  List l = db.getAttributes();
- ☒ Getting satellite attributes (non-primary key and non-reference key):
  List l = db.getRelevantAttributes();
  // In case of single table, getAttributes == getRelevantAttributes()

3. Learning Classes

- ☒ Just like inference classes, Learning has a parent class Learner from which all structural learning classes should inherit
- ☒ ScoreBasedLearner – parent class for learner that has scores in it
- ☒ CIBasedLearner – parent class for conditional-independence-based learner
- ☒ Currently BNJ only has ScoreBasedLearner
- ☒ Needed for **ScoreBasedLearner** to evaluate candidate/structure
- ☒ Idea: examine different scoring schemes to evaluate impact on learning algorithm
- ☒ Currently we have **BDEScore** and **DiscrepancyScore**
- ☒ Others are just skeletons at the moment
- ☒ To make a new scoring scheme, must inherit **LearnerScore**
- ☒ Must override **double getScore(int curNode, int candidate, Set[] parentTable)**
- ☒ **curNode**: index of the node being evaluated
- ☒ **candidate**: index of candidate parent (-1 if not applicable; *i.e.*, for structure scorer)
- ☒ **parentTable**: set of integers of currently evolving structure.

Pour plus de details voir le lien http://www.kddresearch.org/Groups/Probabilistic-Reasoning/Docs/

REFERENCES

1. Hervé Debar, Marc Dacier et Andreas Wespi, "A Revised Taxonomy for Intrusion-Detection Systems ", Annales des Télécommunications, n° 7-8, 2000, 55 p.

2. Ludovic Mé, Zakia Marrakchi, Cédric Michel, Hervé Debar et Frédéric Cuppens. "La détection d'intrusions : les outils doivent coopérer" – Revue de l'Electricité et de l'Electronique. No 5, mai 2001, 50-55.

3. D.E. Denning. "An Intrusion-Detection Model", IEEE transaction on Software Engineering 13, 2, 1987, 222-232.

4. S. Forrest and S.A. Hofmeyr and A. Somayaji. "Computer Immunology", Communications of the ACM, 40, 10, October 1997, 88-96.

5. C. Cachin, M. Dacier, O. Deak, K. Julisch, B. Randell, J. Riordan, A. Tscharner, A. Wespi, and C. Wüest. "Towards a taxonomy of intrusion detection systems and attacks". MAFTIA Project IST-, IBM Research, Septembre 2001, 1999-11583.

6. C. Carver. "Intrusion Response Systems - A Survey". URL: http://faculty.cs.tamu.edu/pooch/course/CPSC665/Spring2001/Lessons/IntrusionDetectionandResponse/rtirs2.doc, 2000.

7. H. Debar, M. Dacier, and A. Wespi. "Towards a taxonomy of intrusion-detection systems". Computer. Networks, 31(9) : 1999, 805-822.

8. Saint Corporation. "Saint documentation contents". http ://www.saintcorporation.com/.

9. Z. Trabelsi,H. Rahmani, K. Kaouech, and M. Frikha. "Malicious sniffing systems detection platform". In Symposium on Applications and the Internet (SAINT2004), Tokyo, Japan, Janvier2004.IEEE Computer Society 2004, 201-207.

10. S. Edwards. “Network Intrusion Detection Systems: Important IDS Network Security Vulnerabilities”. Technical report, Top Layer Networks, Inc., September 2002.

11. M. Cooper, S. Northcutt, M. Fearnow, and K. Frederick. “Intrusion Signatures and analysis. New Riders”, 1 edition, Janvier 2001.

12. D. E. Denning. “An Intrusion-Detection Model”. In IEEE Trans. on Software Eng., V. 13, Février 1987, 222-232.

13. Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. “Bayesian event classification for intrusion detection”. In 19th Annual Computer Security Applications Conference, December 2003.

14. H. Debar, M. Dacier, and A. Wespi. “A revised taxonomy for intrusion-detection systems”. Annales des Télécommunications, 55(8) :, 2000, 361-378.

15. H. Debar, L. Mé, and S. F. Wu, editors. “Recent Advances in Intrusion Detection”, Third International Workshop, RAID 2000, Toulouse, France, October 2-4, 2000, Proceedings, v. 1907 of Lecture Notes in Computer Science. Springer, 2000.

16. F. Cuppens, S. Gombault, and T. Sans. “Selecting appropriate counter-measures in an intrusion detection framework”. In Computer Security Foundation Workshop, Juin 2004.

17. M. Roger and J. Goubault, “Log auditing through model checking”, In proceedings of the 14th IEEE Computer Security Foundations Workshop, pp220-236, 2001.

18. T. Toth. “Improving Intrusion Detection Systems”. PhD thesis, Technischen Universität Wien, Mai2003.

19. D. Anderson, T. Frivold, & A. Valdes, “Next-generation intrusion detection expert system”, 1995.

20. J. Pearl, “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference”, Morgan Kaufmann, 1997.

21. H. S. Teng, K. Chen, and S. Lu. “Security audit trail analysis using inductively generated predictive rules”. In the Sixth Conference on Artificial Intelligence Applications, Piscataway, New Jersey, Mars1990, 24- 29.

22. S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longsta_. “A sense of self for Unix processes”. In Proceedings of the 1996 IEEE Symposium on Security and Privacy. IEEE Computer Society, 1996, 120 p.

23. M. Fisk and G. Varghese. “Fast content-based packet handling for intrusion detection”. Rapport LA-UR-01-5459, University of California, San Diego, Department of Computer science and Engineering, 2001.

24. A. Feldmann and S. Muthukrishnan. “Trade for packet classification”. In INFOCOM (3), Mai 2000, 1193-1202.

25. Calvin Ko and Manfred Ruschitzka and Karl N. Levitt, “Execution Monitoring of Security Critical Programs in a Distributed System: A Specification-based Approach”, Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 1997, 175-187.

26. S. Cho and S. Han. “Two sophisticated techniques to improve hmm-based intrusion detection systems”. In Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection (RAID'2003), LNCS v. 2820, September 2003, 207-219.

27. Calvin Ko and Timotyy Redmond, “Noninterference and Intrusion Detection”, Proceedings of the IEEE Symposium on Security and Privacy, 2002.

28. Jacob Zimmermann and Ludovic Mé and Christophe Bidan, “An Improved Reference Flow Control Model for Policy-Based Intrusion Detection”, Proceedings of the 8th European Symposium on Research in Computer Security (ESORICS), October 2003.

29. R. Sekar and Yong Cai and Mark Segal , “A Specification-Based Approach for Building Survivable Systems ”, Proceedings of the 21st National Information Systems Security Conference (NISSC'98), October 1998, 338-347.

30. W. Lee and D.Xiang. “Information-theoretic measures for anomaly detection”. In Proceedings of the IEE Symposium on Security and Privacy. Page 130. IEEE Computer Society, 2001.

31. D. Barbara, “Applications Of Data Mining In Computer Security”, Kluwer Academic Publishers Group, 2002.

32. B. Le Charlier, ‘’ASAX: Software architecture and rule-based language for universal audit trail analysis’’. Springer Berlin, ISSN: 0302-9743, 1992.

33. J. R. Quinlan, “C4.5, Programs for machine learning”, Morgan Kaufmann San Mateo Ca, 1993.

34. S. T. Eckmann, G. Vigna, and R. A. Kemmerer. “STATL: an attack language for state based intrusion detection”. J. Comput. Secur., 10(1-2) :2002 ,71-103.

35. R Kumar, L Holloway. ‘’Supervisory Control of Deterministic Petri Nets with Regular Specification Languages’’, IEEE Transactions on Automatic Control, 1996.

36. M Roger, J Olivain ‘’LogWeaver : un outil de détection d'intrusions fondé sur la logique temporelle’’, 2002.

37. L. Mé, ‘’Audit de sécurité par algorithmes génétiques’’, Thèse de doctorat, université de Rennes, 1994.

38. W. Lee, S.J. Stolfo, and K.W.Mok. “Mining in a data flow environment: experience network intrusion detection; In proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining, 114-124, 1999.

39. Calvin Ko and Timotyy Redmond, “Noninterference and Intrusion Detection”, Proceedings of the IEEE Symposium on Security and Privacy, 2002.

40. P. Leray and O. Francois. “Réseaux bayésiens pour la classification – méthodologie et illustration dans le cadre du diagnostic médical”. Revue d’Intelligence Artificielle, 18/2004 :, 2004, 169–193.

41. A. Finn V. Jensen. “An introduction to Bayesian Networks”. Taylor and Francis, London, United Kingdom, 1996.

42. D. Heckerman, “A tutorial on learning with Bayesian networks”, Microsoft Research tech. report, MSR-TR-95-06, 1996.

43. K. Murphy, “A Brief Introduction to Graphical Models and Bayesian Networks”, Massachussets Institut of Technology, 1998.

44. T. Mitchell, “Machine Learning”, WCB/McGraw Hill, 1997.

45. C. Kruegel, D. Mutz, W. Robertson, F. Valeur, “Bayesian Event Classification for Intrusion Detection”; Reliable Software Group, University of California, 2003.

46. L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition”, Proceedings of the IEEE, 1990.

47. A. Cynthia S. Hood and Chuanyi Ji. “Proactive network fault detection”. In the INFOCOM ’97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution. IEEE Computer Society, 1997, 1147 p.

48. K.Kendall, “A DataBase of Computer Attacks for evaluation of Intrusion Detection System”, *MIT*, Department of electrical, Engineering and Computer Science, 2001.

49. P. Naïm, P-H. Wuillemin, P. Leray, O. Pourret, and A. Becker. “Réseaux bayésiens” Eyrolles, Paris, 2004.

50. “Kdd cup 99 intrusion detection data et task description”, *University of California Department of Information and Computer Science, 1999.*

51. J. Pearl. “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference”. Morgan Kaufmann,. RNTI – 1, 1988

52. R. S. Boyer and J. Strother Moore. “A fast string searching algorithm In Communications of the ACM”, 20(10), 1977, 762-772.

53. A. Briney. “CISCO Strategies” - Survey, Mars 2003.

54. B. Caswell and M. Roesch. “Snort rules database”. http ://www.snort.org/snort-db/

55. C. Charras and T. Lecroq. “Handbook of Exact String Matching Algorithms”. King's College London, Octobre 2004.

56. G. Cheung and S. McCanne. “Optimal routing table design of ip address lookups under memory constraints”. In Proceeding of IEEE Infocom, 1999, 1437-1444.

57. C. Coit, S. Staniford, and J. McAlerney. “Towards faster string matching for intrusion detection”. In Proc. of the DARPA Information Survivability Conference and Exhibition (DISCEX-02), 2002, 367-373.

58. B. Commentz-Walter. “A string matching algorithm fast on the average”. In Proceedings of the 6th Colloquium, on Automata, Languages and Programming,. Springer- Verlag, 1979, 118-132.

59. Microsoft Corporation. “Unauthorized access to IIS servers through ODBC data access with RDS”, Juillet 1999.

60. M. Crochemore, A. Czumaj, L. G_asieniec, T. Lecroq, T. Plandowski, and W. Rytter. “Fast practical multi-pattern matching”. Information Processing Letters, 71, 1999: 3-4.

61. M. Crochemore and W. Rytter. ‘’Text Algorithms”. Oxford University Press, 1994.

62. F. Cuppens. “Cooperative intrusion detection”. In International Sysmposium on Information superiority: tools for crisis and conflict-management, Paris, Septembre 2001.

63. D. Curry, H. Debar, and B. Feinstein. “The intrusion detection message exchange format”, Janvier 2004. IETF Intrusion Detection Working Group Internet Draft.

64. H. Debar, M. Becker, and D. Siboni. “A Neural Network Component for an Intrusion Detection System”. In Proceedings of the 1992 IEEE Computer Society Symposium on Reserach in Security and Privacy. IEEE, IEEE Service Center, Piscataway, NJ, 1992, 240-250.

65. S. Kummar. “A pattern Matching Approach to Misuse Intrusion Detection”. PhD thesis, Purde University, Department of computer Sciences, Aout 1995.

66. N. Desai. “Increasing performance in high-speed NIDS, a look at snort's internals”. Technical report, SecurityFocus, Mai 2002.

67. Dethy. “Examining port scan methods - Analysing audible techniques”. URL : www.in-for it/informatica/docs/portscan.pdf.

68. S.T. Eckmann. “Translating Snort rules to STATL scenarios”. In Proceedings of the 4th International Workshop on the Recent Advances in Intrusion Detection (RAID'2001), LNCS v. 2212, Octobre 2001, 69-84.

69. L. Mé, Gassata “A genetic Algorithm as an alternative tool for security audit analysis. In proceeding in the 1st International Workshop on the recent advanced Intrusion Detection September 1998.

70. M. Erlinger and S. Staniford-Chen. “Intrusion Detection Exchange Format (IDWG) ”. http ://www.ietf.org/html.charters/idwg-charter.html.

71. H. Welte et al. “The net filter/ ip tables project”. www.netfilter.org.

72. B. Feinstein, G. Matthews, and J. White. “The intrusion detection exchange protocol”, Octobre 2002. IETF Intrusion Detection Working Group Internet Draft.

73. U. Shankarand V. Paxson. “Active mapping: Resisting NIDS evasion without altering traffic”. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, IEEE Computer Society, 2003, 44 p.

74. R. Sommerand V. Paxson. “Enhancing byte-level network intrusion detection signatures with context”. In Proceedings of the10th ACM conference on Computer and communication security, ACM Press, 2003, 262-271.

75. Sourcefire. “Snort 2. 0 rule optimizer”. Technical report.

76. K. Tan. “The application of neural networks to Unix computer security”. In Proceedings of the IEEE International Conference on Neural Networks, V.1, 1995.

77. C. M. Bishop, “Neural Networks for Pattern Recognition”, Oxford University Press, Oxford, U.K, 1995.

78. A. Gelman, J. B. Carlin, H. S. Stern & D. B. Rubin, “Bayesian Data Analysis”, Chapman & Hall, London, 1995.

Articles de l'Auteur

Publications

1. M. Mehdi, S. Zair, A. Anou and M. Bensebti, " A Bayesian Networks in Intrusion Detection Systems", Journal of Computer Science 3 (5): 259-265,2007, ISSN 1549-3636, 2007 Science Publications.

2. M. Mehdi, M. Bensebti, A. Anou and M. Djebari ," Real Time Solution for Computer Network Intrusion Detection", Issue I volume 5, January 2006 ISSN 1109-2750, Transactions on Computers World Scientific and Engineering Academy and Society.

Communications

1. M. Mehdi and M. Bensebti, "Statistical Model for Intrusion Detection System", IEEE Conference, SETIT09, Hammamet Tunisie, 22-26 March 2009.

2. M. Mehdi, M. Bensebti, A. Anou and M. Djebari "La Sécurité dans les Réseaux Ad Hoc", IEEE Conference, JTEA2008, IEEE Symposium on Security Hammamet Tunisie, *02-04 Mai 2008.*

3. M. Mehdi, s. Zair, A. Anou and M. Bensebti, "Bayesian Intrusion Detection System for Computer Network", IEEE Conference, SETIT07, Hammamet Tunisie, *21-25 Mars 2007.*

4. M. Mehdi, M. Bensebti, A. Anou and M. Djebari "La Sécurité dans les Réseaux Ad Hoc", IEEE Conference, SETIT07, Hammamet Tunisie, *21-25 Mars 2007.*

5. M. Mehdi, M. Bensebti, A. Anou and M. Djebari, "A Bayes Model for Real-Time Computer Network Intrusion Detection Systems", IEEE Conference Algeria, *25-26 July 2005.*

6. M. Mehdi, M. Bensebti, A. Anou and M. Djebari, "A Bayesian Classification Model for Real time Network intrusion Detection, World Scientific and Engineering Academy and Society, transactions in computers, *16-18 December 2005* Tenerife, Canary Islands, *Spain.*

7. M. Mehdi, M. Bensebti, A. Anou and M. Djebari, "A Bayes Model for Real-Time Computer Network Intrusion Detection Systems", IEEE Conference Chypre, 25-26 June 2004.