République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université M'hamed BOUGARA de BOUMERDES

Faculté des Sciences
Département d'Informatique

MEMOIRE DE MAGISTER

Spécialité : Informatique
Option : Spécification de Logiciel et Traitement de l'Information

Présenté par :
M^elle^ Rebiha HADAOUI

UN IDS basé sur un algorithme inspiré du fonctionnement de colonies des fourmis

Proposé par : Mr.K.Tamine ***Maître de conférences*** à l'université de limoges

Soutenu devant le jury:

Mr. M. Mezghiche	Professeur à l'université de Boumerdes	Président
Mr. M.Lallem.	Professeur à l'université de Tizi-Ouzou	Examinateur
Mr. M. Ahmed necer	Professeur à l'université de Bab ezzouar	Examinateur
Mr. K. Tamine	HDR. à l'université de Limoges	Rapporteur

Année Universitaire : 2008/2009

Remerciements

Je tiens à exprimer ma profonde reconnaissance et la preuve de ma grande gratitude pour KARIM TAMINE, Maître de conférences à l'université de LIMOGES (FRANCE). Pour m'avoir proposé ce thème, pour son aide, ses conseils sa disponibilité et le suivi qu'il m'apporté tout au long du mon travail.

Je tiens à remercier Monsieur *Slimane SKENDRAOUI*, ingénieur d'Etat en électronique, chef de section maintenance, pour m'avoir soutenue durant toute la période de préparation, pour sa présence morale et physique, pour son aide, pour ces mots encourageons.

Je tiens à remercier également les membres du jury pour avoir bien voulu examiner et avoir accepter de juger ce travail.

Mes remerciements vont également :

A Melle Ghenima BOURKACHE enseignante a l'université de BOUMERDES, pour son aide et ses conseils.

A Omar et Dahdah, pour leur aide et leur disponibilité.

Je dédie ce modeste travail :

A mes très chers adorables parents.

A celle que j'aime le plus au monde, a celle qui ma tout donné, a toi ma grand mère, je te dédie ce travail en te disant que la vie me semble très courte pour exprimer mes remerciements.

A la mémoire de ma grande mère paternelle.

A toutes mes sœurs, beaux frères, nièces et neveu.

A mon frère Rafik que Dieu guide tes pas, te protège et te garde pour ta chère famille.

Que mon grand père et oncles et tantes trouvent en ces quelques lignes ma profonde reconnaissance et la preuve de ma grande gratitude pour leur aide, leur soutien, pour leur présence, pour leur amour, leur maison est toujours ouverte pour moi. Merci de m'avoir considéré comme l'une de vos filles.

A celui qui a ensoleiller ma vie, à mon mari : Slim, je te dédie ce travail en te disant que chaque route est plus courte et chaque poids plus léger quand tu es à mes côtés.

A toutes cousines et cousins.

A ma belle mère et a toutes ma belle famille.

A tout (es) mes ami(es) que j'aime, en particulier : Ghenima BOURKACHE, Nordine TAMANI, Hanafi MAINCEUR, Lila BOUSNINA et Lila IZA.

Rebiha

Introduction générale

I. Introduction

Les réseaux informatiques sont devenus des ressources vitales et déterministes pour le bon fonctionnement des entreprises. De plus, ces réseaux sont ouverts de fait qu'ils sont pour la plus part raccordés à l'Internet.

Cette merveilleuse ouverture qui permet de faciliter la communication, engendre malheureusement des risques importants dans le domaine de la sécurité informatique. Les utilisateurs de l'Internet ne sont pas forcements pleins de bonnes intentions, ils peuvent exploiter les vulnérabilités des réseaux et systèmes pour réaliser leurs attaques. Les conséquences de ces attaques peuvent être lourdes pour un particulier (pertes d'informations, ou pire encore vol d'informations, atteinte à la vie privée..) et pour une entreprise (perte du savoir faire, atteinte à l'image de marque, perte financière..). Pour cela, les administrateurs déploient des solutions de sécurité efficace capable de protéger le réseau de l'entreprise. Dans ce contexte, les IDS constituent une bonne alternative pour mieux protéger le réseau informatique.

Les IDS connaissent de nos jours un essor important et constituent un investissement des entreprises. Ils sont déployés dans des zones précises du réseau ou sur des machines particulières pour compléter le travail des pare-feu, Un pare-feu agit comme une première barrière externe pour repousser les pirates informatiques, tandis qu'un système de détection d'intrusion vise à repérer ceux qui auraient transpercé ce premier périmètre de défense.

Diverses méthodes de détections d'intrusions ont été proposées, elles sont basées principalement sur deux approches : l'approche comportementale et l'approche par scénarios. La première se base sur l'hypothèse que l'on peut définir un comportement normal de l'utilisateur et que toute déviation par rapport à celui-ci est potentiellement suspecte. La seconde s'appuie sur la connaissance des techniques employées par les attaquants : on en tire des scénarios d'attaques et on recherche dans les traces d'audit leurs éventuelles survenues.

II. problématique

Afin de remplir les objectifs des IDS, diverses méthodes de détections d'intrusions ont été proposées, parmi ces méthodes, nous citons la classification.

Plusieurs travaux ont été menés sur les algorithmes de classifications. Cependant, les algorithmes de classifications exploités jusqu'à présent dans le domaine de la sécurité possédant eux même des limites qui peuvent entacher la détection d'intrusions : un arbre de décision par exemple présente un très gros défaut dans le cas où des instances de l'ensemble de tests ne satisfont aucune règle de la base d'apprentissage.

Afin de remédier à ces défauts en particulier et de résoudre les limites des IDS en général, la détection d'intrusions doit s'orienter vers des nouvelles techniques de détection pour mieux assurer la sécurité de réseaux. Pour cela, nous proposons une nouvelle procédure de détection d'intrusions (qui n'a pas été exploitée à ce jour dans le domaine de la sécurité), qui consiste à utiliser des algorithmes de classification basés sur les fourmis artificielles.

Notre travail consiste à réaliser un IDS comportemental basé sur l'algorithme de classification non supervisé **AntClass** inspiré du tri collectif du couvain chez les fourmis.

III. Organisation

Ce mémoire est organisé comme suit :

Partie I: Cette partie présente l'état de l'art, elle est composée de deux chapitres :

- Le premier chapitre de cette partie, représente une introduction aux réseaux informatiques : les objectifs, le mode de fonctionnement .etc. Nous évoquons également dans la deuxième partie de ce chapitre un scénario d'attaques sur les réseaux informatiques et les différentes solutions pour remédier contre ces attaques.

- Dans le deuxième chapitre de cette partie, nous exposons les systèmes de détection d'intrusions : qualités, classifications, caractéristiques .etc. Nous soulignons également les limites actuelles des systèmes de détection d'intrusions réseaux.

Partie II : Cette partie présente notre solution : une architecture d'IDS comportementale qui utilise une méthode de classification qui inspirent du fonctionnement de colonies des fourmis, cette méthode permet une classification non supervisées (il s'agit de la méthode ***AntClass***). Cette partie est structurée en trois chapitres :

- Le premier chapitre de cette partie, expose la base d'apprentissage et de test appelé ***KDD*** que les méthodes vont utiliser pour faire une classification des attaques du système à surveiller.
- Le deuxième chapitre de cette partie , permet la description de la méthode de classification non supervisée *AntClass* et son application pour construire un IDS comportemental , en utilisant la base d'apprentissage et de test *KDD*.
- Le troisième chapitre de cette partie, présente les éléments de mise en œuvre de notre solution : implémentation,test et validation test de notre IDS à base de *AntClass*.

Partie III : Cette partie est constituée d'une conclusion générale et des perspectives pour continuer et améliorer ce travail.

Partie I :

État de l’art

Chapitre I : **Réseaux et techniques de protection contre les attaques réseaux.**

ChapitreII : **Systèmes de Détection d’Intrusions.**

CHAPITRE I.

Réseaux et techniques de protection contre les attaques réseaux.

Sommaire :

.

Partie I: Réseau informatique

I.1. Introduction

Les réseaux informatiques sont nés d'un besoin d'échanger des informations de manière simple et rapide entre les machines. Lorsqu' on travaillait sur une machine, toutes les informations nécessaires au travail étaient centralisées sur celle-ci. Pour des raisons de coûts et de performances, on a manipulé plusieurs machines. Les informations ont du alors être dupliquées sur les différentes machines du même site. Puis on a éprouvé le besoin d'échanger des informations entre des sites, d'où l'évolution du besoin d'échanges de l'information.

I.2. Définition [1]

Un réseau (Network) est un ensemble d'ordinateurs et périphériques interconnectés. Il permet de faire circuler des données informatiques et ainsi d'échanger du texte, des images, de la vidéo ou du son entre chaque équipement selon des règles et protocoles bien définis.

I.3. Objectifs d'un réseau [48][1]

Un réseau informatique permet:

- Le partage des fichiers et des périphériques, applications et imprimantes
- La communication entre personnes (grâce au courrier électronique, la discussion en direct,..)
- La communication entre processus (entre des machines industrielles)
- La garantie de l'unicité de l'information (bases de données)
- Le transfert des données
- Le jeu à plusieurs,…

I.4. Classification des réseaux [3]

On peut classifier les réseaux selon la distance qui sépare les ordinateurs en trois catégories :

- LAN (local area network)
- MAN (metropolitan area network)
- WAN (wide area network)

I.4.1. LAN

LAN signifie *Local Area Network* (en français *Réseau Local*). Il s'agit d'un ensemble d'ordinateurs appartenant à une même organisation et reliés entre eux dans une petite aire géographique par un réseau, souvent à l'aide d'une même technologie (la plus répandue étant Ethernet).

I.4.2. MAN

Les MAN (*Metropolitan Area Network*) interconnectent plusieurs LAN géographiquement proches (au maximum quelques dizaines de kms) à des débits importants. Ainsi un MAN permet à deux noeuds distants de communiquer comme s'ils faisaient partie d'un même réseau local.

I.4.3. WAN

Un WAN (*Wide Area Network ou réseau étendu*) interconnecte plusieurs LAN à travers de grandes distances géographiques. Les WAN fonctionnent grâce à des routeurs qui permettent de choisir le trajet le plus approprié pour atteindre un noeud du réseau. Le plus connu des WAN est Internet.

I.5. Fonctionnement d'un réseau

Pour assurer le bon fonctionnement d'un réseau, il faut réunir les supports physiques nécessaires et prévoir une bonne architecture logicielle et une normalisation de celle-ci s'impose. Deux familles d'architectures ont vu le jour : La première s'appelle ***le Modèle OSI***. La seconde est ***l'architecture TCP/IP***. Détaillons chacune d'elles : **[1]**

I.5.1. Le Modèle OSI (Open System Interconnexion) [1] [3] [48]

Ce modèle est une norme définie par *ISO* (International Standarlization Organisation). Fondé sur un principe énoncé par *JULES CESAR* « Diviser pour mieux régner».Ce modèle est composé de sept couches (elles seront détaillées ultérieurement) : ***couche physique, liaison de données, réseau, session, transport, présentation et application.***

Ce modèle permet la communication entre plusieurs réseaux hétérogènes, cette communication passe donc par un ensemble de couches empilées:

- Chaque couche a un rôle précis (conversion, routage, découpage, vérification etc.)
- Chaque couche dialogue avec la couche juste au dessus et celle juste au dessous : elle fournit des services à la couche dessus et utilise les servies de la couche dessous.
- Chaque couche encapsule les données venant de la couche dessus en y ajoutant ses propres informations avant de les passer à la couche dessous (opération inverse dans l'autre sens).
- Les données traversent les couches vers le bas quand elles sont envoyées et elles remontent les couches à la réception.

Voyons donc le rôle de chacune de ces couches : **[3]**

1. Couche physique : C'est le support de transmission lui-même : un fil de suivre, une fibre optique etc.

2. Couche Liaison de données : En charge d'encodes (ou moduler) les données pour qu'elles soient transportables par le couche physique et fournit également la détection d'erreur de transmission et la synchronisation.

3. Couche réseau : En charge du transport, de l'adressage et du routage des paquets.

4. Couche Transport : En charge de la liaison d'un bout à l'autre. Cette couche s'occupe de la fragmentation des données en petits paquets et vérifie éventuellement qu'elles ont été transmises correctement.

5. Couche Session : En charge d'établir et maintenir des sessions (c'est-à-dire débuter le dialogue entre machines, vérifier que l'autre machine est prête à communiquer, s'identifier, etc..).

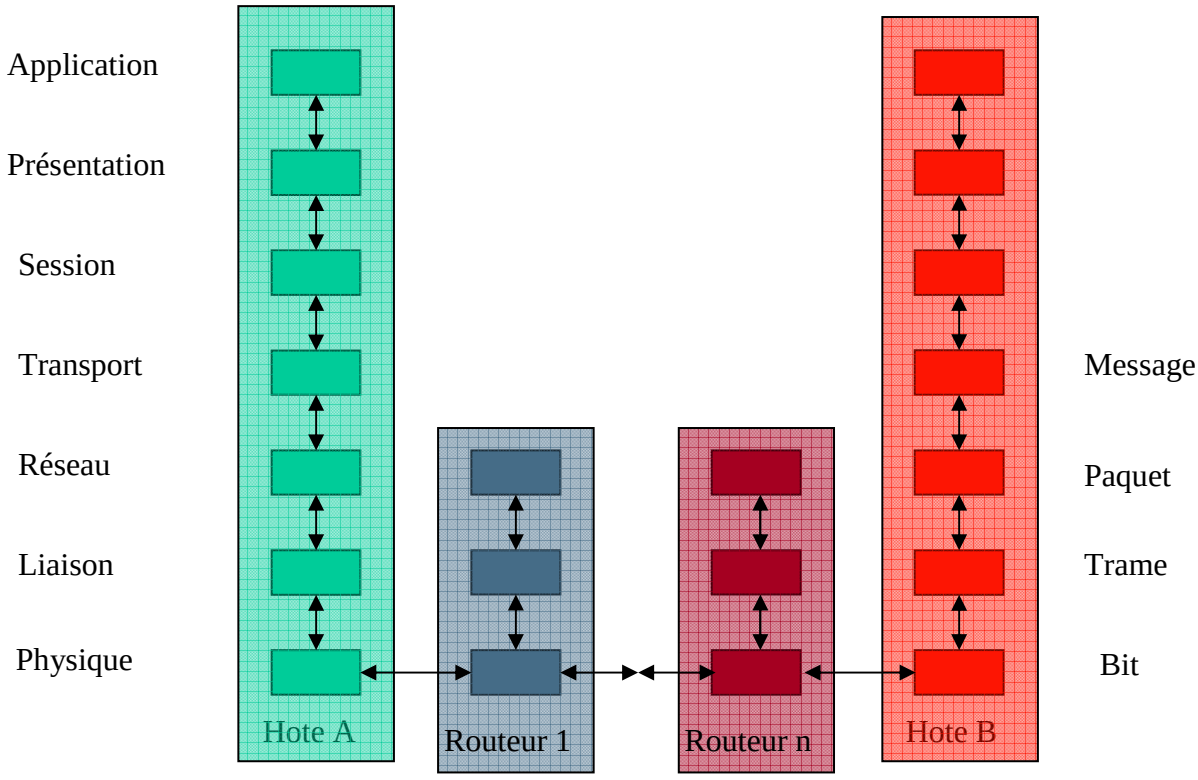


Figure I.1 : Modèle de référence OSI

.

6. Couche Présentation : Encharge de la représentation des données (de telle sorte qu'elle soit indépendante de microprocesseur ou du système d'exploitation par exemple) et éventuellement du chiffrement.

7. Couche application : Représente la couche la plus élevée du modèle OSI, elle utilise les services de la couche présentation (indirectement des autres couches) pour exécuter une application spécifique .l'application peut être un échange des courriers, transfert des fichiers ou toute autre application réseau.

Au niveau de chaque couche un ensemble de protocole est intégré. Un protocole réseau est un langage que vont utiliser toutes les machines d'un réseau pour communiquer entre elles.

HTTP, FTP, TCP, IP, ICMP, et la totalité des autres protocoles entrent dans le modèle OSI.

I.5.2. **Le Modèle TCP/IP [2] [3]**

Développé par l'armée américaine. Ils désigne en fait deux protocoles étroitement liés : un protocole de transport TCP (Transmission Control Protocol), et un protocole réseau IP (Internet Protocol).Le modèle TCP/IP est en fait une architecture réseau à quatre couches : ***couche hôte réseau, Internet, couche transport*** et ***application*** **.**Détaillons chacune de ces couches :

1. Couche hôte réseau : Cette couche semble regrouper les couches : physique et liaison de données du modèle OSI. Elle permet à un hôte d'envoyer des paquets IP sur le réseau

2. Couche Internet : Cette couche est la clé de voûte de l'architecture IP.Cette couche réalise l'interconnexion des réseaux (hétérogènes). Son rôle est de permettre l'injection des paquets dans n' importe quel réseau et l'acheminement de ces paquets indépendamment les uns des autres jusqu'à destination, les paquets peuvent arriver dans le désordre, le contrôle de l'ordre est la tâche des couches supérieures. L'implémentation officielle de cette couche est le protocole IP **[3]**.

3. Couche transport : Son rôle est le même que celui de la couche transport du modèle OSI. Officiellement, cette couche n'a que deux implémentations : le protocole *TCP* et le protocole *UDP* (User Datagram Protocol) **[3]**

4. Couche application : Contrairement au modèle OSI, c'est la couche immédiatement supérieure à la couche transport, tout simplement parce que les couches présentation et session sont apparues inutiles.

On s'est en effet aperçu avec l'usage que les logiciels réseaux n'utilisent que très rarement ces deux couches (Présentation et session), et finalement, le modèle OSI dépouillé de ces deux couches ressemble fortement au modèle TCP/IP.

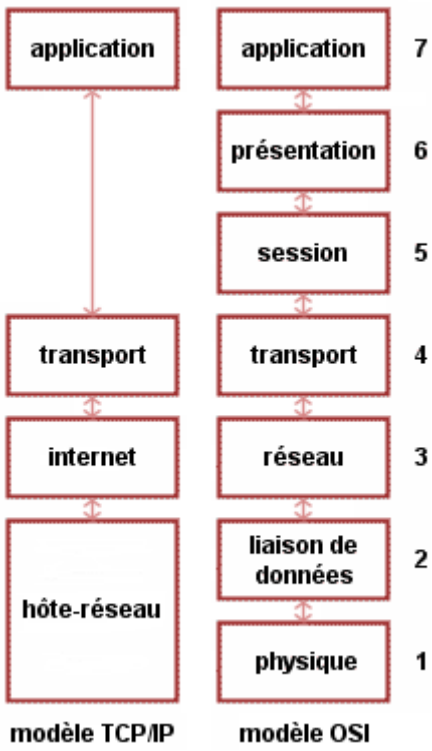


Figure I .2 : Modèle TCP et OSI

TCP/IP est le protocole utilisé dans le réseau Internet. L'implémentation de ce modèle engendre malheureusement des vulnérabilités dus au failles des langages de programmation utilisés dans l'implémentation (exp : langage C). Ces vulnérabilités peuvent être exportées par les attaquant pour réaliser leurs attaques, d'où le problème de la sécurité réseau.

Partie II: Sécurité réseau

II.1. Introduction

Les entreprises ouvrent leur système d'information à leurs partenaires et fournisseurs via internet. Cette merveilleuse ouverture, qui permet de faciliter la communication engendre malheureusement des risques importants dans le domaine de sécurité de l'entreprise.

Les ordinateurs connectés de l'entreprise ont des failles qui peuvent être exploitées par des **hackers**[1] pour réaliser ses attaques. Les conséquences de ces attaques peuvent être lourdes (perte d'information, vol d'information, perte financière, accès à des informations confidentielles, etc..). Par conséquent, il est important d'être conscient de ces menaces et de prendre des mesures adéquates afin de se protéger de ces attaques, pour cela, les administrateurs sécurisent de plus en plus leur système d'informations en utilisant diverses solutions comme : mot de passe, pare feu, la cryptographie, les scanners de vulnérabilités, antivirus, et les systèmes de détection d'intrusions. Nous détaillons dans la suite chacune de ces méthodes et nous soulignons leurs limites.

II.2. Objectifs de la sécurité informatique [5]

La sécurité informatique à plusieurs objectifs, bien sûr liés aux types de menaces ainsi qu'aux types de ressources, etc. Néanmoins, les points principaux sont les suivant : **[5]**

- Empêcher la divulgation non autorisée des données.
- Empêcher la modification non autorisée des données.
- Empêcher l'utilisation non autorisée des ressources réseaux ou informatique de façon générale.

II.3. Services Principaux de la sécurité réseau

Pour remédier aux failles et pour contrer les attaques, la sécurité informatique se base sur un certain nombre de services qui permettent de mettre en place une réponse appropriée à chaque menace. Les principaux services sont : **[6]**

[1] Le terme « hacker » est souvent utilisé pour designer un pirate informatique.

-La confidentialité : La confidentialité consiste à rendre l'information inintelligible à d'autres personnes que les seuls acteurs de la transaction.

-L'intégrité de données :Vérifier l'intégrité des données consiste à déterminer si les données n'ont pas été altérées durant la communication (de manière fortuite ou intentionnelle).

-La disponibilité : Permettant de maintenir le bon fonctionnement du système informatique.

-La non répudiation : Permettant de garantir qu'une transaction ne peut être niée.

-L'authentification : L'authentification consiste à assurer l'identité d'un utilisateur, c'est-à-dire de garantir à chacun des correspondants que son partenaire est bien celui qu'il croit être.

II.4. Objectifs des hackers

Les motivations des hackers (Selon les individus) peuvent être multiples. On y retrouve : **[7]**

- Vérification de la sécurité d'un système.
- Espionnage.
- L'attirance de l'interdit.
- Le désir d'argent (voler un système bancaire par exemple).
- Le besoin de renommées (impressionner des amis).
- L'envie de nuire.
- Pour apprendre.
- Etc.

II.5. Politique des hackers [8] [9]

La Meilleure façon de protéger son système informatique est de procéder de la même manière que les pirates, afin de cartographier les vulnérabilités du système : **[8] [9]**

II.5.1. Reconnaissance du système [17]

Avant qu'un hacker ne s'introduit dans le système informatique, il cherche dans un premier temps les failles c'est-à-dire, des vulnérabilités visibles à la sécurité du système : dans les protocoles, les systèmes d'exploitation, les applications, ou même le personnel d'une organisation. Pour cela, il utilise plusieurs moyens :

- **Reconnaissance passive [9] [17]**

Le hacker partage librement ses découvertes et évite la destruction intentionnelle des données. Une des attaques les plus répandues 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'exploitations utilisées.

- **Reconnaissance active [9]**

A ce niveau, l'attaquant ne se restreint pas à inspecter les données échangées entre les différents hôtes cependant, il initie lui même des connexions réseau pour tester le comportement des machines, il cherche des informations précises concernant les hôtes accessibles, l'emplacement des routeurs et des par feux. Parmi les techniques les plus utilisées pour acquérir ces informations, nous évoquons les utilitaires : **Ping [58]**, **TraceRoute [59]** et **Nmap [55].**

II.5.2. Exploitation du système [9]

Une fois le hacker a localisé les applications vulnérables, il exploite ensuite leurs faiblesses. L'intrus cherche à gagner un accès au réseau, cible en lançant diverses attaques. (Dans la suite nous détaillons quelques attaques).

II.5.3. Préservation d'accès [9]

Les attaquants installent des **portes dérobées**[2] pour pouvoir retourner facilement aux systèmes compromis. Par exemple, ils créent de nouveaux comptes et les utilisent lors des

[2] Porte dérobée : faille créé par le pirate.

prochains accès. Cette procédure est facilement détectable si un administrateur vérifie constamment l'intégrité des fichiers.

II.5.4. Effacement des traces [8] [9]

Une fois la porte dérobée est crée, l'attaquant cherche aussitôt à effacer ses traces. Il 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 événements sur des machines distinguées pour mieux protéger les fichiers de sécurité.

II.6. Différents types d'attaques

II.6.1. Les attaques réseaux [5][6][10] [8]

Les attaques réseaux les plus connues aujourd'hui sont :

- **Spoofing IP [31]**

Le spoofing IP est une technique permettant à un pirate d'envoyer à une machine des paquets semblant provenir d'une adresse IP autre que celle de la machine du pirate. Le spoofing IP n'est pas pour autant un changement d'adresse IP. Plus exactement, il s'agit d'une mascarade de l'adresse IP au niveau des paquets émis, c'est-à-dire une modification des paquets envoyés afin de faire croire au destinataire qu'ils proviennent d'une autre machine.

- **Spoofing ARP**

Le spoofing ARP est une technique qui modifie le cache ARP. Le cache ARP contient une association entre les adresses matérielles des machines et les adresses IP, l'objectif du pirate est de conserver son adresse matérielle, mais d'utiliser l'adresse IP d'un hôte approuvé. Ces informations sont simultanément envoyées vers la cible et vers le cache. A partir de cet instant, les paquets de la cible seront routés vers l'adresse matérielle du pirate.

- **Spoofing DNS**

Le système DNS (Domain Name System) a pour rôle de convertir un nom de domaine en son adresse IP et réciproquement, à savoir : convertir une adresse IP en un nom de domaine. Cette attaque consiste à faire parvenir de fausses réponses aux requêtes DNS émises par une victime. Il existe deux types de méthode: **[5]**

- **DNS ID spoofing** L'attaquant essaie de répondre à un client en attente d'une réponse d'un serveur DNS, avec une fausse réponse et avant que le serveur DNS ne réponde.
- **DNS Cache Poisoning** L'attaquant essaie d'empoisonner le cache (table de correspondance IP- nom _machine) du serveur DNS.

Désynchronisation TCP (Hijacking) [48]

Le protocole TCP permet d'assurer le transfert des données de façon fiable, bien qu'il utilise le protocole IP (qui n'intègre aucun contrôle de livraison de datagramme) grâce à un système d'accusés de réception (ACK) permettant au client et au serveur de s'assurer de la bonne réception mutuelle des données.

Lors de l'émission d'un segment, un numéro d'ordre (appelé aussi numéro de séquence) est associé, et un échange de segments contenant des champs particuliers (appelés *drapeaux*, en anglais *flags*) permet de synchroniser le client et le serveur. Ce dialogue (appelé *poignée de mains en trois temps*) permet d'initier la communication, il se déroule en trois temps, comme sa dénomination l'indique:

- Dans un premier temps, la machine émettrice (le client) transmet un segment dont le drapeau SYN est à 1 (pour signaler qu'il s'agit d'un segment de synchronisation), avec un numéro d'ordre C, que l'on appelle numéro d'ordre initial du client
- Dans un second temps la machine réceptrice (le serveur) reçoit le segment initial provenant du client, puis lui envoie un accusé de réception, c'est-à-dire un segment dont le drapeau ACK est à 1 (accusé de réception) et le drapeau SYN est à 1 (car il s'agit là encore d'une synchronisation). Ce segment contient un numéro de séquence S. Le champ le plus important de ce segment est le champ accusé de réception (ACK) qui contient le numéro d'ordre initial du client, incrémenté de 1

- Enfin, le client transmet au serveur un accusé de réception, c'est-à-dire un segment dont le drapeau ACK est à 1, et dont le drapeau SYN est à zéro (il ne s'agit plus d'un segment de synchronisation). Son numéro d'ordre est incrémenté et le numéro d'accusé de réception représente le numéro de séquence initial du serveur incrémenté de 1

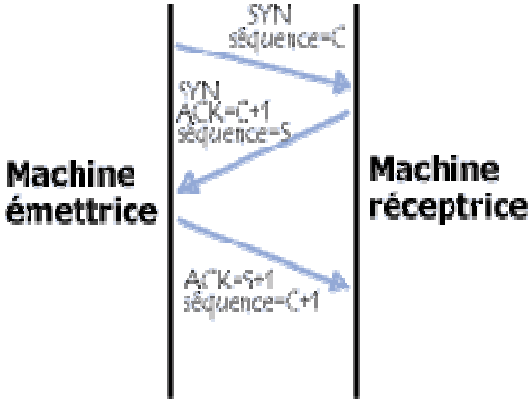


Figure I.3 : Synchronisation TCP **[48]**

L'attaquant peut rediriger le trafic TCP, pour cela il envoie des paquets malformés au client avec une adresse IP correspondant à celle du serveur en y plaçant des mauvais numéros de séquences, le client va croire qu'il a perdu la connexion et stoppera ses échanges avec le serveur. Mais si l'attaquant envoie les bons numéros de séquences au serveur, il récupèrera la connexion pour lui.

II.6.2. Les attaques applicatives [5][6] [8]

- **Injection SQL [5]**

Les attaques par **injection de commandes SQL** sont des attaques visant les sites web s'appuyant sur des bases de données relationnelles.

Dans ce type de sites, des paramètres sont passés à la base de données sous forme d'une requête SQL. Ainsi, si le concepteur n'effectue aucun contrôle sur les paramètres passés dans la requête SQL, il est possible à un pirate de modifier la requête afin d'accéder à l'ensemble de la base de données, voire à en modifier le contenu.

Le principe de cette attaque consiste à injecter du code supplémentaire au sein des requêtes a fin de leurrer l'interpréteur SQL.

Par exemple contourner les mécanismes d'authentification via les entrées imprévues.
Considérons par exemple la requête SQL (II.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 (II.1) se transforme en (II.2) **[5]** et sera toujours vérifiée si l'utilisateur $varNom existe dans la table utilisateur.

select * from utilisateur where nom = $varNom and mot2passe = $varPasse (II.1)
select * from utilisateur where nom = $varNom and True (II.2)

L'attaquant peut créer son propre compte, en utilisant le nom d'un utilisateur légitime requête (II.3) **[5]**

SELECT * from client where mon='joe ;insert into utilisateurs values('mon_login,'mon_password')' (II.3)

- **Les bugs [5]**

Tout logiciel comporte des bogues dont certains représentent des trous de sécurité ou des anomalies qui permettent de violer le système sur lequel tourne le programme. Si c'est un programme d'application réseau, ces trous peuvent être exploités à distance via Internet. Les plus connus de ces bugs et les plus intéressants, en ce qui concerne leur exploitation sont les buffers overflows.

Buffer overflow : consiste à mettre plus d'informations (et surtout d'autres informations) en mémoire que celle-ci n'est disposée à en recevoir.

Conséquences :

- Le débordement du buffer peut écraser l'adresse de retour au programme appelant : plantage (attaque de type déni de service).

- L’adresse de retour peut être remplacée par l’adresse d’un code malicieux.

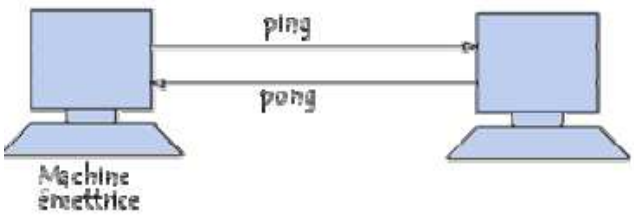
II.6.3. Les attaques par déni de service [5][13]

Les attaques par déni de service (souvent abrégé ***DOS***, en anglais Denial of service) consistent à paralyser temporairement (rendre inactif pendant un temps donné) des serveurs afin qu’ils ne puissent être utilisés et consultés. Le but d’une telle attaque n’est pas de récupérer ou d’altérer des données, mais de nuire à des sociétés dont l’activité repose sur un système d’information. En l’empêchant de fonctionner.

- **La technique dite du smurf [48]**

La technique du *smurf* est basée sur l’utilisation de serveurs broadcast pour paralyser un réseau. Un serveur broadcast est un serveur capable de dupliquer un message et de l’envoyer à toutes les machines présentes sur le même réseau que lui. Le scénario d’une attaque est le suivant :

La machine attaquante envoie un **ping [58]** à un (ou plusieurs) serveurs broadcast en falsifiant sa propre adresse IP (l’adresse à laquelle le serveur devrait théoriquement répondre par un pong) et en fournissant l’adresse IP de la machine cible. Lorsque le serveur broadcast va dispatcher le ping sur tout le réseau, toutes les machines du réseau vont répondre par un pong, que le serveur broadcast va rediriger vers la machine cible. Ainsi lorsque la machine attaquante adresse le ping à plusieurs serveurs broadcast situés sur des réseaux différents, l’ensemble des réponses de tous les ordinateurs des différents réseaux vont être reroutées sur la machine cible.



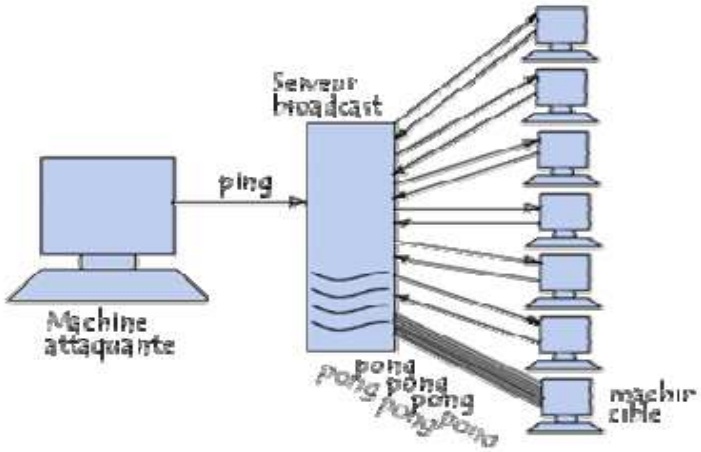


Figure I.4 : Exemple d'attaque **[48]**

- **SYN FLOOD [6]**

Son objectif est de rendre indisponible un service TCP offert sur une machine. Le principe de cette attaque est de créer des connexions TCP semi-ouvertes sur la machine cible afin de remplir la file d'attente où sont stockées les demandes d'ouverture de connexions. L'attaquant envoie un grand nombre de requêtes SYN à la machine cible et remplace son adresse source avec l'adresse d'une machine indisponible ou inexistante afin que les réponses SYN/ACK ne soient jamais reçue et que donc les messages ACK ne soient jamais générés, ce qui signifie que la file d'attente restera pleine. Les conséquences de cette attaque sont que toutes les requêtes arrivant sur le port TCP cible seront ignorées et de ce fait le service fourni sur ce port sera indisponible. Dans certains cas, la machine peut aussi devenir indisponible.

- **Fragmentation [5]**

L'attaquant sature la connexion en envoyant des fragmentations déclanchant des exceptions (faille de la pile TCP/IP de *Windows* 95 et 98).

II.6.4. Les attaques virales [5]

Il existe principalement quatre types de menaces distinctes :

- **Virus** : Se reproduisent en infectant le corps de programmes hôtes
- **Vers** : Le vers se duplique et se propage à travers le réseau, par courrier électronique par exemple.
- **Chevaux de Troie** : Exécutent des tâches malignes en se cachant dans un programme sain. Il peut par exemple voler des mots de passe, copier des données, ou exécuter tout autre action nuisible.

Trappes (portes dérobées) : Permet à un utilisateur externe de prendre le contrôle d'une application par des moyens détournés.

II.7. Outils de sécurité

Nous avons constaté que les attaquants disposent de plusieurs moyens pour réussir leurs attaques. 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 pares feux, la cryptographie, le mot de passe, 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.

II.7.1. Cryptographie, Signature électronique et Certificat [14] [15] [16]

-Cryptographie [16]

Le mot cryptographie est un terme générique désignant l'ensemble de techniques permettant de chiffrer des messages. Chiffrer un message consiste à le transformer au moyen d'un algorithme mathématique afin de le rendre inintelligible, sauf pour celui qui possède le moyen (une clé) de le déchiffrer. L'encryption des informations électroniques transitent par le réseau, est utilisée pour assurer la confidentialité et l'authenticité des transactions.
Le chiffrement se fait généralement à l'aide d'une clé de chiffrement, le déchiffrement nécessite quant à lui une clé de déchiffrement. Nous distinguons deux types de clés : [16]

- **Les clés symétriques :** Il s'agit de clés utilisées pour le chiffrement ainsi pour le déchiffrement. Nous parlons alors de chiffrement symétrique ou de chiffrement à clé secrète.

Le Data Encryption Standard (DES) est un exemple de crypto système à clé secrète.

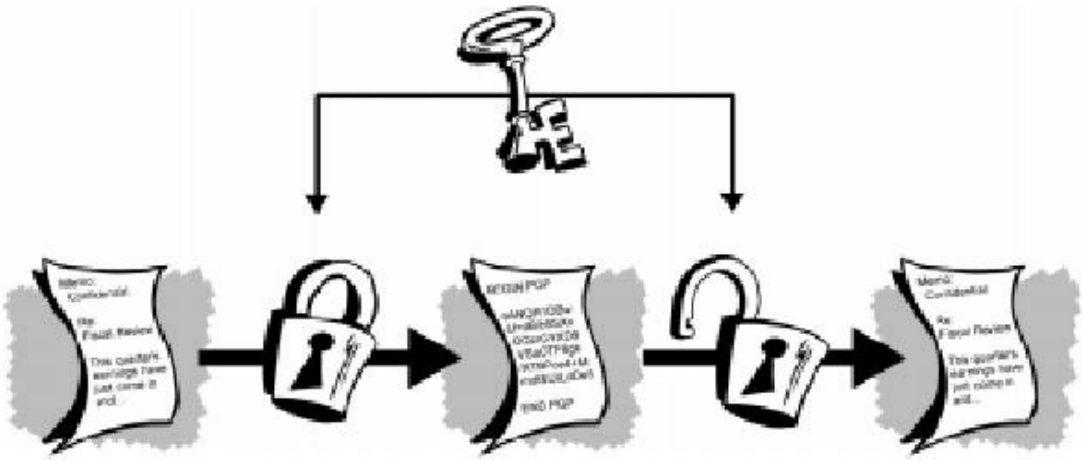


Figure I.5 *:* Les clés symétriques **[16]**

- **Les clés asymétriques** : Il s'agit de clés utilisées dans le cas du chiffrement asymétrique (aussi appelé chiffrement à clé publique). Dans ce cas nous utilisons une paire de clé, une clé publique qui chiffre les données, et une clé privée correspondante, aussi appelée clé secrète, qui sera utilisée pour le déchiffrement. Algorithme de Signature Digitale (DSA) est un exemple d'algorithme a clé publique.

Les attaquants peuvent construire (nous l'appelons aussi cryptanalyse) un message chiffré en clair à l'aide des méthodes mathématiques. Ainsi, tout crypto système doit nécessairement être résistant aux méthodes de cryptanalyse.

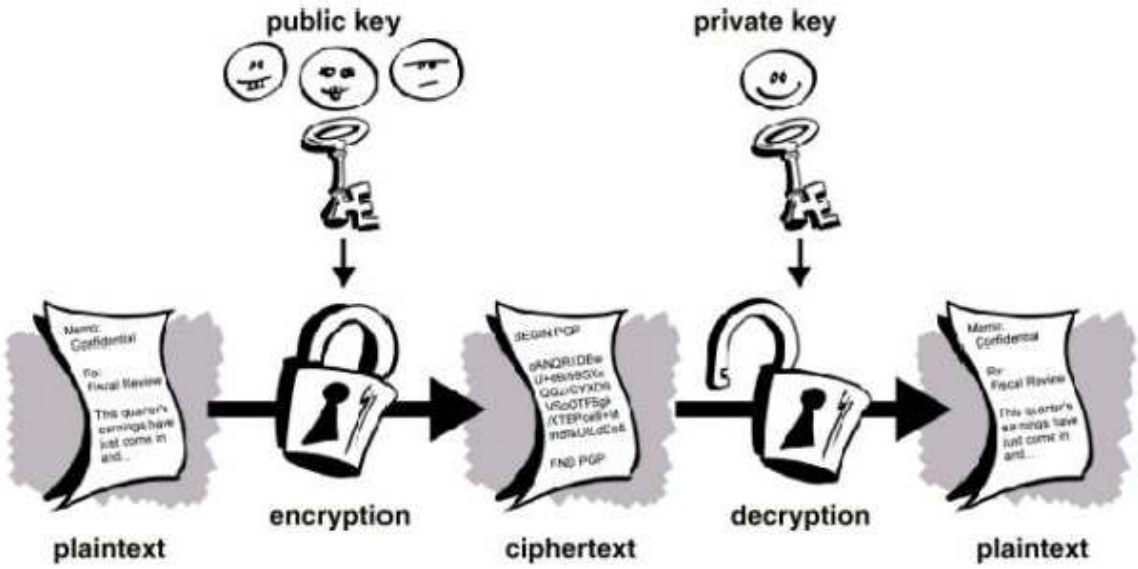


Figure I.6 *:* Les clés asymétriques **[16]**

-Signature électronique [14][16]

Signature électronique est un code digital permet à la personne qui reçoit une information de contrôler l'authenticité de son origine , et également de vérifier que l'information en question est intacte. Aussi, les signatures électroniques permettent l'authentification et le contrôle de l'intégrité et également la non répudiation.

-Certificat [15]

Certificat est Document électronique, carte d'identité émis par une autorité de certification. Il valide l'identité des interlocuteurs d'une transaction électronique, associe une identité à une clé publique d'encryptions et fournit des informations de gestion sur le certificat et le détenteur.

II.7.2. Mots de passes

Une personne peut être authentifiée par une combinaison d'une identification et d'un mot de passe, (code secret personnel).Le mot de passe doit posséder certaines

caractéristiques qui sont : non trivial, difficile à deviner, régulièrement modifié. Cependant si l'attaquant accède au fichier de mot de passe, il pourra s'introduire dans le système sécurisé.

II.7.3. Firewall [22]

Un Firewall est un assemblage matériel (ordinateur) et des logiciels installés sur celui ci dont l'objectif principal est de protéger le réseau interne contre les accès et actions non autorisés en provenance de l'extérieur, en contrôlant le trafic entrant et sortant.

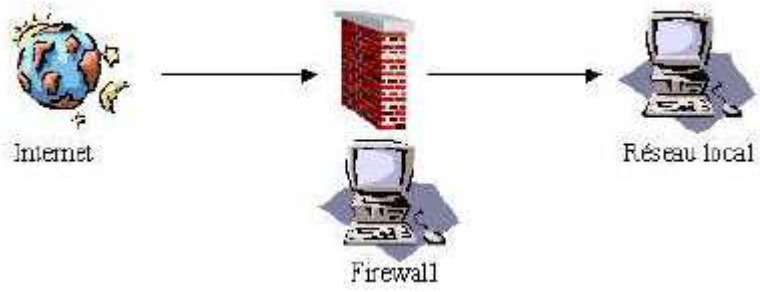


Figure I.7 *:* Placement d'un firewall [22]

Il existe plusieurs types de techniques de firewall : **[31]**

- La technique de filtrage des paquets : chaque paquet d'informations entrant ou sortant est accepté ou rejeté selon des règles établies par l'utilisateur.
- La technique des serveurs Proxy : qui empêche l'extérieur de connaître les adresses internes du réseau.
- La technique des passerelles : qui fournissent des systèmes de sécurité pour établir des connexions TCP/IP entre l'extérieur et l'intérieur ou pour certains services comme *FTP* et *TELNET.*

Les inconvénients d'un firewall sont **[22]** :

- Ne couvre pas tous les risques de sécurité. Par exemple il n'assure pas la confidentialité des informations, n'authentifie pas l'origine des informations, ne vérifie pas l'intégrité des informations, ne protège pas contre les attaques internes.
- Une très forte configuration de firewall augmente la sécurité mais peut alerter le fonctionnement du réseau.
- Le pirate peut détruire le système et ainsi permettre l'accès à tous les individus .C'est en général ce qui se passe.
- l'attaquant peut contourner le firewall

II.7.4. Scanners de vulnérabilités [9]

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 **[19]**, Whisker **[20]** et Saint **[21]**.

Cependant les scanners présentent quelques limites qui peuvent être résumées en trois 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.

II.7.5. Fichiers historiques [6]

La consultation régulière des fichiers historiques constitués doit notamment permettre de vérifier les anomalies dans le trafic des transactions. Par exemple, les massages répétitifs en

provenance d'une même adresse extérieure et rejetés par le Firewall peuvent être un signe d'essai d'intrusion. Cependant, ces fichiers peuvent être consultés par les hackers afin d'effacer leurs traces. Pour cela, les administrateurs sauvegardent des copies de ces fichiers (back up) dans des endroits sécurisés et sur des machines différentes.

II.7.6. Réseau Privé Virtuel [22]

Un réseau VPN (*Virtuel Privat Network*) est un service qui permet d'établir des connexions sécurisées privées (c'est-à-dire faire un réseau privé) sur un réseau publique comme Internet.

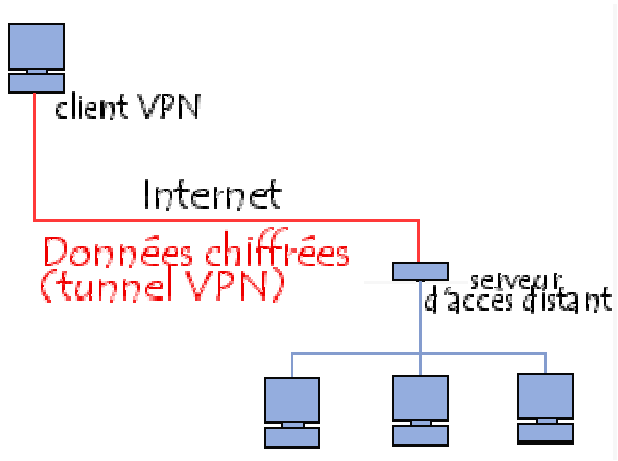


Figure I.8 : VPN **[22]**

VPN repose sur un protocole appelé protocole de ***tunneling*** **[22]** , le principe de tunneling consiste à construire un chemin virtuel après avoir identifié l'émetteur et le destinataire. Par la suite, la source chiffre les données et les achemine en empruntant ce chemin virtuel. Il existe trois types standards d'utilisation des VPN : **[22]**

-Le VPN d'accès : permet à des utilisateurs itinérants d'accéder au réseau privé.

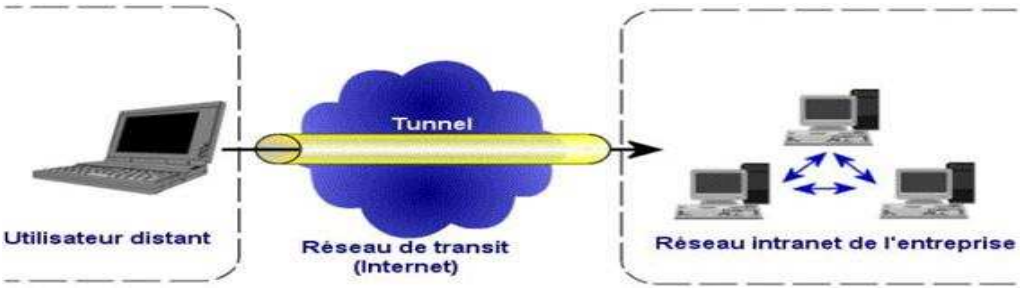


Figure I.9 : VPN d'accès **[22]**

-L'intranet VPN : est utilisé pour relier au moins deux intranet entre eux. Ce type de réseau est particulièrement utile au sein d'une entreprise possédant plusieurs sites distants.

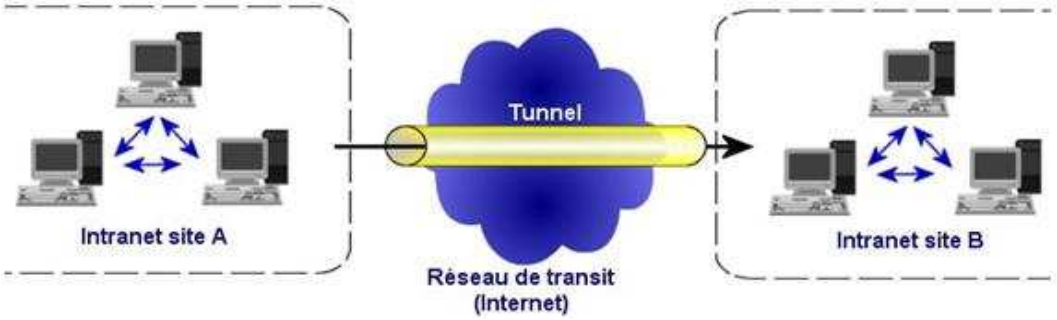


Figure I.10 : VPN intranet **[22]**

-L'extranet VPN : une entreprise peut utiliser le VPN pour communiquer avec ses clients et ses partenaires. Elle ouvre alors son réseau local à ces derniers. Dans Ce cadre, il est fondamental que l'administrateur du VPN puisse tracer les clients sur le réseau et gérer les droits de chacun sur celui-ci.

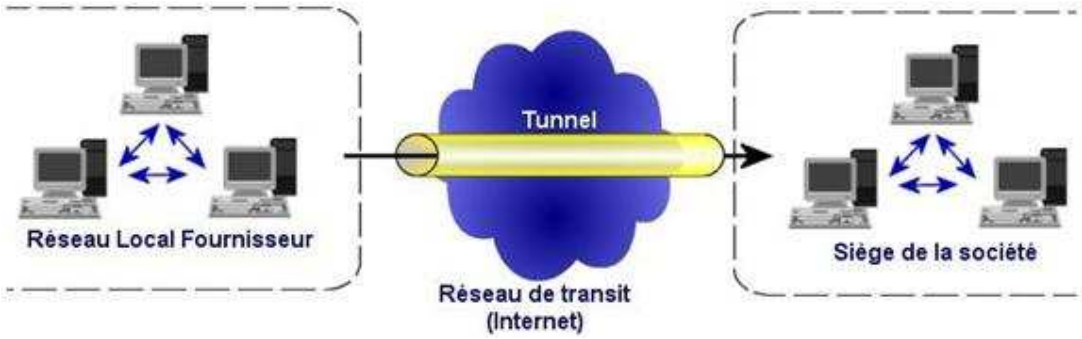


Figure I.11 : VPN extranet **[22]**

II.7.8. Pot de miel [9] [18]

Les pots de miels sont des systèmes qui simulent plusieurs services réseaux pour leurrer des intrus en exposant des vulnérabilités connues délibérément.

Un attaquant pense que ces services vulnérables sont actifs et qu'il peut les utiliser pour s'introduire dans le réseau. Il s'y colle pendant un certain temps .Pendant ce temps l'administrateur enregistre les activités de l'intrus pour découvrir ses actions et ses techniques.

Une fois ces techniques sont connues. L'administrateur emploie ces informations plus tard pour durcir la sécurité sur les serveurs réels.

II.7.9. Systèmes de détection d'intrusions

- **Contexte**

Le concept de système de détection d'intrusions a été introduit en 1980 par *Anderson* **[24]**. Mais le sujet n'a pas eu beaucoup de succès. Il a fallu attendre la publication d'un modèle de détection d'intrusions par *Denning* **[25]** en 1987 pour marquer réellement le départ du domaine. En 1988, il existait au moins trois prototypes. **[25]**

La recherche dans le domaine s'est ensuite développée, le nombre de prototypes s'est énormément accru. Le gouvernement des États-Unis a investi des millions de dollars dans ce type de recherches dans le but d'accroître la sécurité de ses machines. **[25]**

- **Intrusion**

Une intrusion est toute utilisation d'un système informatique à des fins autres que celles prévues, généralement dues à l'acquisition de privilèges de façon illégitime. **[26]**

Une intrusion dans un système informatique est aussi définie par *Heady* et *al*. **[27]** comme: ***« N'importe quel ensemble d'actions essayant de compromettre l'intégrité, la confidentialité ou l'accessibilité d'une ressource*** ».

Actuellement les ***IDS*** sont très populaires à cause de : **[23]**

-l'évolution continue des attaques.

-l'apparition de nouvelles attaques.

- la nécessite de détecter et réagir le plus vite possible aux attaques survenant dans le réseau.

Snort **[34]** est un exemple de ***IDS*** Open Source disponible au grand public basé sur l'approche par connaissance (nous détaillons cette approche dans le chapitre suivant).

Conclusion

Nous avons constaté dans ce chapitre, que la sécurité réseau est un point primordial. Les administrateurs déploient des solutions de sécurité efficaces, capables de protéger le réseau de l'entreprise. Dans ce contexte, les IDS constituent une bonne alternative pour mieux protéger le réseau informatique. Nous détaillons dans le chapitre II, les qualités d'un IDS, Nous discutons aussi sur ses deux approches à savoir l'approche comportementale et la détection par connaissance. Nous rappelons, que notre contribution consiste à proposer une architecture d'IDS comportementale, qui utilise une méthode de classification qui inspire du fonctionnement de colonies des fourmis.

CHAPITRE II.

Système de Détection d'Intrusions

Sommaire :

II.1. Introduction

Nous avons présenté au Chapitre I un scénario possible d'attaques sur le réseau de l'entreprise et les différents mécanismes de protections contre ces attaques. Malgré leur grand intérêt, ils présentent des lacunes dues à l'évolution des techniques utilisées par les hackers. Ainsi et dans l'impossibilité de détecter toutes les attaques, le système de détection d'intrusions tente de déceler les attaques qui passent inaperçues à travers les mécanismes de sécurité. Un IDS a pour fonction d'analyser en temps réel ou différé le trafic réseau et de détecter les différentes attaques lancées contre le réseau de l'entreprise.

Après la courte introduction des systèmes de détection d'intrusions présentée au Chapitre I, dans ce chapitre, nous détaillons les qualités requises d'un IDS, son utilité , le critère de choix et un état de l'art des approches proposées dans la littérature.

II.2. Système de détection d'intrusions [7]

Système de détection d'intrusions est un équipement permettant de surveiller l'activité d'un réseau ou d'un hôte donné, afin de détecter toute tentative d'intrusion et éventuellement de réagir à cette tentative.

H.Debar **[28]** simplifie le système de détection d'intrusions dans un détecteur qui analyse les informations en provenance du système surveillé (voir figure II.1). **[4]**

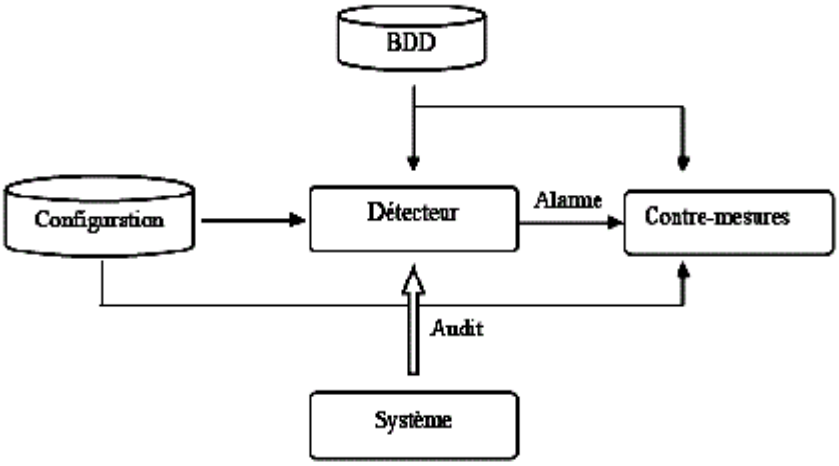


Figure II.1 : Modèle simplifié d'un système de détection d'intrusions **[4]**

Le détecteur analyse trois types d'informations : les informations de long terme relatives aux techniques utilisées dans la détection (Base de données de signatures), les informations de configuration qui déterminent l'état courant du système, et les informations d'audit (Section II.3) qui décrivent les événements survenus dans le système.

II.3. Fichier historique [29]

Fichier historique permet d'enregistrer tout ou une partie des actions effectuées sur le système. L'analyse de ses informations permet de détecter d'éventuelles intrusions. Les systèmes d'exploitation génèrent des fichiers historiques, certaines applications aussi. Les différents évènements du système sont enregistrés dans un journal, qui devra être analysé fréquemment, voire en permanence. Sur les réseaux, il est indispensable de disposer d'une base de temps commune pour estampiller les évènements.

Voici les types d'informations à collecter sur les systèmes pour permettre la détection d'intrusions : On y trouve les informations sur les accès au système (qui a accédé, quand et comment), les informations sur l'usage fait du système (utilisation du processeur, de la mémoire ou des entrées/sorties) et les informations sur l'usage fait des fichiers. Le fichier historique doit également permettre d'obtenir des informations relatives à chaque application (le lancement ou l'arrêt des différents modules, les variables d'entrée et de sortie et les

différentes commandes exécutées), Les informations sur les violations éventuelles de la sécurité (tentatives de commandes non autorisées) ainsi que les informations statistiques sur le système seront elles aussi nécessaires.

Notons que ces nombreuses informations occupent beaucoup de place et sont très longues à analyser. Ces informations devront être, au moins pour un temps, stockées quelque part avant d'être analysées par le système de détection d'intrusions.

II.4. Caractéristiques d'un système de détection d'intrusions [9]

Les caractéristiques suivantes sont souhaitables dans un IDS :

- fonctionner en permanence avec une supervision manuelle minimale.
- être tolérant aux pannes dans le sens où il doit récupérer après une défaillance ou une réinitialisation de la machine.
- résister aux tentatives de corruption, c'est-à-dire, il doit pouvoir détecter s'il a subit lui même une modification indésirable.
- utiliser un minimum de ressources du système sous surveillance.
- être facilement configurable pour implémenter une politique de sécurité spécifique d'un réseau.
- s'adapter au cours du temps aux changements du système surveillé et du comportement des utilisateurs.
- être difficile à tromper.

Comme la taille des réseaux a tendance à croître, on peut ajouter les caractéristiques :

- être scalable.
- être robuste, c'est-à-dire l'arrêt d'un composant ne doit pas entraîner une défaillance totale.

II.5. Emplacement d'un système de détection d'intrusions [22]

Pour protéger le réseau contre les attaques externes, le meilleur emplacement pour un système de détection d'intrusion, peut être juste après le routeur ou le Firewall. Cependant, un IDS peut être placé dans chaque segment de réseau, pour détecter les menaces internes. (FigureII.2) **[22]**

II.6. Classification des systèmes de détection d'intrusions [28]

Nous pouvons classifier les systèmes de détections d'intrusions selon cinq critères (cités ci-dessous) qui ne sont pas forcement mutuellement exclusif (figure II.3) :

a) Méthode de détection utilisée.
b) Mode de fonctionnement des mécanismes de détections.
c) Sources de données à analyser.
d) Réponse aux attaques.
e) Fréquence d'utilisation.

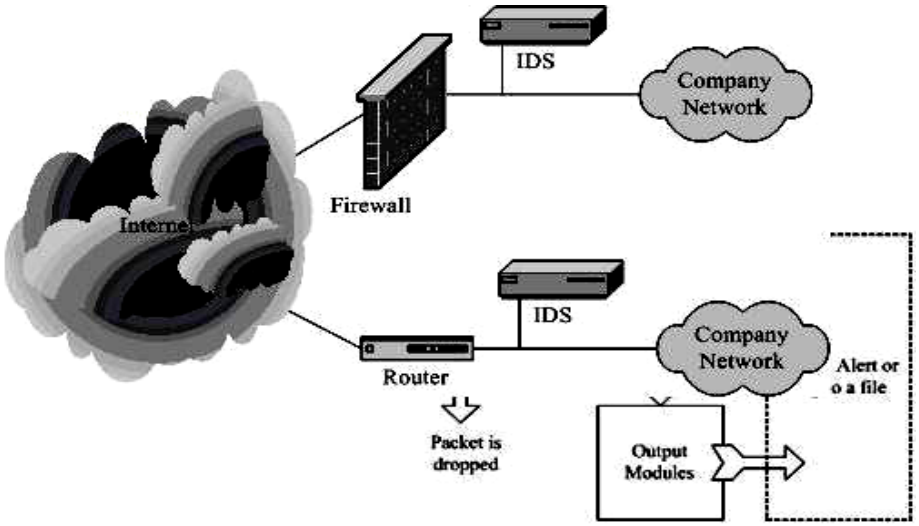
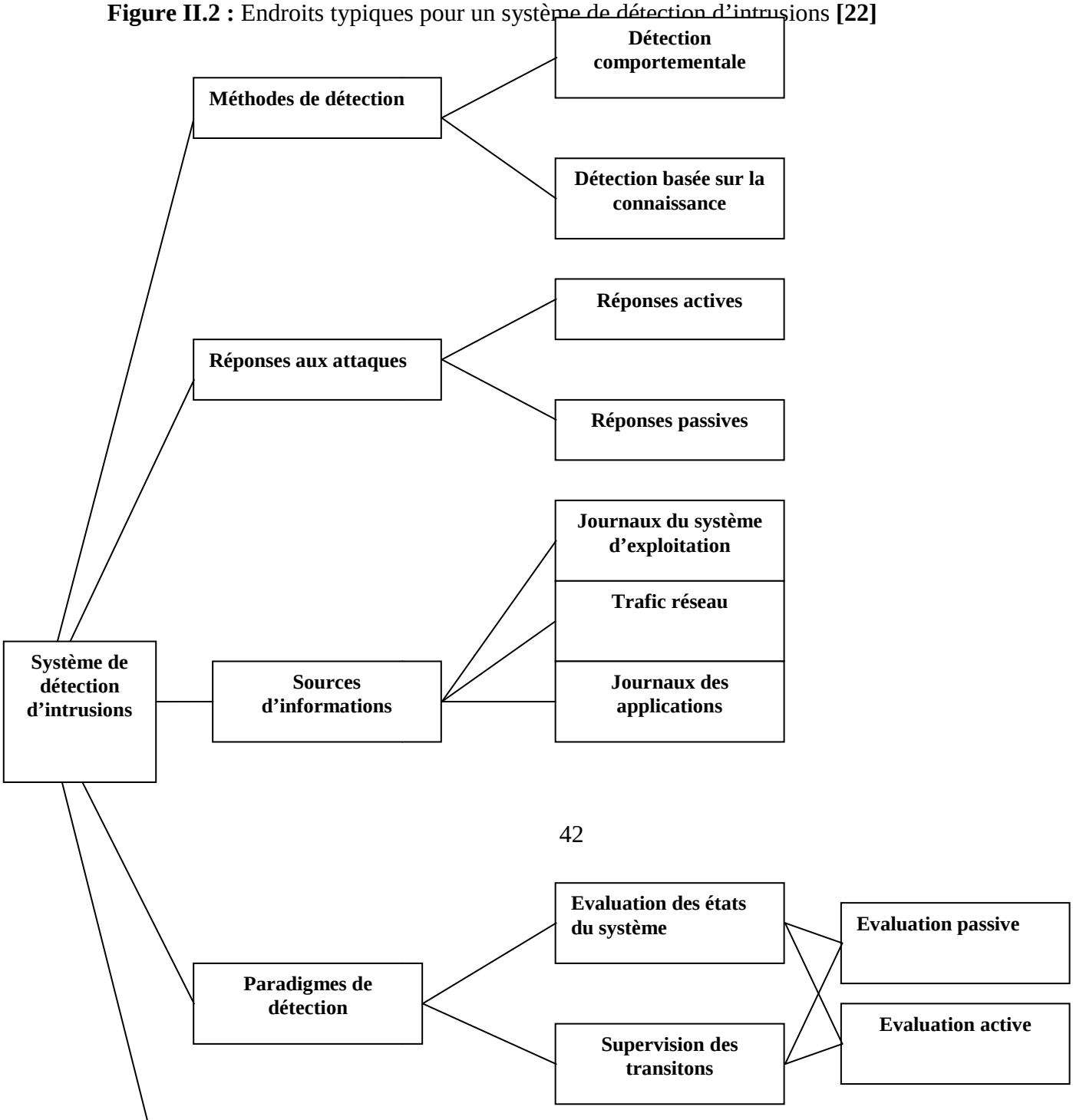


Figure II.2 : Endroits typiques pour un système de détection d'intrusions **[22]**



FigureII.3 : Classification des systèmes de détection d'intrusions **[28]**

II.6.1. Méthodes de détection [12] [32] [24]

Il existe deux grandes catégories de méthodes de détection explorées dans la littérature : 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 événement suspect. La deuxième approche, appelée détection par connaissances, recherche explicitement les signatures des attaques connues dans les fichiers de sécurité et le trafic réseau.

II.6.1.1. L'approche comportementale [60]

Une approche, proposée par *Anderson* **[24]** puis reprise et étendue par *Denning* **[25]**, consiste à utiliser des méthodes basées sur l'hypothèse selon laquelle l'exploitation d'une vulnérabilité du système implique un usage anormal de celui-ci. **[33]**

La mise en œuvre des IDS comportementaux comprend toujours une phase d'apprentissage au cours de laquelle ils vont découvrir le fonctionnement normal des éléments surveillés. Une fois cet apprentissage effectué, ces IDS signaleront les divergences

par rapport au fonctionnement de référence. Les modèles comportementaux peuvent être élaborés à partir d'analyses statistiques ou de techniques proches de l'intelligence artificielle

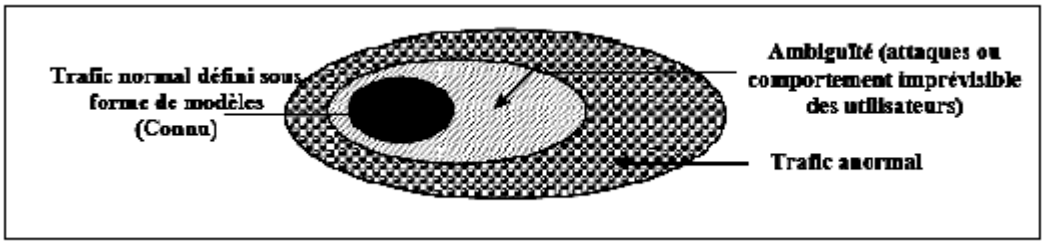


Figure II.4 : Détection d'anomalies **[13]**

Cette approche possède les avantages suivants : **[33] [13]**

- la détection d'anomalie permet de détecter un comportement non usuel et ainsi offre la possibilité de trouver des symptômes d'une attaque sans en connaître les détails.
- peut permettre de produire de l'information qui peut être utilisée pour définir des nouvelles signatures utilisables pour les systèmes à signatures.
- Pas besoin d'une base d'attaque.

Cependant, l'approche comportementale possède les inconvénients suivants **: [13] [33]**

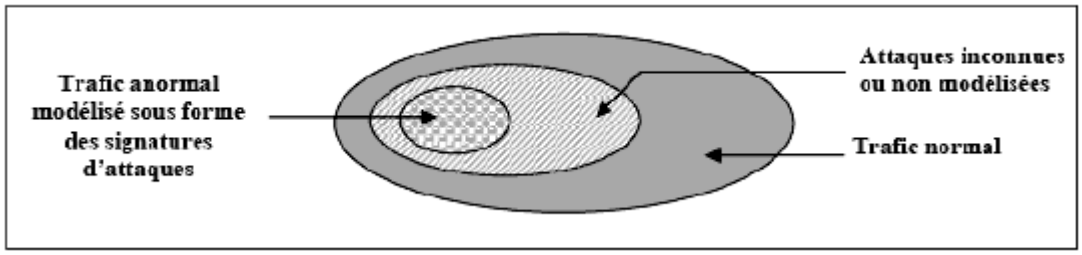
- produit une quantité énorme de fausses alertes à cause du caractère imprévisible des utilisateurs et des réseaux.
- demande une intense phase d'apprentissage pour caractériser la normalité des comportements.

- utilisateur pouvant changer lentement de comportement dans le but d'habituer le système à un comportement intrusif (faux négatifs).
- choix délicat des mesures à retenir pour un système cible donné.
- Pour un utilisateur au comportement erratique, toute activité est normale.

II.6.1.2. L'approche par scénarios [26] [24]

Cette approche consiste à rechercher, dans les fichiers de sécurité et le trafic réseau, les empreintes (ou signatures) d'attaques connues, à l'instar des anti-virus.

Une signature est habituellement définie comme une séquence d'événements et de conditions relatant une tentative d'intrusion. La reconnaissance est alors basée sur le concept de *pattern matching* (analyse de chaînes de caractères présente dans le paquet, à la recherche de correspondance au sein d'une base de connaissance). Si une attaque est détectée, une alarme peut être remontée.



FigureII.5 : Détection d'abus **[13]**

Cette approche possède les avantages suivants : **[41] [22]**

- censée être très puissante pour reconnaître une attaque sans générer trop de fausses alertes.
- capable de diagnostiquer rapidement l'utilisation d'une attaque ou outil d'attaque.
- ce type de détecteur reste assez facile à mettre en oeuvre, ne nécessitant pas de phase d'apprentissage (ce qui élimine le risque de sur apprentissage ou de déformation volontaire du profil).

Cependant, Ce type de détecteur est purement réactif, il ne peut détecter que les attaques dont il possède la signature, ce qui implique un taux plus élevé de faux négatifs. De ce fait, il nécessite des mises à jour quotidiennes. De plus, ce système de détection est aussi bon que la base de signatures.

Approche comportementale ou approche par scénarios ? **[13]**

Nous avons constaté, que chacune de ces deux approches présente des avantages et des inconvénients. Il semble donc indispensable d'hybrider l'approche comportementale avec l'approche par scénarios de manière à profiter des avantages de l'une et de l'autre.

Les modèles comportementaux sont de plus en plus intégrés à des IDS initialement basés sur une bibliothèque de signatures. En effet, certains éditeurs ont déjà fait le choix de compléter leur base de signature, par un modèle comportemental basique permettant de signaler des événements non identifiables.

II.6.2. Réponses des IDS [24] [9]

Lors de la détection d'une attaque, un système de détection d'intrusions, peut adopter plusieurs comportements. Les IDS peuvent émettre des réponses actives qui influent directement sur la source d'attaque, par exemple un IDS peut réagir en réparamétrant un par- feu, pour mettre en place des règles de blocage temporaire de certains flux réseau anormaux, comme ils peuvent se restreindre à des réponses passives en diffusant une alerte identifiant l'attaque détectée. Une liste des réponses actives et passives est présentée dans le tableau II.1 : **[9]**

Dans la pratique, peu d'administrateur de sécurité envisage concrètement de mettre en place, des contres mesure automatique (réponse active). Il laisse au lecteur le soin de deviner la dénomination d'une réponse active, après détection d'une attaque.

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 service les ports et les services attaquées
	Avertir l'utilisateur
	Tracer l'origine de la connexion
	Forcer une nouvelle authentification
	Restreindre les activités d'un utilisateur

Tableau II.1. Réponses aux attaques des systèmes de détection d'intrusions **[9]**

II.6.3. Sources de données à analyser [9][60][61]

Les IDS s'appuient généralement sur des sources de données différentes. Certains IDS dit ***IDS réseau*** ou ***NIDS*** (Network IDS) examinent les paquets transportés par le réseau, et se déploient généralement dans des endroits précis du réseau par exemple, dans la zone démilitarisée ou juste avant ou / et après un Firewall. D'autre dit ***IDS Système*** ou ***HIDS*** (Host IDS) examinent les données des journaux de sécurité établis par le système d'exploitation et les applications qui tournent sur ces machines .Ces IDS sont déployés directement sur les hôtes du réseau.

II.6.3.1 NIDS (*Network Based Intrusion Detection System*) [11]

Le rôle essentiel d'un NIDS est l'analyse et l'interprétation des paquets circulant sur un réseau. L'implantation d'un NIDS sur un réseau se fait de la façon suivante (figure II.6) : des capteurs sont placés aux endroits stratégiques du réseau et qui captent les paquets circulant .Ces paquets sont envoyés à un analyseur sécurisé, qui les analyse et les traite éventuellement en utilisant la base de signatures et il génère des alertes. Cet analyseur est généralement situé sur un réseau isolé, qui relie uniquement les capteurs et l'analyseur. **[7]**

Les capteurs placés sur le réseau sont placés en mode furtif, de façon à être invisibles aux autres machines. Pour cela, leur carte réseau est configurée en mode *ou* aucune adresse IP n'est configurée. **[7]**

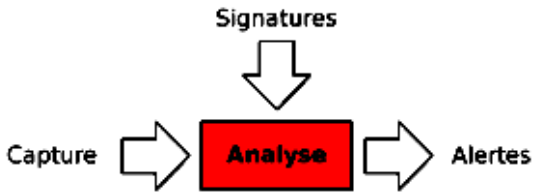


Figure II.6 : Composants d'un NIDS **[7]**

Les avantages des NIDS sont les suivants : **[7] [14] [11]**

- ils peuvent être complètement cachés sur le réseau, donc un attaquant ne saura pas qu'il est contrôlé.
- un système NIDS unique peut être employé pour contrôler le trafic d'un grand nombre de systèmes cibles potentiels.
- il peut capturer le contenu de tous les paquets envoyés à un système cible.
- une seule tâche à effectuer : regarder le trafic et le traiter.
- déployer un NIDS à un faible impact sur un réseau existant.
- les NIDS sont des systèmes à temps réel.

Les inconvénients des NIDS : **[7] [14] [11]**

- le taux élevé de faux positifs qu'ils génèrent.
- ils ne peuvent donner d'alarmes que si le trafic correspond aux règles ou aux signatures pré configurées.
- ils peuvent manquer le trafic intéressant si le trafic est important sur la bande passante ou si des routes altérées sont utilisées.
- il ne peut pas déterminer si une attaque a réussi.
- il ne peut pas examiner le trafic chiffré.
- il faut des configurations spéciales sur les réseaux commutés pour que le NIDS puisse voir tout le trafic.

Quelques outils NIDS « open sources »

- Snort **[34]**
- Benids **[35]**
- Hank **[36]**
- Prelude **[37]**
- Firestorm **[38]**
- Bro **[39]**

II.6.3.2 HIDS (Host Based Intrusion Detection System) [14]

Les *HIDS* analysent le fonctionnement, ou l'état des machines sur lesquelles ils sont installés, afin de détecter les attaques. Pour cela, ils auront pour mission d'analyser les journaux systèmes, de contrôler l'accès aux appels systèmes, de vérifier l'intégrité des systèmes de fichiers, etc. Ils sont en général, placés sur des machines sensibles, susceptibles de subir des attaques et possédantes des données sensibles pour l'entreprise. Les serveurs web et applicatifs peuvent notamment être protégés par un *HIDS*.

Les avantages des *HIDS* sont les suivants : [7] [11] [14]

- il est possible de constater immédiatement l'impact d'une attaque et donc de mieux réagir.
- il est possible d'observer les activités se déroulant sur l'hôte avec précision et d'optimiser le système en fonction des activités observées.
- ils permettent de détecter plus facilement les attaques de type *Cheval de Troie*, alors que ce type d'attaque est difficilement détectable par un *NIDS*.
- les *HIDS* peuvent souvent fonctionner dans des environnements avec un trafic réseau chiffré.
- ils permettent également de détecter des attaques impossibles à détecter avec un *NIDS*, car elles font partie du trafic crypté.
- ils génèrent peu de faux positifs, permettant d'avoir des alertes pertinentes.

Les inconvénients des *HIDS* : [7] [11] [14]

- ils peuvent être identifiés et mis hors service par un attaquant.

- ils ne peuvent donner l'alerte que si les entrées des journaux d'événements ou les appels au système correspondent à des signatures ou des règles pré configurées.
- sensibles aux attaques de type Déni de Service.
- ils sont assez gourmands en CPU et peuvent parfois altérer les performances de la machine hôte.

Quelques outils HIDS

- Tripwire **[43]**
- SWATCH **[44]**
- DragonSquire **[45]**
- Tiger **[46]**
- Security Manager **[47]**

II.6.3.3 IDS hydrides (*NIDS+HIDS*) [14] [6] [31]

Les systèmes de détection d'intrusions hybrides, rassemblent les caractéristiques de plusieurs systèmes de détection d'intrusions différents. En pratique, on ne retrouve que la combinaison de *NIDS* et *HIDS*. Ils permettent, en un seul outil de surveiller le réseau et l'hôte. Les sondes sont placées dans des points stratégiques, et agissent comme *NIDS* et/ou *HIDS* suivant leurs emplacements. Toutes les sondes remontent alors les alertes à une machine qui va centraliser, agréger, et lier les informations d'origines multiples.

II.6.4 Paradigme de détection [22]

Un IDS, peut tenter d'identifier des attaques en s'appuyant sur : des informations relatives aux transitions ayant lieu dans le système (l'exécution de certains programmes, de certaines séquences d'instructions, l'arrivée de certains paquets réseau, etc.) ou bien en étudiant l'état de certaines parties du système (par exemple, l'intégrité des programmes

stockés, les privilèges des utilisateurs, etc.). Durant ces deux types d'inspection, l'IDS récupère les informations en interrogeant directement le système ou en écoutant passivement les événements.

II.6.5. Mode de supervision [22]

Une autre caractéristique des systèmes de détection d'intrusions, c'est leur fréquence d'utilisation :

– ***Périodique*** : certains systèmes de détection d'intrusions, analysent périodiquement les fichiers d'audit à la recherche d'une éventuelle intrusion ou anomalie passée. Cela peut être suffisant dans des contextes peu sensibles, par exemple du fait qu'un *HIDS* analyse des fichiers traces transmis seulement toutes les heures.

– ***Continue*** : la plupart des systèmes de détection d'intrusions récents, effectuent leur analyse des fichiers d'audit ou des paquets réseau de manière continue, afin de proposer une détection en quasi temps réel. Cela est nécessaire dans des contextes sensibles (confidentialité). C'est toutefois coûteux en temps de calcul car il faut analyser à la volée tout ce qui se passe sur le système.

II.7. Méthodes de classification et d'IA pour la détection d'intrusions

Plusieurs algorithmes de classification et d'IA, peuvent être utiles dans la détection d'intrusions. Ces algorithmes génèrent des classificateurs, sous forme d'arbre de décision, de règles ou de réseau de neurones, etc. Une application dans la détection d'intrusions serait d'appliquer ces algorithmes à une quantité suffisante de données d'audit normales ou anormales, pour générer un classificateur capable d'étiqueter comme appartenant à la catégorie normale ou anormale de nouvelles données d'audit. Parmi ces techniques de classification, nous citons :

II.7.1. réseaux bayésiens [42]

Les réseaux bayésiens sont des outils de raisonnement avec des informations incertaines dans le cadre de la théorie des probabilités. Les réseaux bayésiens, utilisent des graphes acycliques dirigés pour la représentation des relations causales et des probabilités

conditionnelles (de chaque noeud dans le contexte de ses parents), pour exprimer l'incertitude sur ces relations.

Valdes **[49]** a proposé une nouvelle approche hybride, pour la détection d'intrusions se basant sur les réseaux bayésiens, en utilisant une forme simplifiée des réseaux bayésiens, appelée réseaux bayésiens naïfs, composée de deux niveaux: une racine qui représente la nature de la session (normal et les différents types d'attaques), et plusieurs noeuds enfants, chacun d'entre eux correspond à un attribut de la connexion.

Les réseaux bayésiens naïfs ont plusieurs avantages dus, en particulier, à leur construction qui est très simple. Par ailleurs, l'inférence (classification) est assurée de façon linéaire (alors que l'inférence dans les réseaux bayésiens qui ont une structure générale est connue comme un problème NP complet **[3]**. En plus, la construction des réseaux bayésiens naïfs est incrémentale, dans le sens qu'elle peut être facilement mise à jour (notamment, il est toujours possible de prendre en considération de nouvelles classes). Cependant, les réseaux bayésiens naïfs travaillent sous une hypothèse d'indépendance très forte entre les attributs dans le contexte de la nature de la session. Une telle hypothèse n'est pas toujours valable et peut entacher les résultats.

II.7.2. Arbre de décision [50] [51]

Les arbres de décision, représentent l'une des techniques les plus connues et les plus utilisées en classification. Leur succès est notamment dû à leur aptitude à traiter des problèmes complexes de classification. En effet, ils offrent une représentation facile à comprendre et à interpréter, ainsi qu'une capacité à produire des règles logiques de classification.

Un arbre de décision est composé de:

Noeuds de décision : contenant chacun un test sur un attribut.

Branches : correspondant généralement à l'une des valeurs possibles de l'attribut sélectionné.

Noeuds feuilles : comprenant les objets qui appartiennent à la même classe.

L'utilisation des arbres de décision dans les problèmes de classification se fait en deux principales étapes:

1. Etape de construction de l'arbre : Cette étape se base sur un ensemble d'apprentissage, et consiste à sélectionner, pour chaque noeud de décision, l'attribut test 'approprié'. Elle consiste aussi à définir la classe libellant chaque feuille.

2. Etape de classification : Pour trouver la classe de l'objet, il suffit de suivre le chemin en partant de la racine de l'arbre de décision jusqu'aux feuilles en effectuant les différents tests à chaque noeud selon les valeurs des attributs de l'individu à classer.

Plusieurs IDS ont été proposés à base des arbres de décision, en utilisant la base KDD dans l'étape de construction de l'arbre (chapitre III).

Benfarhat **[42]** a proposé des IDS comportementaux à base des réseaux bayésiens naïfs et des arbres de décision. L'étude expérimentale est effectuée sur la base KDD.

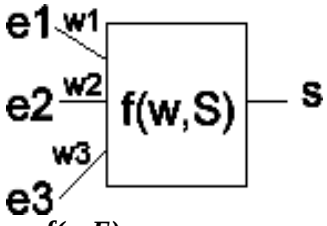
II.7.3. Réseaux de neurones [52] [58]

Les réseaux de neurones formels sont à l'origine une tentative de modélisation mathématique du cerveau humain. Les premiers travaux datent de 1943 et sont l'oeuvre de MM. *Mac Culloch et Pitts* **[53]**.Ils présentent un modèle assez simple pour les neurones et explorent les possibilités de ce modèle.

L'idée principale des réseaux de neurones "modernes" est la suivante : **[52]**

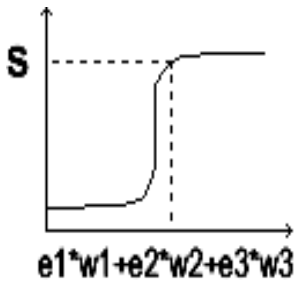
On se donne une unité simple, un neurone, qui est capable de réaliser quelques calculs élémentaires. On relie ensuite entre elles un nombre important de ces unités et on essaye de déterminer la puissance de calcul du réseau ainsi obtenu. Il est important de noter que ces neurones manipulent des données numériques et non pas symboliques.

Voici comment sont généralement modélisés les neurones (de manière logicielle, ou matérielle par des automates) : **[54]**



s = f(w,E)

En général on a : **[54]**



s = f(w1*e1+w2*e2+w3*e3), avec f une sigmoïde

La fonction **f** peut être une sigmoïde comme sur l'exemple ci-dessus.

Un réseau de neurones peut donc être représenté par les poids synaptiques (w) des différents neurones. Ces poids varient au cours du temps, en fonction des entrées présentées (E). Le grand problème étant de savoir comment modifier ces poids.

Caractéristiques [54]

1. Apprentissage, mémoire et oubli :

Une caractéristique des réseaux de neurones est leur capacité à apprendre (par exemple à reconnaître une lettre, un son...). Mais cette connaissance n'est pas acquise dès le départ. La plupart des réseaux de neurones apprennent par l'exemple (de la même manière qu'un enfant apprend à reconnaître un chien à partir d'exemples de chiens qu'il a vu). Ils ont donc une capacité à classer, généraliser, à mémoriser et aussi à oublier. Un réseau de neurones reconnaîtra d'autant plus facilement un objet qu'il l'aura vu souvent.

2. Connectivité :

La connectivité des réseaux, c'est à dire la manière dont les neurones sont connectés, peut être totale (tous les neurones sont connectés entre eux) ou par couche (les neurones d'une couche ne sont connectés qu'aux couches suivantes - il existe des réseaux mono couches ou multicouches.

3. Apprentissage supervisé / non supervisé :

Les réseaux de neurones se divisent en deux principales classes, les réseaux à apprentissage supervisés et les réseaux à apprentissage non supervisés **[54][52]**.
Pour les réseaux à apprentissage supervisés , on présente au réseau des entrées, et au même temps les sorties que l'on désirerait pour cette entrée. Par exemple on lui présente en entrée une lettre " a " manuscrite et en sortie un code correspondant à la lettre " a ". Le réseau doit alors se reconfigurer, c'est-à-dire calculer ses poids afin que la sortie qu'il donne corresponde bien à la sortie désirée.
Pour les réseaux à apprentissage non supervisé, on présente une entrée au réseau et on le laisse évoluer librement jusqu'à ce qu'il se stabilise.

4. Calcul des poids synaptique:

La rétro-propagation **[54]** est une méthode de calcul des poids pour un réseau à apprentissage supervisé qui consiste à minimiser l'erreur quadratique de sortie (somme des carrés de l'erreur de chaque composante entre la sortie réelle et la sortie désirée).

Debar Hervé a proposé un IDS à base des réseaux de neurones. En modélisant le comportement habituel des utilisateurs d'un système informatique. Ce modèle s'appuie sur une architecture de réseaux de neurones relativement peu étudiée mais qui permet de faire de la prédiction avec des taux de succès importants. Ce modèle est ensuite étudié pour l'application

plus particulière à la détection d'intrusion. Ce modèle réagit de manière forte lorsque l'utilisateur sur lequel il a été entraîné est remplacé par un autre utilisateur.

II.8. Les systèmes de détection d'intrusions actuels [6]

Le premier système de détection d'intrusions a été proposé en 1980 par *James ANDERSON[24]*. Il en existe maintenant beaucoup d'autres, commerciaux ou non. La majorité de ses systèmes se basent sur les deux approches, comportementale et par scénarios.

Stefan AXELSSON donne un modèle d'architecture de base pour un système de détection d'intrusions : un module s'occupe de la collecte d'informations d'audit, ces données étant stockées quelque part, un module de traitement des données qui interagit avec ces données de l'audit et les données en cours de traitement, ainsi qu'avec les données de référence (signatures, profils) et de configuration entrées par l'administrateur du système de sécurité. En cas de détection, le module de traitement remonte une alarme vers l'administrateur du système de sécurité ou vers un module. Une réponse sera ensuite apportée sur le système surveillé par l'entité alertée. Les imperfections de ce type de systèmes monolithiques et même des systèmes de détection d'intrusions en général sont à prendre en compte. *stefano MARTINO* souligne que si un certain nombre de techniques ont été développées jusque là, pour les systèmes de détection d'intrusions, la plupart analysent des évènements au niveau local et se contentent de remonter une alarme sans agir. Ils détectent de plus, les activités dangereuses d'un utilisateur sans se préoccuper du code dangereux.

II.9. Imperfection des systèmes de détections d'intrusions [7]

Les systèmes de détections d'intrusions de nos jours présentent quelques imperfections, dans la plupart des cas, ces systèmes sont faits d'un seul bloc ou module qui se charge de toute

l'analyse. Ces systèmes monolithiques exigent beaucoup de données d'audit, de ce fait ils utilisent beaucoup de ressources de la machine surveillée. L'aspect monolithique pose également des problèmes de mise à jour et constitue un point d'attaque unique pour ceux qui veulent s'introduire dans le système d'informations. D'autres imperfections plus générales sont relevables dans les systèmes de détection d'intrusions actuels :

- Même en implémentant les deux types d'approches, certaines attaques sont indécelables et les systèmes de détection sont eux-mêmes attaquables. Les approches comportementale et par scénarios ont elle-même leurs limites.
- Les groupes de travail sur ce sujet sont relativement fermés et il n'y a pas de méthodologie générique de construction. Aucun standard n'a pour l'instant vu le jour dans ce domaine. Des groupes y travaillent, notamment au sein de la DARPA et de l'IETF.
- Les mises à jour de profils, de signatures d'attaques ou de façon de spécifier des règles sont généralement difficiles. De plus, les systèmes de détection d'intrusions demandent de plus en plus de compétence à celui qui administre le système de sécurité.
- Les systèmes de détection, sont généralement écrits pour un seul environnement et ne s'adapte pas au système surveillé, alors que les systèmes d'informations sont la plupart du temps, hétérogènes et utilisés de plusieurs façons différentes.
- Aucune donnée n'a été pour l'instant publiée, pour quantifier la performance d'un système de détection d'intrusions. De plus, pour tester ces systèmes, les attaques sont de plus en plus difficiles à simuler.

Conclusion

Nous avons présenté tout au long de ce chapitre, une étude des systèmes de détection d’intrusions. Afin de remplir ces objectifs, diverses méthodes de détections d’intrusions ont été proposées, elles sont basées principalement sur deux approches : l’approche comportementale et l’approche par scénarios. Parmi ces méthodes nous soulignons la classification. Plusieurs travaux dans le cadre de la détection ont été menés sur les algorithmes de classifications. Cependant, ces algorithmes possèdent des limites qui peuvent entacher la détection d’intrusions : un arbre de décision présente un très gros défaut dans le

cas où des instances de l'ensemble de test ne satisfont aucune règle de la base d'apprentissage. Afin de remédier a ces défauts en particulier et de résoudre les limites des IDS en général , nous proposons une nouvelle méthode de détection d'intrusions (**qui n'a pas été exploitée à ce jour dans le domaine de la sécurité**) qui consiste à utiliser des algorithmes basés sur les fourmis artificielles. Pour cela, nous appliquons l'algorithme de classification non supervisé ***AntClass*** sur une base de donnée appelée **base d'apprentissage KDD**.

Notre contribution est présentée dans la deuxième partie de ce mémoire, elle est composée de trois chapitres : le premier s'intéresse à la base d'apprentissage et de *test KDD,* le second traite l'algorithme *AntClass* et la conception de notre IDS que nous nommons ***IDSAC « système de détection d'intrusions à base de AntClass » ,***le troisième chapitre présente l'implémentation ,test et validation du système ***IDSAC*** .

Partie II :

Notre contribution

Proposition d'IDS comportemental à base d'une méthode basée sur le fonctionnement de colonies des fourmis :
Méthode de classification non supervisée
«AntClass »

CHAPITRE III.

Base d'apprentissage et de test KDD

Sommaire :

III.1. Introduction

Nous avons constaté que les IDS contiennent plusieurs lacunes vu l'évolution des techniques utilisées par les attaquants, afin de remédier à ces lacunes, la détection d'intrusions doit s'orienter vers de nouvelles méthodes de détection, pour mieux assurer la sécurité de réseaux.

Nous proposons dans cette thèse, une nouvelle approche de détection d'intrusions qui consiste à utiliser un algorithme basé sur les fourmis artificielles.

Notre travail consiste à réaliser un IDS comportemental basé sur l'algorithme de classification non supervisé *AntClass*. Notre IDS a besoin donc d'une phase d'apprentissage, pour cela, nous appliquons l'algorithme de classification non supervisé *AntClass* sur une base de données appelée **base d'apprentissage KDD**, pour former le profil normal du système . Afin de tester notre IDS, nous utilisons une base de test appelée **base de test KDD** contenant des connexions normales ,et des connexions considérées comme étant des attaques. Cette base de test est très appropriée pour évaluer un système de détection d'intrusions de type comportemental, puisqu'elle contient des attaques qui ne figurent pas dans la base d'apprentissage **.**Dans ce présent chapitre, nous décrirons la base *KDD.*

III.2. Description de la Base KDD [`42] [57] [62]

La base *KDD* est une base de données orientées détection d'intrusions. Elle présente des lignes *TCP/IP* où chaque ligne est une connexion caractérisée par ***41*** attributs, telle que *la durée de la connexion, le type du protocole, etc*. En tenant compte des valeurs de ses attributs, chaque connexion dans *KDD* est considérée comme étant une connexion normale ou bien une attaque.

Les données *KDD* sont en fait des données formatées fournies par *DARPA*. Ces données présentent *7* semaines de données libellées pour l'apprentissage et *2* semaines de données non libellées pour le test (Correspondant au trafic réseau simulant un réseau local d'US Air force). Il existe des travaux récents utilisant ces données (voir chapitre II).

III.3. Attaques de la base KDD[62]

La base KDD recense 38 attaques possibles qui peuvent être regroupées en quatre catégories (listées dans le tableau III.1) : **[42]**

III.3. 1. Déni de sevice-Denial-Of-Service(DOS) :

Il s'agit d'empêcher par tous les moyens les utilisateurs de se servir des ressources disponibles en temps normal. Ces attaques sont à but purement destructeur, au sens où une telle attaque ne permet pas à l'attaquant de compromettre une machine, ni de voler des informations. De telles attaques sont souvent très simples à mettre en place et donnent une sensation de puissance à l'attaquant, ce qui explique leur fréquence.

III.3.2. Attaques de type Remote to User (R2L) :

Pour ce genre d'attaques, les attaquants essaient d'exploiter la vulnérabilité du système afin de contrôler la machine distante.

III.3.3. Attaques de type User to Root Attaks (U2R) :

L'attaquant essaie d'avoir les droits d'accès à partir d'un poste, afin d'accéder au système.

III.3.4. Reconnaissance –Probing :

Ces attaques ne sont pas destructrices au sens où elles empêchent une entité de fonctionner correctement, mais permettent d'acquérir des informations parfois cruciales, pour mener une attaque de plus grande envergure plus tard.

DOS	Probing	R2L	U2R
Apache2	Ipsweep	Ftp_write	Buffer_overflow
Back	Mscan	Guess_passwd	Httptunnel
Land	Nmap	Imap	Loadmodule
Mailbomb	Portsweep	Multihop	Xterm
Neptune	Saint	Named	Perl
Pod	Satan	Phf	Ps
Processtable		Dict	Rootkit
Smurf		Snmpguess	
Teardrop		Spy	
Udpstorm		Sqlattack	
		Warezclient	
		Warezmaster	
		Xlock	
		Xsnoop	
		Guest	

Tableau III.1 : Types d'attaques **[42]**

La KDD contient deux types de bases de connexions : [57]

1. **Base d'apprentissage KDD**

- Enregistrement :

  - 41 attributs + nom de classe pour apprendre

-Fichiers au format texte

- ~5 millions de connexions (10% (494000) utilisées)
  - 4 classes d'attaques + trafic normal
    - Probing : scan de port (nmap, satan …)
    -DoS : déni de Service (syn flooding, smurf …)
    - U2R : acquisition des privilèges d'un super utilisateur (buffer overflow)
    -R2L : accès illégitime à partir d'une machine distante(password guessing)
    -Normal : trafic légitime

2. Base de test KDD [57]

- Enregistrement :
  - 41 attributs + nom de classe pour vérifier
- ~ 311000 connexions
  - 4 classes d'attaques enrichies + trafic normal
    - Probing : scan de port (mscan, saint)
    - DoS : déni de Service (apache2, …)
    - U2R : acquisition des privilèges d'un super utilisateur (sqlattack…)
  - R2L : accès illégitime à partir d'une machine distante (snmpguess, snmpgetattack…)
    -Normal : trafic légitime

III.4. Attributs [42]

Les attributs caractérisant chaque connexion de la base *KDD* **[42]**, sont détaillés dans le *tableau III.2*. En effet, on peut distinguer les attributs basiques des connexions TCP individuelles, les attributs relatifs au contenu, les attributs relatifs aux temps calculés en utilisant des fenêtres de temps de deux secondes et les attributs basés sur l'hôte, calculés en utilisant des fenêtres de temps de 100 connexions . Ces attributs sont utilisés pour caractériser les attaques qui scannent les hôtes (ou les ports) en utilisant un intervalle de temps plus large que deux secondes.

	Attributs basiques
A1	durée de la connexion (nb de secondes)
A2	type du protocole, ex. tcp, udp, etc.
A3	service réseau (destination) ex. http, telnet
A4	statut de la connexion (normal ou erreur)
A5	nb de données (en octets) de la source vers la destination
A6	nb de données (en octets) de la destination vers la source
A7	1 si la connexion est de/vers le même hôte/port; 0 autrement
A8	nb de fraguements "erronnés"
A9	nb de paquets urgents
	Attributs relatifs au contenu
A10	nb d'indicateurs "hot"
A11	nb d'essais login ratés
A12	1 si succès du login ; 0 autrement
A13	nb de conditions de "compromis"
A14	1 si la racine shell est obtenu; 0 autrement
A15	1 si la commande on a la commande "racine su" ; 0 autrement
A16	nb d'accès à la "racine"
A17	nb de créations d'opérations de fichiers
A18	nb de shell prompts
A19	nb d'opérations sur les fichiers de contrôle d'accès
A20	nb de commandes outbound dans une session ftp
A21	1 si le login appartient à la liste "hot" ; 0 autrement
A22	1 si le login est login "invité" ; 0 autrement
	Attributs basés sur le temps utilisant des fenêtres de temps de deux secondes
A23	nb de connex. pour le *même hôte*
A24	nb de connex. pour le *même service*
A25	% de connex. pour le *même hôte* ayant l'erreur "SYN"
A26	% de connex. pour le *même service* ayant l'erreur "SYN"
A27	% de connex. pour le *même hôte* ayant l'erreur "REJ"
A28	% de connex. pour le *même service* ayant l'erreur "REJ"
A29	% de connex. pour le *même hôte* utilisant le *même service*
A30	% de connex. pour le *même hôte* utilisant *différents services*
A31	% de connex. pour le *même service* utilisant *différents hosts*
	Attributs basés sur le temps utilisant des fenêtres de temps de 100 connex.
A32	nb de connex. pour le *même hôte*
A33	nb de connex. pour le *même hôte* utilisant le *même service*
A34	% de connex. pour le *même hôte* utilisant le *même service*
A35	% de connex. pour le *même hôte* utilisant *différents services*
A36	% de connex. pour le *même hôte* ayant le port src
A37	% de connex. pour le *même hôte* et le *même service* utilisant *différents hosts*
A38	% de connex. pour le *même hôte* ayant l'erreur "SYN"
A39	% de connex. pour le *même hôte* et le *même service* ayant l'erreur "SYN"
A40	% de connex. pour le *même hôte* ayant l'erreur "REJ"
A41	% de connex. pour le *même hôte* et le *même service* ayant l'erreur "REJ"

Tableau III.2 : Liste des attributs **[42]**

Conclusion

Dans ce chapitre, nous avons exposé la base d'apprentissage et de *test KDD* qui est utilisée dans notre étude expérimentale, en effet, nous allons valider l'algorithme *AntClass* sur la base d'apprentissage *KDD* , afin de construire le profil normal du système à surveiller , ensuite utiliser la base de *test KDD* pour tester notre IDS. Dans le chapitre suivant, nous allons détailler la méthode *AntClass*.

CHAPITRE IV.

Système de détection d'intrusions à base de la méthode *AntClass*

Sommaire :

Partie I: La description de la méthode de classification non supervisée *AntClass*

IV.1. Introduction

Après avoir présenté la KDD, à présent nous nous intéressons à l'algorithme de classification ***AntClass,*** nous soulignons les motivations qui nous ont poussées à choisir cet algorithme et nous proposons l'architecture de notre IDS.

Le problème de la classification est tout à fais adapté à une résolution distribuée. Les fourmis, en effet, font fasse à ce genre de problème de quand leur nid est dérangé et qu'elles doivent rassembler leurs œufs en fonction de leurs développements. Les motivations qui nous ont poussées à choisir cet algorithme sont les suivantes : Pour classer des données , ou partitionner un ensemble d'objet, de nombreux algorithmes déterministes (par exemple, les centres mobiles que nous présentons dans les sections suivantes) nécessitent qu'une partition initiale soit fournie à l'algorithme. C'est l'inconvénient majeur de ces méthodes : la partition obtenue à partir de cette initialisation risque d'être localement optimale, le seul moyen de contourner ce problème étant de lancer la méthode avec une partition initiale différente. La même remarque pourrait être faite concernant le nombre de classes :Les centres mobiles nécessitent que le nombre de classes soit initialisé, ce qui diminue l'intérêt de la méthode pour un expert cherchant justement à connaître ce nombre de classes.

IV.2. Ce que font les fourmis réelles

Dans la nature, les fourmis offrent un modèle stimulant pour le problème du partitionnement. L'exemple du tri collectif du couvain ou de la constitution de cimetières est le plus marquant. Certains travaux expérimentaux, montrent que certaines espèces de fourmis sont capables d'organiser spatialement divers éléments du couvain : les oeufs, les larves et les nymphes **[40]**.

Le modèle de règles utilisé est relativement simple : **[40]**

- lorsqu'une fourmi rencontre un élément du couvain, la probabilité qu'elle s'en empare est d'autant plus grande que cet élément est isolé ;
- lorsqu'une fourmi transporte un élément du couvain, elle le dépose avec une probabilité d'autant plus grande que la densité d'éléments du même type dans le voisinage est grand.

Pour rassembler en tas un ensemble d'éléments (d'objets) de même type, les probabilités de ramasser un objet (P_p) et de déposer (p_d) ont été explicitées **[41]** :

- quand une fourmi ne transporte aucun élément, sa probabilité de ramasser un, rencontré sur son chemin, est donnée par la formule:

$$p_p = \left(\frac{k_1}{k_1 + f}\right)^2 \qquad \textbf{(IV.1)}$$

Où ***k1*** est une constante positive et ***f*** correspond à la proportion d'éléments perçus dans le voisinage de la fourmi.

Quand il y a peu d'objets dans le voisinage de l'objet convoité par la fourmi, ***f*** <<***k1*** ce qui signifie que p_p est proche de 1 et l'objet a beaucoup de chance d'être ramassé. Inversement, quand le voisinage est dense en éléments, ***f***>>***k1*** et alors P_p est proche de 0.

- quand une fourmi chargée d'un objet se déplace, sa probabilité de déposer l'objet est donnée par :

$$p_d = \left(\frac{f}{k_2 + f}\right)^2 \qquad \textbf{(IV.2)}$$

Où ***k2*** est une constante positive. ***f*** correspond au nombre d'objets rencontrés durant les ***T*** derniers déplacements divisés par le nombre maximum d'objets qui auraient pu être rencontrés.

L'adaptation de ce principe à des objets de plusieurs types, par exemple deux : ***A*** et ***B***, peut alors se faire en particularisant $f : f_A$ et f_B correspondent à la proportion d'objets de type ***A*** et ***B.*** Le comportement de rassemblement se transforme alors en tri.

Ces principes ont trouvé leurs premières applications en robotique collective et de nombreux travaux en découlent. **[40]**

Dans ce qui suit, nous allons présenter l'inspiration de Lumer et Faieta **[40] [41],** qui est aussi la base de l'algorithme *AntClass*.

IV.3. Les fourmis artificielles

Lumer et Faieta ont proposé un algorithme utilisant une mesure de dissimilarité entre objets (sous la forme d'une distance euclidienne) **[40]**.

Les objets correspondent à des points d'un espace numérique à ***M*** dimensions sont plongés dans un espace discret de dimension moindre (typiquement de dimension 2). Cet espace discret, s'apparente alors à une grille ***G*** dont chaque case peut contenir un objet. Les agents se déplacent sur ***G*** et perçoivent une région ***Rs*** de ***s*** x***s*** cases dans leur voisinage. **La figure IV.1** donne un exemple de grille avec une fourmi (représentée par **x**) et son périmètre de détection (en trait épais). Les objets sont représentés, par des carrés dont l'intérieur (invisible pour la fourmi) représente la classe d'origine.

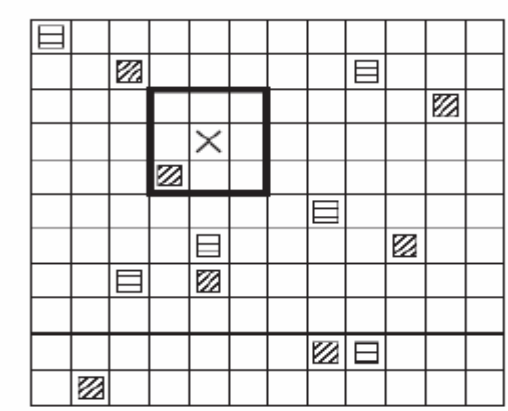


Figure IV.1 : Grille utilisée par Lumer et Faita **[40]**

Les formules **(IV.3)** et **(IV.4)** sont modifiées de la façon suivante : **[40]**

$$p_p(o_i) = \left(\frac{k_1}{k_1 + f(o_i)}\right)^2$$

(IV.3)

$$p_d(o_i) = \begin{cases} 2f(o_i) & \text{si } f(o_i) < k_2 \\ 1 & \text{si } f(o_i) \geq k_2 s \end{cases}$$

(IV.4)

Comme on peut le remarquer, la fonction de densité locale dépend de l'objet considéré o_i et de sa position sur la grille $\boldsymbol{r}(o_i)$. Elle est calculée de la manière suivante :

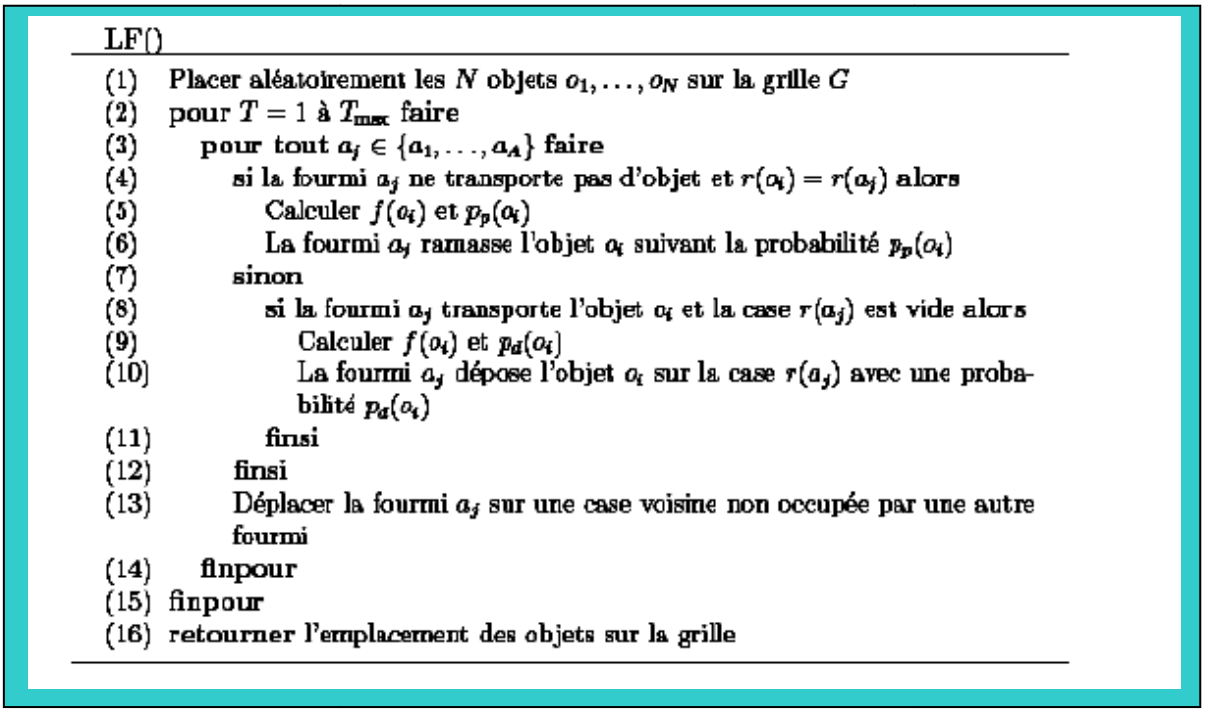
$$f(o_i) = \begin{cases} \frac{1}{s^2} \sum_{o_j \in R_s(r(o_i))} 1 - \frac{d(o_i, o_j)}{\alpha} & \text{si } f > 0 \\ 0 & \text{sinon} \end{cases}$$

(IV.5)

$\boldsymbol{f}(o_i)$ est alors une mesure de la similarité moyenne de l'objet o_i avec les objets o_j présents dans son voisinage. $\boldsymbol{\alpha}$ est un facteur d'échelle déterminant dans quelle mesure la dissimilarité entre deux objets est prise en compte. L'algorithme suivant donne les étapes de la méthode en utilisant $\boldsymbol{A}$ fourmis $\{a_1, a_2, \ldots\ldots, a_A\}$. Les paramètres ont les valeurs suivantes : ***k1 = 0.1***, ***k2 = 0.15***,

$\boldsymbol{s = 3}$, $\boldsymbol{\alpha = 0.5}$ et $\boldsymbol{T_{max} = 10^6}$.

Algorithme LF



LF()

(1) Placer aléatoirement les N objets $o_1, \ldots, o_N$ sur la grille G
(2) pour $T = 1$ à T_{max} faire
(3) pour tout $a_j \in \{a_1, \ldots, a_A\}$ faire
(4) si la fourmi a_j ne transporte pas d'objet et $r(o_i) = r(a_j)$ alors
(5) Calculer $f(o_i)$ et $p_p(o_i)$
(6) La fourmi a_j ramasse l'objet o_i suivant la probabilité $p_p(o_i)$
(7) sinon
(8) si la fourmi a_j transporte l'objet o_i et la case $r(a_j)$ est vide alors
(9) Calculer $f(o_i)$ et $p_d(o_i)$
(10) La fourmi a_j dépose l'objet o_i sur la case $r(a_j)$ avec une probabilité $p_d(o_i)$
(11) finsi
(12) finsi
(13) Déplacer la fourmi a_j sur une case voisine non occupée par une autre fourmi
(14) finpour
(15) finpour
(16) retourner l'emplacement des objets sur la grille

Algorithme IV.1 : Algorithme LF **[41]**

IV.4. Algorithme *AntClass* [40]

L'algorithme *AntClass* est une méthode de classification inspirée de la colonie de fourmis, proposée par *Monmarche* et *Grilles Venturini* **[40].** Cet algorithme utilise des heuristiques basées sur les fourmis, introduit une classification hiérarchique dans la population de fourmis artificielles qui seront aussi capables de transporter des tas d'objets. De plus, *Ant Calss* inclus une hybridation avec l'algorithme des ***centres mobiles*** (détaillé ultérieurement) afin de résoudre les inconvénients inhérents à la classification par les fourmis, par exemple pour accélérer la convergence en éliminant les erreurs évidentes de classification.

AntClass permet de découvrir automatiquement les classes dans des données numériques sans connaître le nombre de classes à priori, sans partition initiale et sans paramétrage délicat.

IV.4. 1. Principe de fonctionnement [40]

- **Répartition des objets sur une grille**

Lumer et *Faieta* (algorithme *LF*) **[41]** ont fait évoluer des agents fourmis sur une grille ***G*** où les objets à partitionner sont positionnés. Le positionnement des objets est initialement aléatoire, *Lumer* et *Faieta* ont réparti les points de l'espace numérique à *M* dimensions dans lequel les objets sont définis vers un espace discret à deux dimensions. Le caractère aléatoire de la disposition initiale n'est pas obligatoire, on pourrait par exemple, envisager de placer les objets suivant le résultat obtenu par une analyse factorielle des correspondances (*AFC*) en discrétisant les coordonnées de chaque objet sur les premiers axes d'inertie (on n'est pas obligé de se limiter aux deux premiers axes, si la grille comporte plus de deux dimensions).

Par rapport à la grille utilisée par l'algorithme *LF*, nous introduisons plusieurs différences : **[40]**

- la grille ***G*** est toroïdale, ce qui signifie que les fourmis passent d'un côté de la grille à l'autre en un seul pas. Cette caractéristique n'est pas présente pour l'algorithme *LF* et cela représente l'inconvénient, de provoquer des effets de bord indésirables (par exemple, la répartition des positions explorées par les fourmis peut ne plus être uniforme) .
- ***G*** est de forme carrée et sa taille est déterminée automatiquement en fonction du nombre d'objets à traiter (ce qui n'est pas précisé pour *LF*). Si ***N*** représente le nombre d'objets, ***G*** comporte ***L*** cases par côté :

$$\mathbf{L = [\sqrt{2N}\ ]}$$

(IV.6)

Cette formule permet de s'assurer que le nombre de cases est au moins égal au nombre d'objets. Par contre, si la grille est trop grande, les fourmis vont perdre beaucoup de temps à y chercher les objets.

- dans *LF*, chaque case ne peut contenir qu'un seul objet, une classe est alors représentée par un amas d'objets (Figure IV.1). Dans le cas *AntClass*, plusieurs objets peuvent être placés sur une seule case, ce qui forme un tas. Dans ce cas, une classe correspond à un tas et une partition est donnée par l'ensemble des tas présents sur la grille. *La figure IV.2* illustre cette caractéristique.

Le principal avantage de cette façon de procéder, est qu'il n'est plus nécessaire de définir un voisinage pour l'estimation de la densité $f(o_i)$ d'objets similaires à l'objet o_i (Formule IV.5). Dans *LF*, la similarité des objets est estimée localement (justement, grâce au voisinage), ce qui peut impliquer qu'un amas comportant beaucoup d'objets, peut avoir des caractéristiques très différentes à deux de ses extrémités.

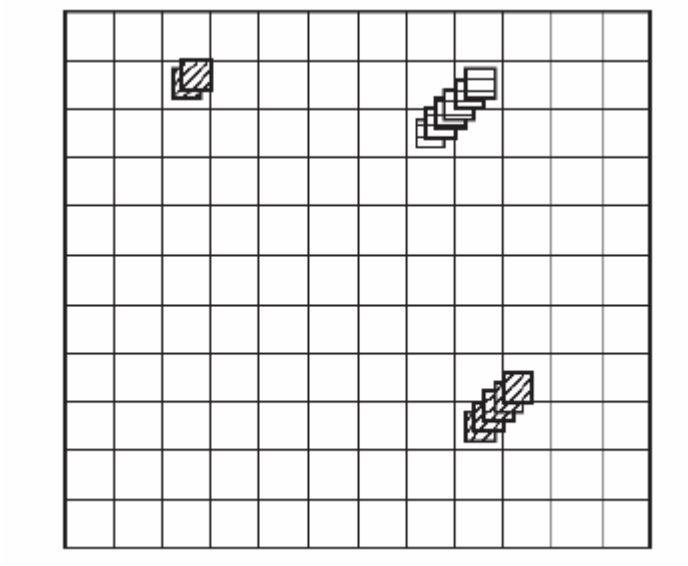


Figure IV.2 : Construction des tas d'objets sur la grille par l'algorithme *AntClass***[40]**

- **Déplacement des fourmis [40]**

A l'initialisation, les A fourmis $\{a_1, \ldots, a_A\}$ sont disposées aléatoirement sur la grille, en vérifiant qu'une case ne peut accueillir qu'une seule fourmi. A chaque itération de l'algorithme, chaque fourmi a_i se déplace aléatoirement sur la grille à une vitesse $v(a_i)$. Concrètement, si $(x(a_i), y(a_i))$ sont les coordonnées de la fourmi a_i sur G, après un déplacement, la nouvelle position sera dans l'intervalle $([x(a_i)-v(a_i), x(a_i)+ v(a_i)], [y(a_i)-v(a_i), y(a_i)+v(a_i)])$ en tenant compte du fait que G est toroïdale. Enfin la direction choisie par une

fourmi dépend de sa direction précédente : elle a une probabilité égale à 0.6 de continuer tout droit et de 0.4 de changer de direction. Dans ce cas, elle a une chance sur deux de tourner de 45 degrés à gauche ou à droite.

Dans LF, chaque fourmi ne peut transporter qu'un seul objet à la fois. *Monmarche* et *Grilles Venturini* **[40]** ont généralisé ce point en dotant chaque fourmi d'une capacité de transport $c(a_i)$.

Afin de déterminer les règles que les fourmis vont utiliser sur la grille pour manipuler les objets, il faut leur donner certaines mesures, de dispersion d'un tas ou de la dissimilarité entre deux objets par exemple. Nous définissons ces notations, que nous utiliserons dans notre travail : **[40]**

-La distance maximale entre deux objets de l'ensemble *O* :

$$d^*(O) = \max_{(i,j)\in\{1,\dots,N\}^2} \{d(\mathbf{x}_i, \mathbf{x}_j)\}$$

(IV.7)

- La distance moyenne entre deux objets de l'ensemble O :

$$\bar{d}(O) = \frac{2}{N(N-1)} \sum_{(i,j)\in\{1,\dots,N\}^2, i<j} d(\mathbf{x}_i, \mathbf{x}_j)$$

(IV.8)

- La distance maximale entre les objets d'un tas $\boldsymbol{T}_j$ et son centre de gravité $\boldsymbol{g}_j$.

$$d_g^*(T_j) = \max_{\mathbf{x}_i\in T_j} \{d(\mathbf{x}_i, \mathbf{g}_j)\}$$

(IV.9)

- La distance moyenne entre les objets d'un tas $\boldsymbol{T_j}$ et son centre de gravité $\boldsymbol{g_j}$:

$$\bar{d}_g(T_j) = \frac{1}{|T_j|} \sum_{\mathbf{x}_i\in T_j} d(\mathbf{x}_i, \mathbf{g}_j)$$

(IV.10)

- **Ramassage d'objets [40]**

Si la fourmi a_i ne transporte pas d'objet et qu'elle se trouve sur une case contenant un objet ou un tas d'objets T_j, elle a une probabilité p_p de ramasser un objet :

$$p_p(T_j) = \begin{cases} 1 & \text{si } |T_j| = 1 \\ \min\left\{\left(\frac{\bar{d}_g(T_j)}{d(O)}\right)^{k_1}, 1\right\} & \text{si } |T_j| = 2 \\ 1 - 0.9\left(\frac{d_g(T_j)+\varepsilon}{d_g^*(T_j)+\varepsilon}\right)^{k_1} & \text{sinon} \end{cases} \quad \textbf{(IV.11)}$$

Où ε est une petite valeur positive (10^{-5}) et $|T_j|$ le nombre d'objets dans le tas. La fourmi ramasse les objets jusqu'à ce que sa capacité soit atteinte, en choisissant à chaque fois l'objet o_i le plus éloigné du centre de gravité du tas. Si la case ne contient qu'un seul objet ($|T_j| = 1$), il est systématiquement ramassé par la fourmi. Si la case contient un tas de deux objets ($|T_j| = 2$ et $T_j = \{o_u, o_v\}$), la probabilité de ramasser l'un des objets dépend de la distance entre les deux objets et le centre de gravité g_j ($d(x_u,g_i)=d(x_v,g_i)=d_g(T_j)$) et de la distance moyenne entre tous les objets ($\bar{d}(O)$). Enfin, si le tas se compose de plus de deux objets, p_p est proche de 1, quand la distance moyenne au centre est négligeable devant la distance au centre de l'objet le plus éloigné.

k_1 est un paramètre réel positif, permettant de contrôler la forme de la densité de $p_p(T_j)$ quand $|T_j| > 2$.

Si la capacité de la fourmi est supérieure à 1, elle ramasse autant d'objets que sa capacité le lui permet.

- **Dépôt d'objets [40]**

Si la fourmi transporte un objet, et qu'elle se trouve sur une case contenant un ou plusieurs objets, sa probabilité de déposer l'objet o_i sur le tas T_j est donnée par :

$$p_d(o_i, T_j) = \begin{cases} 1 & \text{si } d(x_i, g_j) \leq d_g^*(T_j) \\ 1 - 0.9\min\left\{\left(\frac{d(x_i,g_j)}{d(O)}\right)^{k_2}, 1\right\} & \text{sinon} \end{cases} \quad \textbf{(IV.12)}$$

Où k_2 est un paramètre réel positif, permettant de contrôler la forme de la densité de $p_d(o_i, T_j)$ quand $d(x_i, g_j) > dg^*(T_j)$. Si l'objet o_i transporté par la fourmi est moins éloigné du centre g_j du tas T_j que l'objet du tas le plus éloigné de ce centre, elle dépose systématiquement o_i. Sinon, plus la distance entre o_i et g_j ($\mathbf{d(x_i, g_j)}$) est grande par rapport à la distance moyenne entre les objets de la base ($\bar{d}(O)$), plus la probabilité de déposer sera faible.

Si la capacité de la fourmi est supérieure à ***1*** et qu'elle transporte plusieurs objets, la probabilité de déposer le tas T_i qu'elle transporte sur le tas T_j est calculée de la même façon que pour un objet unique en remplaçant x_i par le centre de gravité g_i des objets transportés.

- **Patience des fourmis [40]**

S'il y a trop de fourmis par rapport au nombre de tas ou d'objets, on peut faire face au problème suivant : tous les tas (dans le cas d'une grande capacité de transport des fourmis) ou tous les objets sont transportés ce qui n'offre plus de possibilité aux fourmis de déposer ce qu'elles transportent. Dans le cas où les fourmis ont une capacité de transport égale à 1, il suffit de s'assurer que le nombre de fourmis est nettement inférieur au nombre d'objets. Par contre quand les fourmis ont une capacité de transport supérieure, le problème peut réapparaître. La solution la plus immédiate est de doter les fourmis d'une certaine patience, qui peut être individuelle, et notée $p(a_i)$. Quand la fourmi a effectué plus de $p(a_i)$ déplacements sans avoir réussi à déposer les objets qu'elle transporte, elle les dépose sur la case où elle se trouve si elle est vide ou l'une de son voisinage, dans le cas contraire. Par la suite, cette patience sera utilisée en particulier quand la capacité $c(a_i)$ d'une fourmi est supérieure à ***1***.

- **Mémoire des fourmis [41]**

Lumer et *Faieta* **[41]** ont introduit un mécanisme de mémoire à court terme pour accélérer le processus d'agrégation. Quand un objet o_i est ramassé par une fourmi a_i, il est comparé aux $m(a_i)$ mémoires de la fourmi (*où* $m(a_i)$ *représente la taille de la mémoire de la fourmi* a_i). Elle se dirige ensuite vers l'emplacement de l'objet le plus similaire à o_i.

Dans *AntClass*, *Monmarche* et *Grilles Venturini* **[40]** ont adapté cette technique en remplaçant la comparaison des objets sur la distance les séparant par la distance entre le

centre de gravité du tas transporté par la fourmi et les tas qu'elle a mémorisé, puisque nos fourmis peuvent transporter plusieurs objets. Dans le cas où la fourmi ne transporte qu'un seul objet, il se confond avec le centre de gravité du tas qu'il forme à lui tout seul. La fourmi gère sa mémoire sous la forme d'une liste *FIFO* où les positions les plus anciennes sont oubliées quand la fourmi mémorise de nouvelles positions. Cette mémorisation est effectuée quand la fourmi dépose un ou plusieurs objets sur un tas.

Concernant la taille de la mémoire à utiliser et en dehors de toute expérimentation, on peut envisager qu'une grande mémoire, sera coûteuse à gérer du point de vue du temps de calcul puisqu'à chaque ramassage d'un ou plusieurs objets, la fourmi calcule la distance entre le centre de gravité de ce qu'elle vient de ramasser et le centre de gravité de chacun des tas mémorisés. Mais si la mémoire est trop petite, comme la fourmi choisit de se diriger vers le tas dont le centre de gravité est le plus proche, son éventail de choix pourrait être trop restreint. Ce qui signifie que son déplacement pourrait être inutile si elle ne dépose rien sur le tas qu'elle a choisi d'atteindre.

La mémoire d'une fourmi est statique dans le sens où le centre du tas dont elle mémorise la position ne varie plus de son point de vue. Si elle revient sur un tas, il est possible qu'il ait été suffisamment modifié pour que sa mémoire ne l'ait pas bien orientée, il peut même avoir disparu.

IV.4.2. Hybridation avec les centres mobiles [40]

Le partitionnement construit par les fourmis n'est pas obligatoirement optimal au sens de l'inertie ***intra-classe (Iw)***, même si les règles de ramassage et de dépôt des objets tendent à agglomérer les objets à des tas dont le centre est proche. L'algorithme des *centres mobiles* (ou plus couramment *K-Means*) offre une réponse simple à la non uniformité de la partition construite par les fourmis. Comme les fourmis agissent localement, il peut rester un certain nombre d'objets méritants d'appartenir à une classe dont le centre est beaucoup plus proche. *K-Means* est alors utilisé pour corriger ce type de défaillance.

Dans les versions précédentes de *AntClass* , *Monmarche* et *Grilles Venturini* **[40]** avaient décidé d'affecter les objets isolés (c'est-à-dire se trouvant sur une fourmi ou seul sur la grille) au tas dont le centre de gravité était le plus proche. Cela est intéressant quand une fourmi est interrompue dans une opération qu'elle pourrait accomplir ou pour les objets qui n'auraient pas été visités sur la grille. Ce deuxième point peut être évité en donnant plus d'itérations aux fourmis, certes au détriment du temps de calcul. Cependant, on peut raisonnablement penser qu'un objet dont les paramètres sont trop bruités ait du mal à trouver une place sur un tas, il a donc plus de chance d'être seul sur la grille (si une fourmi impatiente l'a finalement laissé tomber) ou bien transporté par une fourmi au moment de l'interruption. Il peut être alors intéressant qu'il reste isolé afin d'attirer l'attention, l'algorithme *K-Means* ayant beaucoup de chance de le laisser seul dans sa classe.

L'algorithme *AntClass* désigne une succession d'itérations des fourmis et des centres mobiles. Les fourmis ont pour tâche de réduire le nombre de classes et les centres mobiles d'améliorer globalement la partition découverte par les fourmis.

L'algorithme IV.2 donne la structure générale de la méthode de classification par les fourmis (appelée ***Ants*** par la suite).

L'algorithme IV.3 donne le schéma général de *AntClass*. L'algorithme *K-Means* est initialisé avec la partition obtenue par *Ants*.

Le tableau IV.1 résume les paramètres à fixer pour l'algorithme *Ants*. Ces paramètres sont spécifiés pour chacune des $T_{AntClass}$ itérations de *AntClass*.

Algorithme Ants : regroupement des objets par les fourmis. Les indications entre crochets concernent le cas où les fourmis ont une capacité de transport supérieure à ***1***.

Algorithme Ants

```
(1)  pour t = 1 à T faire
(2)     pour k = 1 à A faire
(3)        Déplacer la fourmi a_k sur une case non occupée par une autre fourmi
(4)        si il y a un tas d'objets T_j sur la même case que a_k alors
(5)           si la fourmi a_k transporte un objet o_i [un tas d'objets T_i] alors
(6)              Déposer l'objet o_i [le tas T_i] transporté par la fourmi sur le tas
                 T_j suivant la probabilité p_d(o_i, T_j) [p_d(T_i, T_j)]
(7)           sinon
(8)              /* La fourmi ne transporte pas d'objet */   Ramasser
                 l'objet o_i le plus dissimilaire du tas T_j [jusqu'à ce que la capacité
                 c(a_k) de la fourmi soit atteinte ou que le tas soit vide] selon la
                 probabilité p_p(T_j)
(9)           finsi
(10)       finsi
(11)    finpour
(12) finpour retourner la grille G
```

Algorithme IV.2 : Algorithme *Ants* **[40]**

Algorithme AntClass

(1) Soit P_0 la partition initiale formée de N classes.
(2) **pour** $t = 1$ à T_{AntClass} **faire**
(3) Initialiser la grille G à partir de la partition P_{t-1} (un tas par classe)
(4) $G' \leftarrow$ ANTS(G)
(5) Construire la partition P' associée à la grille G'
(6) $P_t \leftarrow$ K-MEANS(P')
(7) **finpour**
(8) **retourner** la partition $P_{T_{\text{AntClass}}}$

Algorithme IV.3 : Algorithme *AntClass* **[40]**

Paramètre	Description
A	nombre de fourmis
T	nombre de déplacements de chaque fourmi
$c(a_i)$ $\forall i \in \{1, \ldots, A\}$	capacité de transport des fourmis
$m(a_i)$ $\forall i \in \{1, \ldots, A\}$	taille de la mémoire de chaque fourmi
$v(a_i)$ $\forall i \in \{1, \ldots, A\}$	vitesse sur la grille de chaque fourmi
$p(a_i)$ $\forall i \in \{1, \ldots, A\}$	patience de chaque fourmi
k_1, k_2	paramètres de calcul des probabilités p_p et p_d

Tableau IV.1 : Paramètres de l'algorithme *Ant* **[40]**

IV.5. Algorithme des centres mobiles [40]

L'algorithme *des centres mobiles* (*K-Means* ou *C-Means*) est une technique de partitionnement qui utilise l'erreur quadratique (c'est-à-dire, inertie intra classe) comme critère d'évaluation d'une partition. Dans un premier temps, les objets sont regroupés autour de $\boldsymbol{K}$ centres arbitraires $\boldsymbol{c_1, \ldots, c_K}$ de la manière suivante : la classe $\mathbf{c_i}$ associée au centre $\mathbf{c_i}$ est constituée de l'ensemble des points les plus proches de $\mathbf{c_i}$ que de tout autre centre. Géométriquement, cela revient à partager l'espace des points en $\boldsymbol{K}$ zones définies par les plans médiateurs des segments $\boldsymbol{[c_i, c_j]}$. La figure IV.3 donne l'exemple d'une partition associée à trois centres dans le plan. Les centres de gravité $\boldsymbol{g_1, \ldots, g_K}$ sont ensuite calculés à partir des

classes qui viennent d'être formées. On recommence l'opération en prenant comme centre de classe les centres de gravité trouvés et ainsi de suite, jusqu'à ce que les objets ne changent plus de classe. L'algorithme ci-dessous résume toutes ces opérations.

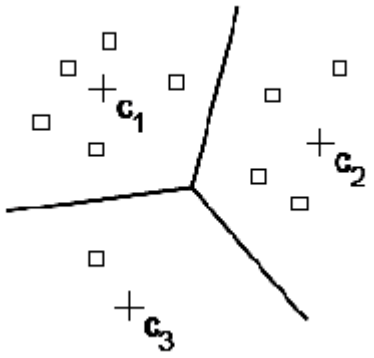


Figure IV.3 : Exemple de partition obtenue par les *centres mobiles* **[40]**

Algorithme K-means (centres mobiles)

K-MEANS(Partition P de K classes)

(1) tantque l'inertie intraclasse ne s'est pas stabilisée faire
(2) Générer une nouvelle partition P' en affectant chaque objet à la classe dont le centre est le plus proche
(3) Calculer les centres de gravité des classes de la nouvelle partition P'
(4) $P \leftarrow P'$
(5) fintantque
(6) retourner P

Algorithme IV.4 : Centres mobiles **[40]**

Partie II: Application de la méthode *AntClass* sur la base KDD

Le premier IDS que nous allons concevoir sera nommé ***IDSAC*** **(Système de Détection d'Intrusion basé sur l'algorithme AntClass)**. ***IDSAC*** est un système de détection d'intrusions comportemental, donc il nécessite une phase d'apprentissage qui modélise le comportement normal du système à surveiller, pour cela, nous allons appliquer la méthode *AntClass* sur la base d'apprentissage *KDD*,que nous avons présenté dans le chapitre III. Cette application consiste à classifier les différentes connexions TCP de la base d'apprentissage *KDD*. Il est important de tester notre système ***IDSAC***, pour cela ,nous utiliserons une base de test (base de test *KDD*) contenant des connexions TCP normales et des connexions considérées comme étant des attaques.

IV.6. L'objectif du présent travail

Notre Système a pour but de sécuriser les systèmes informatiques, tout en essayant de satisfaire le maximum des caractéristiques souhaitées d'un IDS qui sont les suivantes :

- il doit fonctionner de manière continue avec une présence humaine minimale
- il doit être capable de détecter des attaques.
- il doit être extensible, on peut donc ajouter des terminaux à sécuriser sans pour autant mettre en péril la sécurité du reste du réseau.
- il doit être capable de superviser un nombre important de stations tout en fournissant des résultats de manière rapide et précise.
- il doit fournir un service minimum de crise, c'est-à-dire que si certains composants de l'IDS cessent de fonctionner, les autres composants doivent être affectés le moins possible

par cette dégradation

IV.7. Structure IDSAC

Le système ***IDSAC*** est structuré de deux grandes phases : (voir figure VI.4)

1. *phase d'apprentissage* : modélise le profil normal de fonctionnement du réseau.

2. *phase de test* : permet de tester le système ***IDSAC***.

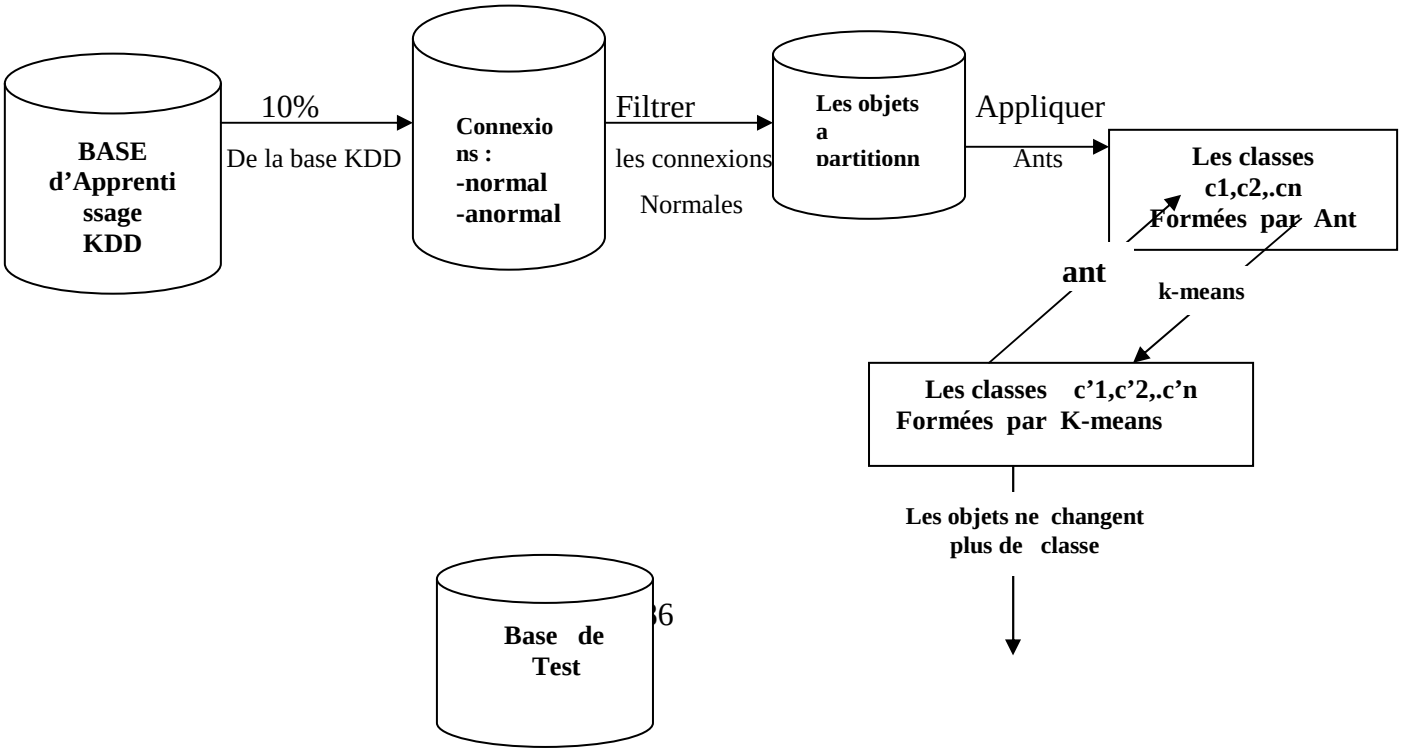
1. Phase d'apprentissage

Pour la modélisation du comportement normal du système, nous traitons 10% de la base d'apprentissage *KDD* (correspondant à 494019 de connexion). Nous devons d'abord construire la base des objets à partitionner, ces objets représentent uniquement les connexions étiquetées normales de la base d'apprentissage *KDD*.

Chaque objet correspond à une connexion, définie par un enregistrement à 41 attributs, tel qu'il est illustré dans le tableau III.2. Cette Base des objets à partitionner est soumise à un classificateur basé sur l'algorithme de *AntClass* , pour former le profil normal de fonctionnement du système.

Après avoir positionné aléatoirement les connexions sur la grille G, nous allons faire évaluer des agents fourmis sur cette grille. Chaque fourmi est représentée par un processus léger qui s'exécute en parallèle avec d'autres processus .l'ensemble des processus se partage les données. Les fourmis sont déposées aléatoirement sur la grille contenant les objets à

partitionner en vérifiant qu'une case ne peut accueillir qu'une seule fourmi à la fois. Les fourmis exécutent trois modules (***ramasser, avancer*** et ***déposer)*** plusieurs fois (***T*** fois) pour former des classes d'objets (C_1,C_2,..C_n) .Les classes construites par les fourmis ne sont pas optimales. Pour corriger cette faille, ces classes sont soumises à l'algorithme *K-Means* pour obtenir une nouvelle partition de classes (C'_1, C'_2,..,C'_n), cet ensemble de classes est à son tour soumis à *Ants* , puis à *K-Means*. Ainsi, les connexions seront regroupées en un ensemble de classes, constituant le profil normal de fonctionnement du réseau.



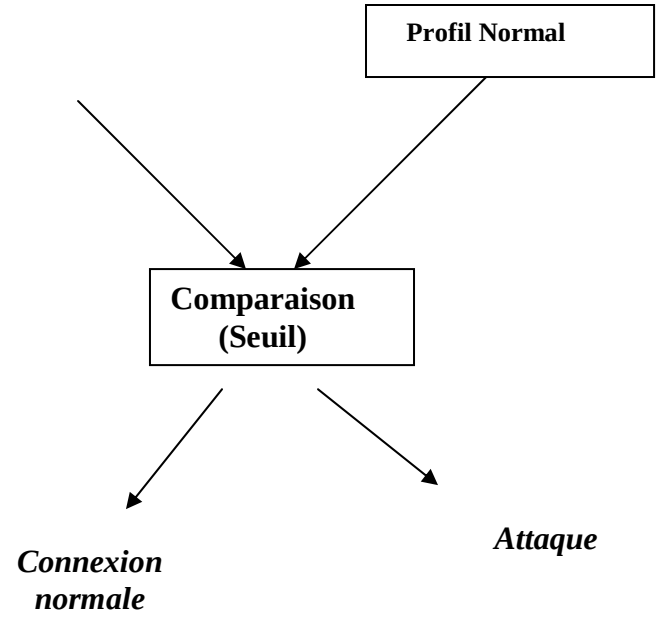


Figure IV.4 *:* Structure IDSAC

2. Phase de test

Dans cette phase, nous utilisons une base de test *KDD*, qui contient des connexions normales et des attaques. Le test se fait en comparant à chaque fois une connexion avec les classes du profil normal obtenu, par le calcul de la distance entre cette connexion et les centres de gravité de l'ensemble des classes du profil normal, ensuite la comparer à un seuil.

1.1. Filtrage des connexions normales

Le premier travail à faire consiste à construire les connexions normales à partitionner. Pour se faire, nous avons proposé le module nommé ***lecture.*** Ce module utilise le fichier d'entrée *nommé* ***baseKDD.txt*** contenant 494019 connexions (97278 connexions normales et 396741 connexions anormales) et génère le fichier contenant uniquement les connexions normales, ce fichier de sortie nommé ***normal.txt.*** Chaque ligne du fichier ***baseKDD.txt,*** représente une connexion. Lorsqu'une ligne est lue par le module *lecture* , il vérifie la valeur de la sous chaîne ***Der*** (où *Der* représente les 6 derniers caractères de la ligne) .Si la valeur

de ***Der*** égale à ***normal*** alors la sous chaîne ***ligne-Der*** est copiée dans le fichier de sortie *normal.txt* **.**

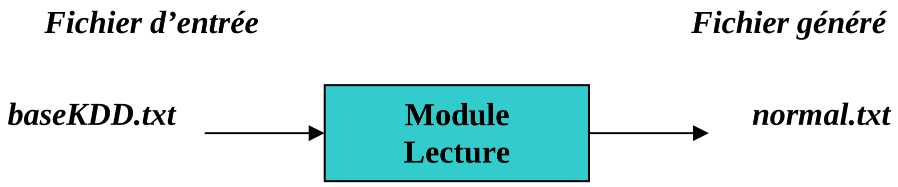


Figure IV.5 *:* Filtrage des connexions normales

Exemple

Si le fichier ***baseKDD.txt*** contient les lignes suivantes :

0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal
0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal
0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal
0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.
0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.
0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.
0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.

Alors le fichier *normal.txt* généré par le module *lecture* contiendra les lignes suivantes:

0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00

0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00

0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00

1.2. Connexion

Chaque connexion étiquetée *normale* de la base d'apprentissage *KDD* est modélisée par une structure de type enregistrement nommée ***connexion,*** contenant 41 champs où que champ représente un attribut.

1.3.Génération du profil normal

Après avoir généré les connexions normales (c'est-à-dire, le fichier *normal.txt)*, nous passons à l'application de l'algorithme *AntClass* . L'algorithme *AntClass* consiste à regrouper les différentes connexions normales (les lignes du fichier *normal.txt*) pour obtenir les classes C_1, C_2..,C_k .Ces classes sont copiées dans un fichier nommé ***phasea.txt*** qui représente le profil normal du système à surveiller.

1.3.1. Grille

L'algorithme *AntClass* utilise une grille sur laquelle les connexions à partitionner et les agents fourmis sont positionnés. Cette grille correspond à un vecteur de **L** éléments ***e*** .

$$\mathbf{L=[\sqrt{2N}\ ]}$$

ou N représente le nombre de connexions à partitionner). Chaque élément ***e*** de ce vecteur correspond à son tour à un vecteur de cases ***c*** . Chaque case ***c*** peut contenir une ou plusieurs connexions de la base KDD et accueillir une et une seule fourmi. .La case ***c*** est représentée par un enregistrement composé de deux champs :le premier champ nommé ***first*** de type *entier*, possédant l'indice du vecteur contenant les connexions de la case ***c*** , le deuxième champ nommé ***second*** est de type *booléen* indiquant si la case ***c*** est occupée par une fourmi.

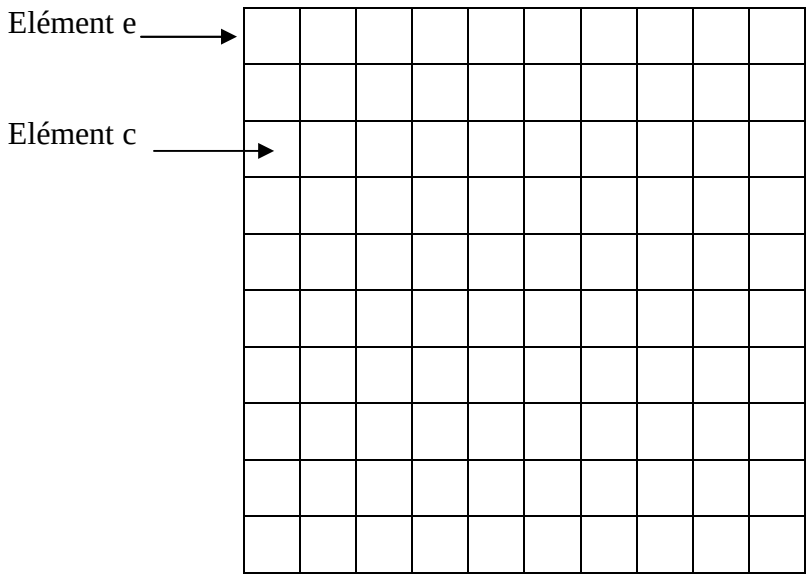


Figure IV.6: grille G

1.3.2. Les agents fourmis

La population d'agents A{a_1,a_2,..a_n} que nous allons utiliser possède les paramètres suivants :

- **Position** : elle représente les coordonnées de la fourmi a_i sur la grille G.
- **Vitesse** : à chaque itération de l'algorithme *AntClass* , la fourmi a_i se déplace aléatoirement sur la grille avec une vitesse $v(a_i)$. Si $(x(a_i),y(ai))$ est la position de la fourmi a_i sur la grille, la nouvelle position après un déplacement sera dans l'intervalle $[x(a_i)-v(a_i),x(a_i)+v(a_i)],[(y(a_i)-v(a_i),y(a_i)+v(a_i)]$.
- **Mémoire** : pour accélérer le processus d'agrégation, chaque fourmi est dotée d'une mémoire de longueur m(a_i). Quand une connexion o_i est ramassée par une fourmi a_i, elle est comparée aux m (a_i) mémoire de la fourmi, elle calcule la distance entre la connexion o_i transporté et les tas qu'elle a mémorisé. Elle se dirige ensuite vers l'emplacement de la connexion la plus similaire à o_i . Lorsque la fourmi a_i transporte plusieurs connexions, la comparaison des connexions est remplacée par la distance entre le centre de gravité de tas transporté et les tas de m(a_i).

- Capacité : représente le nombre de connexion que la fourmi peut transporter en même temps.

-Orientation : chaque fourmi est dotée d'une orientation, les valeurs possibles pour cette propriété sont (voir Figure IV.6) :

- **Nord**
- **Nord Est**
- **Est**
- **Sud Est**
- **Sud**
- **Sud Ouest**
- **Ouest**
- **Nord Ouest**

Chaque fourmi peut changer de direction en tournant de 45 degrés à droite.

Exemple

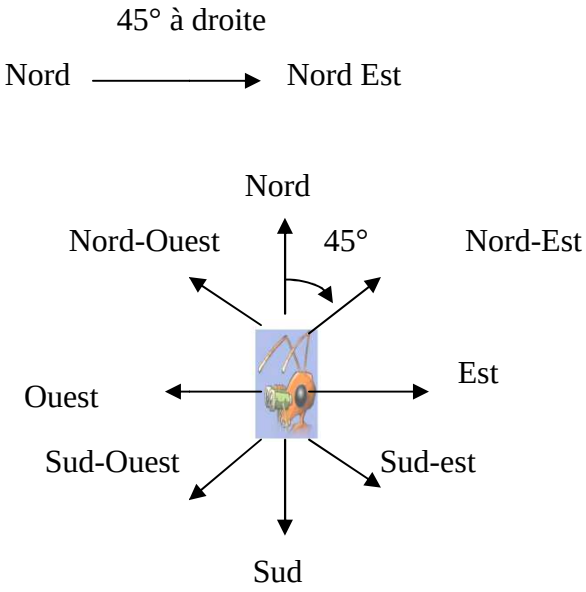


Figure IV.7: Directions d'une fourmi

- **Patience** : lorsque les fourmis ont une capacité supérieure à 1, on peut faire face au problème suivant : toutes les connexions sont transportées, ce qui n'offre plus de possibilité aux fourmis de déposer ce qu'elles transportent. La solution est de doter les fourmis d'une patience qui peut être individuelle. Quand la fourmis a_i effectue plus de $p(a_i)$ déplacement sans avoir réussi à déposer ,le tas transporté, elle le dépose sur la case sur laquelle elle se trouve si cette case est vide, ou l'une de son voisinage, dans le cas contraire.

Toutes ces propriétés peuvent être individuelles.

1.3.3. Déplacement des fourmis

Au début, les fourmis sont disposées aléatoirement sur la grille G. Chaque fourmi se déplace aléatoirement sur la grille avec une vitesse v. La direction choisie par la fourmi

dépend de sa direction précédente : elle a une probabilité égale à 0.6 de continuer tout droit et une probabilité de 0.4 de changer de direction (tourner de 45 degrés à droite).

1.3.4. Ramassage de connexions

Si la fourmi a_i se trouve dans une case contenant des connexions et quelle ne transporte pas de connexions , elle ramasse $c(a_i)$ connexions où $c(a_i)$ représente la capacité de la fourmi a_i. La probabilité p_p de ramasser une connexion est (voir formule IV.1).

1.3.5. Dépôt de connexions

Si la fourmi transporte un ou plusieurs connexions, et qu'elle se trouve sur une case contenant des connexions, la probabilité p_d de déposer les connexions transportés est (voir formule IV.2).

1.3.6. Répartition des connexions sur la grille G

Initialement, les connexions à partitionner sont disposées aléatoirement sur la grille, dans notre travail, nous avons envisagé de disposer les connexions comme suit :

Déposer les connexions par groupe de 10, en commençant à partir de la première colonne de la grille G.

10									
10									
10									
10									
10									

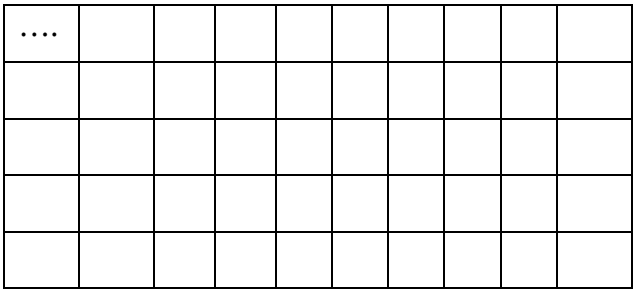


Figure IV.8 : Répartition des connexions sur la grille G

Dans notre algorithme, nous utilisons une population d'agents hétérogènes, c'est-à-dire les agents ont des paramètres individuels différents.

Algorithme Ants

Pour t=1 à T faire
Pour i=1 à A faire
Avancer()
Si la fourmi a_i se trouve sur une case contenant des connexions alors
Si la fourmi a_i ne transporte pas de connexion alors
Ramasser()
Sinon
Deposer()
Finsi
Finsi
Finpour
Finpour retourner la grille G

Algorithme IV.5 : Algorithme Ants (implémentation)

Une fois les fourmis et les connexions sont repartitionnées sur la grille G, les fourmis commencent à se déplacer. Chaque fourmi a_i se déplace avec la vitesse v_i sur une

case non occupée par une autre fourmi en exécutant la méthode ***avancer()*** (voir chapitre V) si cette case contient un tas de connexions et que la fourmi a_i transporte un ou plusieurs connexions , la fourmi a_i dépose un ou plusieurs connexions sur cette case en exécutant la méthode ***Deposer()*** (voir chapitre V). Si la fourmi a_i ne transporte pas d'objets et qu'elle se trouve sur une case contenant un tas d'objets, elle ramasse une ou plusieurs connexions en exécutant la méthode ***Ramasser()*** (voir chapitre V), les objets ramassés sont comparés aux $m(a_i)$ mémoire de la fourmi a_i, ensuite la fourmi a_i se dirige vers l'emplacement des objets les plus similaires aux objets ramassés en exécutant la méthode *avancer()*. Ce processus est répété ***T*** fois où ***T*** représente le nombre d'itérations de l'algorithme *Ants*.

Traitement des connexions isolées

Les connexions isolées sont les objets qui se trouvent sur une fourmi après l'exécution de ***T*** fois l'algorithme *Ants* ou les objets se trouvant seuls sur la grille. Dans notre algorithme, nous avons décidé d'affecter les connexions se trouvant sur la fourmi au tas dont le centre de gravité est le plus proche. Cela est intéressant quand une fourmi est interrompue dans une opération qu'elle pouvait accomplir. Pour le deuxième cas de connexions isolées est traité en donnant plus d'itérations aux fourmis.

1.3.7. Algorithme K-Means

L'algorithme des *K-Means* est l'outil le plus populaire utilisé dans les applications scientifiques et industrielles de Clustering. Le nom dérive du fait que, pour représenter chacune des *K* classes C_k , on utilise le centroide *(*ou centre de gravité*)* (formule IV.1)

L'inertie *intra classe* constitue le critère à optimiser. Elle est définie comme la moyenne des carrés des distances des connexions de la classe au centre de gravité de celle-ci. On cherche ainsi à construire des classes compactes.

L'inertie intra classe associée à la classe C_k s'écrit formellement :

$$I_k = 1/|Ck| \sum_{o_j \in Ck} d^2(o_i, g_k)$$

L'objectif est alors de minimiser la somme de l'inertie intra classe sur l'ensemble des classes.

L'algorithme *AntClass* est interrompu pour lancer le *K–Means* afin d'optimiser le partitionnement construit par les fourmis au sens de *l'inertie intra classe*. Comme les fourmis agissent localement, il peut rester un certain nombre de connexions méritant d'appartenir à une classe dont le centre est beaucoup plus proche. *K-Means* est alors lancé en exécutant la méthode ***centremobile ()*** (voir chapitre V) pour corriger ce type de défaillance. Le *K-Means* est itérativement répété jusqu'à ce que ce critère d'arrêt soit atteint « *aucune modification n'a eu lieu (c'est-à-dire les connexions ne changent plus de classe, ou si le nombre maximum d'itérations a été atteint)* ». Puis reprendre l'algorithme *AntClass* avec une nouvelle partition et des paramètres différents.

Algorithme K-Means

```
iter<-0
change<-1
Tant que ((iter <MAXITERATIONS) ET (change>0)) faire
change<-0
    Pour k<-0 a n-1 faire // n est le nombre de classe
        Pour f<-0 a L(C_K)-1  faire //L(C_k) est la taille de la classe C_k
                mindist<-d(o_f,g_0) // distance euclidienne
                id=0
          Pour j<-1 a  n -1 faire
                dist<-d(o_f,g_j)
           Si  dist< mindist alors
                mindist=dist
                id<-j
            Finsi
         Fin pour

          Si (id !=k)  alors
                change <- change+1
                C_j<-o_f
          Fin si
        Finpour
    Finpour

Iter=iter+1
Fin tantque
```

Algorithme IV.6 : Algorithme *K-Means* (implémentation)

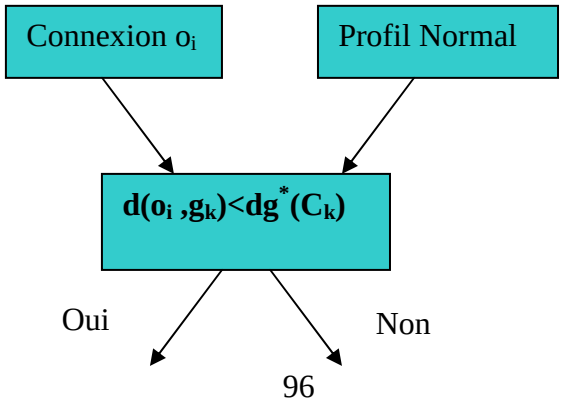
Pour optimiser le temps d'exécution de l'algorithme *K-Means*, nous avons pensé à trier chaque classe C_k en utilisant la distance entre chaque connexion et le centre de gravité de la classe comme critère de tri. Ce tri permet alors de placer au début de la classe les connexions les plus éloignées du centre de gravité.

L'objectif de ce tri est le suivant :

Si l'objet o_i de la classe C_k se trouve dans la classe dont le centre de gravité est le plus proche, alors toutes les connexions de la classe C_k qui viennent après l'objet o_i se trouvent aussi dans la classe dont le centre de gravité est le plus proche, donc on a plus besoin de traiter ces connexions, ceci permet de mettre fin au traitement de la classe C_k et de passer aussitôt au traitement de la classe suivante.

2. Phase de test

Après la phase d'apprentissage, vient la phase de détection d'anomalies (phase de test).Dans cette phase, nous utilisons une base de test *KDD* contenant des connexions normales et des connexions anormales (ou attaques). Chaque connexion de cette base est caractérisée par 41 attributs (voir tableau III.2). Lorsque ***IDSAC*** reçoit une connexion o_i de cette base, il cherche tout d'abord la classe dont le centre de gravité est plus proche de cette connexion , c'est-à-dire, la classe C_k telle que $d(g_k,o_i) < d(g_j,o_i))$ ***avec*** $j=\{1,..,n\}$ où n représente le nombre de classes. Ensuite, il compare $d(g_k,o_i)$ avec $d^*g(C_k)$ (la distance maximale entre les connexions de la classe C_k et son centre de gravité g_k voir formuleIV.9). Si cette distance est inférieure à $d^*g(C_k)$ alors la connexion o_i est normale, sinon la connexion o_i est une attaque. Cette comparaison est implémentée via une méthode nommée ***vérification ( )*** (voir chapitre V).



Normal *Attaque*

Figure VI.9 : Phase de test

Optimisation de l'algorithme K-Means

```
Iter<-0
change<-1
 Tant que ((iter<MAXITERATIONS) ET (change>0)) faire
        change<-0
    Pour k<-0 a n-1 faire //*** n est le nombre de classe
        trier la classe Ck
        change1=1
       int  iter1=0
Tant que ((f<l(ck)-1) ET (change1>0)) faire
        mindis=d(of,g0)
       Id=0
  Pour j<-0 a  n -1 faire
      dist<-d(of,gj)
          Si ( dist< mindist alors
                mindist=dist
                id<-j
         Finsi
  Fin pour
          Si (id !=k)  alors
               change <- change+1
                Cj<-of
        Else
              change1=0
         Fin si
              f=f+1
Fin tantque
Fin pour
```

Algorithme IV.7 : Optimisation de l'algorithme K-Means

Conclusion

Dans ce chapitre, nous avons exposé l'algorithme *Ant Cass* et la conception de notre système de détection d'intrusions basée sur cet algorithme. Cet IDS opère en deux phases. Dans la première phase, nous avons détaillé la manière dont le profil normal du système à surveiller est généré en appliquant l'algorithme *AntClass* sur la base d'apprentissage KDD.

Dans la seconde phase, c'est-à-dire phase de détection d'anomalies, nous avons expliqué comment ***IDSAC*** détecte les anomalies en utilisant le profil normal construit dans la première phase.

Dans le prochain chapitre, nous décrirons l'implémentation, test et validation de ***IDSAC***.

CHAPITRE V.

Implémentation, test et validation de IDSAC

Sommaire :

Partie I: L'implémentation du système de détection d'intrusions IDSAC

Partie I: L'implémentation du système de détection d'intrusions IDSAC

V.1. Introduction

Après une étude détaillée sur les IDS et AntClass et après avoir expliqué en détail la conception de notre premier IDS, nous sommes enfin arrivés à la phase finale de notre premier travail qui consiste à implémenter les différentes phases de notre architecture et tous les modules nécessaires au bon fonctionnement de notre système ***IDSAC***.

V.2. Environnement de développement

Pour l'implémentation de notre IDS, nous avons utilisé le langage Java sous windows. Le langage Java a été développé afin de pouvoir générer des applications indépendantes des machines et de leur système d'exploitation. Une autre caractéristique du langage est de pouvoir écrire des applications structurellement distribuées sur des réseaux. Il appartient, par ailleurs, à la famille des langages objets purs où rien ne peut exister en dehors des classes. Le langage Java possède une syntaxe inspirée du C++.. La puissance du langage Java réside dans la facilité d'utilisation, l'efficacité des programmes. Parmi les caractéristiques fondamentales du langage, nous citons **[30]**

- Java s'affranchit des plates formes : il fonctionne en mode interprété par opposition aux langages compilés et peut s'exécuter sur de nombreux systèmes d'exploitations.
- Java est résolument basée sur la technologie objet et emprunte de nombreux éléments au C++.
- Java propose un mode de fonctionnement adapté aux applications réseaux et Internet.
- Java offre la possibilité de créer des applications multitâches de façon simple.

V.3. Machine et Système d'exploitation utilisés

Pour Programmer et tester ***IDSAC***, nous avons utilisé une machine équipée d'un microprocesseur *Intel Pentium 4 core 2 duo* à *1.86GHz* et d'une RAM de *1GB* avec *windows XP ver2002 service pack 2* comme système d'exploitation.

V.4. Description du logiciel

Notre logiciel est basé sur deux phases principales qui sont : *phase d'apprentissage* qui génère le comportement normal du système et la *phase de détection d'anomalies* qui permet d'assurer la sécurité, en détectant toute intrusion, en s'appuyant sur le résultat de la première phase.

V.4.1. phase d'apprentissage

Notre IDS est basé sur l'approche comportementale, sa mise en œuvre comprend une phase d'apprentissage au cours de laquelle il va découvrir le fonctionnement normal des éléments surveillés. Pour cela, nous allons appliquer l'algorithme *AntClass* pour classifier les connexions normales de la base d'apprentissage KDD, ensuite ces classes seront soumise à l'algorithme *K- Means*, pour obtenir une nouvelle partition, celle-ci sera soumise à son tour à *AntClass* pour obtenir une autre nouvelle classe qui sera à son tour soumise à *K-Means*. La partition finale représentera le profil normal.

V.4.1.1. filtrage des connexions normales

Nous allons travailler sur une base contenant 10% de la base KDD, notre système va utiliser seulement les connexions normales incluses dans cette base, pour réaliser ce filtrage, nous proposons le module nommé ***lecture***.

Lecture()

```
public class lecture {
public static void main(String[] arguments) {
String v;
int pos;
 int nombre=0;
 int compteur=0;
 String w="";
FileWriter fichierSortie  = null;
FileReader monFichier = null;
BufferedWriter tampon1 = null;
BufferedReader tampon = null;
/***************/(suite)
```

```
//*******suite
try
{

fichierSortie = new FileWriter("C:\\IDSAC\\normal.txt",true);
tampon1 = new BufferedWriter(fichierSortie );
monFichier = new FileReader("C:\\KDD\\baseKDD.txt");
tampon = new BufferedReader(monFichier);

        while (true)
{   String ligne = tampon.readLine();
       String d;
 //*** Vérifie la fin de fichier
       if (ligne == null)
            break;
      while (compteur<500)
{
       if (ligne==null)
            break;
int nb=0;int posi=0; pos =ligne.length();
          if (pos>=0)
{
v=ligne.substring(pos-7,pos-1);
        if (v.equals("normal"))
 {
           nombre=nombre+1;
           compteur=compteur+1;
           w=new StringBuffer().append(w).append(ligne).append(";").toString();
}

}
        if(compteur<500)
          ligne= tampon.readLine();

}
```

```
//**********suite
try {
tampon1.flush();
tampon1.close();
monFichier.close();
tampon.close();
fichierSortie .close();
} catch(IOException exception1) {
exception1.printStackTrace();
}
}
System.out.println(" le nombre de ligne est:"+nombre);
}
```

Programme V.1 : Lecture ()

Ce programme lit des pages (blocs) de 500 lignes à partir du fichier d'entrée ***baseKDD.txt***, contenant des connexions normales et des connexions anormales. Il génère le fichier de sortie ***normal.txt*** contenant seulement les connexions étiquetées normales.

V.4.1.2. Application AntClass

- **Grille G**

AntClass utilise une grille sur laquelle sont déposées les connexions et les fourmis. Cette grille est représentée par une matrice de case. Chaque case *c* est représentée à son tour par un enregistrement composé de deux champs. Le premier champ nommé *first* est de type *entier*, le deuxième nommé *second* est de type *booléen*. Chaque case de la matrice peut recevoir une est une seule fourmi et une ou plusieurs connexions. Pour représenter les connexions de différentes cases, nous avons utilisé un vecteur d'éléments V nommé **contenucase .**V correspond à son tour à un vecteur de connexions. Si la case c_i est occupée par une fourmis, alors la valeur de son deuxième champ égale à **occupé** (**second=occupée**). Sinon (**second=non occupée**) indiquant que la case c_i est libre est qu'elle est prête à recevoir une fourmi. Le premier champ de la case c_i est un indice vers un élément de *contenucase* **,** cet élément contient les connexions de la case c_i.

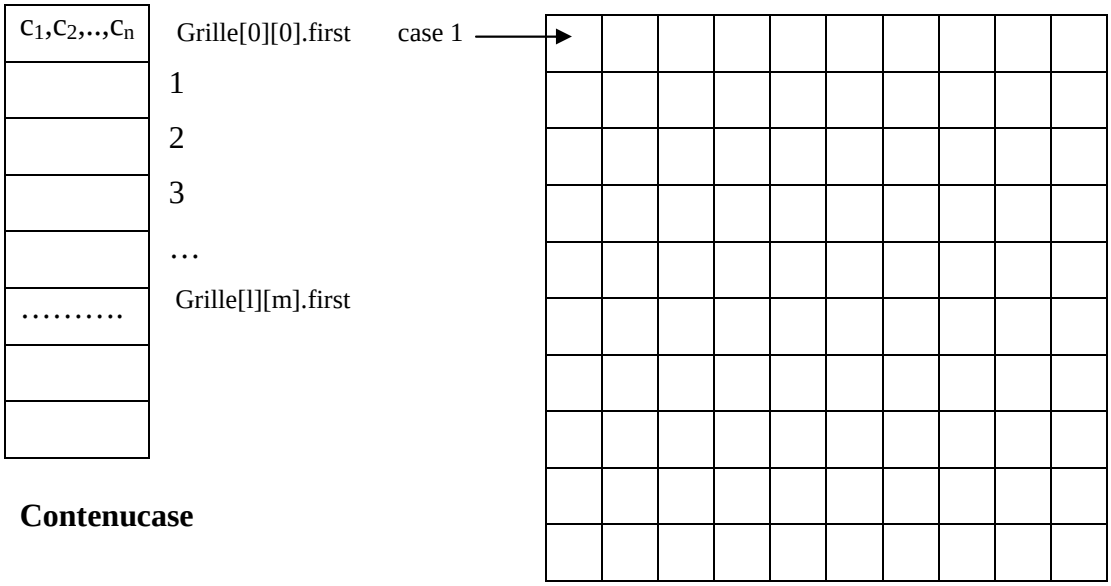
```
public static  cases [][] grille ;
class cases{
        int first;
        String second;
        }

public static Vector [] contenucase;
```

Exemple

Les connexions de la case ***1*** sont les éléments du vecteur suivant :

contenucase [Grille[0][0].first]



Contenucase

Grille

- **Connexions**

Les connexions de la base *KDD* sont caractérisées par 41 attributs (voir tableau III.2).Dans notre programme, chaque connexion est un enregistrement à 41 champs de la classe ***connexion*** , où chaque champ représente un attribut.

```
Class connexion{
   double x1;int x2;int x3;int x4;double x5;double x6;double x7;double
x8;double x9;
   double x10;double x11;double x12;double x13;double x14;double
x15;double x16;
    double x17;double x18;double x19 ;double x20;double x21;double
x22;double x23;
   double x24;double x25;double x26;double x27;double x28;double
x29;double x30;
   double x31;double x32 ;double x33;double x34;double x35;double
x36;double x37;
   double x38;double x39;double x40;double x41;}
```

Connexion(double,int,int,int,double, double) :est la fonction qui permet de créer une connexion contenant comme champs, les paramètres de cette fonction..

- **Fourmis**

Une fourmi est une structure de type thread (annexe B) qui peut s'exécuter en parallèle avec d'autres processus légers. Cette structure contient les champs suivants :

```
Class fourmis extends Thread{
int v;                          //**vitesse
int c;                          //**capacité
int p;                          /**/patience
int m;                          //**longueur de la mémoire
memo [] memoire;                //**mémoire
int x,y;                        //**position  de la fourmi sur la grille
int o;                          //**orientation
Vector objetramas;              //**les objets se trouvant sur la fourmi
                                }
class memo {
        int a;
        int b;
        connexion c;
          }
```

Fourmis (int,int,int,int,int,in,int,vector) :est la fonction qui permet de créer une fourmi.

Chaque fourmi peut exécuter les opérations suivantes :

Programme V.2 : déclaration des fourmis

- **Rammaser( )** :est la fonction qui permet à une fourmi a_i de ramasser une ou plusieurs connexions les plus dissimilaires du tas T_j (T_j est l'ensemble des connexions de la case occupée par la fourmi a_i), selon la probabilité $p_p(Tj)$.Les connexions ramassées par la fourmi a_i sont insérées dans un vecteur nommé **objetramas .**

Probabilité $p_p(T_j)$ est représentée par une fonction nommée ***probarammaser( )*** qui est définit comme suit :

probarammaser()

```
Public double probarammaser()
 {
 double pro;
            //verifier le contenu de la case
                if(contenucase[grille[x][y].first].size()==1)
                {
                        pro=1;
                    return pro;
                 }
// calculer la distance moyenne entre les connexions  de la case et le centre de gravite

double dg_=metod.dis_moy_obj_cent(cotenucase[grille[x][y].first]);

    //*******suite
    if(cotenucase[grille[x][y].first].size()==2)

    {
    // calculer  la distance moyenne entre deux connexions de la case d(σ⁻)
                    double moyen=metod.dis_moyenne(cotenucase[grille[x][y].first]);
                    pro=Math.min(Math.pow(dg_/moyen,k1),1);
                    return pro;
                    }

                else

                   {
    //calculer la distance maximale entre les connexions et le centre de gravite dg*
                    double dg;
                    dg=metod.dis_max_obj_cent(cotenucase[grille[x][y].first]);
                    double slim=(dg_+0.00001)/(dg+0.00001);
```

Programme V.3 : Fonction *probarammaser( )*

dis_moyenne(Vector V) :permet de calculer la distance moyenne entre deux connexions de l'ensemble V (voir formule III.8).

dis_max_obj_cent(Vector V):permet de calculer la distance maximale entre les connexions du vecteur V et son centre de gravité(voir formule III.7).

metod.dis_moy_obj_cent(Vector V):permet de calculer la distance moyenne entre les connexions du vecteur V et son centre de gravité(voir formule III.10).

- **Deposer( ):**est la fonction qui permet à une fourmi a_i de déposer une ou plusieurs connexions suivant la probabilité $b_d(T_i,T_j)$, T_i représente le tas de connexions transporté par la fourmi a_i, T_j est le tas de connexions de la case occupée par la fourmi a_i. Lorsque la fourmi a_i dépose le tas Ti elle mémorise sa position. Probabilité $\mathbf{b_d(T_i,T_j)}$ est représentée par une fonction nommée ***probdeposer( )*** ,elle est définit comme suit :

probadeposer(Vector t)

```
public double probadeposer(Vector t)

 {
double pro;
connexion gra1=new connexion();
connexion gra2=new connexion();
gra1=metod.gravite(contenucase[grille[x][y].first]);
gra2=metod.gravite(t);

//calculer la distance maximale entre les connexions et le centre de gravite dg*
double dg=metod.dis_max_obj_cent(contenucase[grille[x][y].first]);
double  dis=metod.distance(gra2,gra1);

            if (dis<= dg)
```

Programme V.4 : Fonction *probadeposer(Vector t)*

gravite(vector V) :est la fonction qui permet de calculer le centre de gravité des différentes connexions du vecteur V.

- **Avancer( ) :**est la fonction qui permet de déplacer une fourmi a_i avec une vitesse v. Lorsque la fourmi a_i transporte des connexions, elle se dirige vers l'emplacement de connexion le plus similaire aux connexions transportées en consultant sa mémoire m_i. La fourmi a_i gère sa mémoire sous forme d'une liste FIFO ,où les positions les plus anciennes sont oubliées quand la fourmis a_i mémorise de nouvelles positions. Cette mémorisation est effectuée quand la fourmis a_i dépose une ou plusieurs connexions sur la case c_i de la grille G. La mémoire m_i est représentée par un vecteur d'enregistrements, chaque enregistrement est composé de trois champs. Le premier et le deuxième champs sont de type *entier* indiquant la position de la case ci sur laquelle la fourmis a_i dépose le tas transporté, le troisième est de type *connexion* représentant le centre de gravité du tas déposé par la fourmi a_i sur la case c_i.

```
memo [] memoire ;

class memo {
        int a;
        int b;
        connexion c;

              }
```

Programme V.5 : mémoire de la fourmi

- **tourneradroite( )** : est la fonction qui permet à une fourmis a_i de changer de direction , elle est définie comme suit :

tourneradroite()

```
public  void tourneradroite()
{   switch(o)
        {
        case Nord:o=NordEst; break;
        case NordEst:o=Est;break;
        case Est:o=SudEst;break;
        case SudEst:o=Sud;break;
        case Sud:o=SudOuest;break;
        case SudOuest:o=Ouest;break;
        case Ouest:o=NordOuest;break;
        case NordOuest:o=Nord;break;
        }
}
```

Programme V.6 : fonction *tourneradroite()*

- **Les connexions isolées**

Les connexions isolées sont les connexions qui se trouvent sur une fourmi, lorsque cette dernière est interrompue dans une opération quelle pouvait accomplir, ou les connexions non visitées par les fourmis. Dans notre programme, nous avons traité ces deux problèmes comme suit : pour le premier cas, nous avons décidé d'affecter les connexions au tas dont le centre de gravité est le plus proche. Pour le deuxième cas, nous avons donné plus d'itérations aux fourmis.

- **La fonction run( )**

Les agents fourmis sont modélisés par des threads qui héritent de la classe **Java .lang.Thread** (annexe B). Il suffit alors simplement de redéfinir la méthode **run( )** , pour contenir le code des traitements qui sera exécuté par les agents fourmis.

Pour créer et activer un agent fourmi, il faut instancier un objet de la classe threads, et appeler sa méthode **start( ).** Il est obligatoire d'appeler la méthode **start( )** qui va appeler elle-même la méthode **run( )**.

Run( )

```
Public void run()
{
isole=0;
memoire=new memo[m];
//initialiser la memoire

        for(int i=0;i<m;i++)
           {
           memoire[i]=new memo();
           memoire[i].c=new connexion();
           }

       for (int s=0; s<T; s++)


//*********suite
```

```
//******suite

     {
                if ((contenucase[grille[x][y].first].size()!=0) && (isole==0))
                   ramasser();
   alea=Math.random();
   avancer();
                if (objetramas.size()!=0)
                    deposer();
               else
     if (contenucase[grille[x][y].first].size()!=0)
                    ramasser();
    isole=isole+1;

                  if ((isole >=T) && (objetramas.size()!=0))
              { //traiter les connexions isolées
                    connexion a1=new connexion();
                   a1=metod.gravite(objetramas);
                    int min=0;
  for (int f=0;f<cotenucase.length;f++)

                if (cotenucase[f].size()!=0)
                 {
                    min=f; break;
                 }
for(int i=0;i<cotenucase.length;i++)

{
               if (cotenucase[i].size()!=0)
if(metod.distance(a1,(metod.gravite(cotenucase[i])))<=metod.distance(a1,(metod.gravite(c
otenucase[min]))))
                   min=i;

}
for (int k=0;k<objetramas.size();k++)

//inserer  les connexions
  cotenucase[min].add(objetramas.get(k));
//vider  objetramas
  objetramas.removeAllElements();
}
try {
sleep(100);
}
catch(InterruptedException e){};

} }
```

Programme V.7 : Fonction *Run()*

distance (connexion a, connexion b) : est la fonction qui permet de calculer la distance entre les deux connexions a et b.

V.4.1.3. l'algorithme K-Means

Le travail des agents fourmis est interrompus après *T* itérations pour lancer *les centres mobiles* (*K-Means*) , puis reprendre l'algorithme *AntClass* avec la nouvelle partition construite par *K-Means* .

Centres mobile()

```
static public void mobile()
{
int iterations=0;
int itermax=45;
int change=1;
connexion  f=new connexion();
connexion d=new connexion();
Vector  cgrav=new Vector();
      // calculer les centres de gravite
for(int i=0;i<fourmis.cotenucase.length;i++)
            if (fourmis.cotenucase[i].size()!=0)
                cgrav.add(new centreb(i,metod.gravite(fourmis.cotenucase[i])));

       while ((change>0) &&(iterations<=itermax))

 {          change=0;
           iterations=iterations+1;

for(int i=0;i<fourmis.cotenucase.length;i++)
{
            if (fourmis.cotenucase[i].size()!=0)
{
         //trier les  classes
          metod.trie(fourmis.cotenucase[i]);
          int change1=1;
           int k=0;

while (k<fourmis.cotenucase[i].size() && change1>0)

                                          //********suite
```

```
//********suite

{    if(((connexion)fourmis.cotenucase[i].get(k)).traite==false)

                                                    {
            ((connexion)fourmis.cotenucase[i].get(k)).traite=true;
           int min=0;

for(int j=0;j<cgrav.size();j++)
          {

           if(metod.distance((connexion)(fourmis.cotenucase[i]).get(k),((centreb)cgrav.get(min)).l)>
(metod.distance((connexion)(fourmis.cotenucase[i]).get(k),((centreb)cgrav.get(j)).l)))
                min=j;
           }

//verifier si la connexion a changé de place
            if ( ((centreb)(cgrav.get(min))).s!=i)
            {
           change=change+1;

fourmis.contenucase[((centreb)(cgrav.get(min))).s].add((connexion)(fourmis.contenucase[i]).get(
k));
fourmis.contenucase[i].removeElementAt(k);
            }
else

change1=0;

    }

  k=k+1;

}

}
}

// Considérer toutes les connexions non traitées

for(int i=0;i<fourmis.contenucase.length;i++)


//******suite
```

```
      //***suite
    if (fourmis.contenucase[i].size()!=0)
  for(int k=0;k<fourmis.contenucase[i].size();k++)
 ((connexion)fourmis.contenucase[i].get(k)).traite=false;

//calculer le centre de gravite
cgrav.removeAllElements();

for(int i=0;i<fourmis.contenucase.length;i++)

        if (fourmis.contenucase[i].size()!=0)

cgrav.add(new centreb(i,metod.gravite(fourmis.contenucase[i])));
System.out.println("*****************la taille de centregravit est"+cgrav.size());

}

}
```

Programme V.8 : Fonction *centremobile()*

gravite (Vector V) : est la fonction qui calcule et retourne le centre de gravité des connexions du vecteur V.

tr**ie (vector V)** : est la fonction qui permet de trier les éléments de V dans l'ordre décroissant suivant la distance entre l'élément et son centre de gravité.

V.4.1.4. Paramètres utilisés

V.4.1.4.1. Paramètres utilisés pour AntClass

Nous avons appliqué l'algorithme *AntClass* sur la base d'apprentissage et de test *KDD*. Dans notre cas, deux itérations du couple *Ant+ K-Means* sont effectuées. Les paramètres utilisés pour *AntClass* sont les suivants :

- *TAntClass*=2
- pour la première itération de *AntClass* :

  - A=40(nombre de fourmis)
  - T=10000(Nombre d'itérations)
  - $M(a_i) = 0 \ \forall i \in \{1,..,A\}$ {taille de la mémoire}
  - $C(a_i) = 1 \forall i \in \{1,..,A\}$ {capacité de transport }
  - $V(a_i) \in \{1,2,3\} \quad \forall i \in \{1,..,A\}$ {vitesse}
  - $P(a_i) = 1000 \ \forall i \in \{1,..,A\}$ {patience}
  - K1=1
  - K2=1

- pour la deuxième itération de *AntClass* :

  - A=40(nombre de fourmis)
  - T=10000(Nombre d'itérations)
  - $M(a_i) = 5 \ \forall i \in \{1,..,A\}$ {taille de la mémoire}
  - $C(a_i) = \infty \forall i \in \{1,..,20\}$ {capacité de transport }
  - $C(a_i) = 1 \forall i \in \{21,..,40\}$ {capacité de transport }
  - $V(a_i) = 1 \quad \forall i \in \{1,..,A\}$ {vitesse}
  - $P(a_i) = 1000 \ \forall i \in \{1,..,A\}$ {patience}
  - K1=2
  - K2=2

V.4.1.4.2. Paramètres utilisés pour les centres mobiles

Le travail des agents fourmis est interrompus après T itérations (dans notre cas T=10000) pour lancer les centres mobiles. Les centres mobiles est exécuté jusqu'à ce que : le nombre maximum d'itérations a été atteint ou jusqu'à ce que les connexions ne changent plus de classe. Dans notre cas, ce nombre d'itérations maximum est initialisé à 45 (*MAXITERATIONS* = 45).

V.4.2. Phase de détection d'anomalies

Notre IDS doit être capable de différencier entre deux types de connexions (normale, anormale). Lorsque il reçoit une connexion o_i , il calcule la distance entre cette connexion et les centres de gravité de toutes les classes obtenues dans la phase d'apprentissage , ensuite il compare entre la distance (o_i,g_i) et la distance maximale entre les connexions de la classe Ci (gi est le centre le plus proche de la connexion o_i) .

Partie II: Test et résultat

V.5. Résultat de IDSAC sur un jeu de test

Pour tester notre IDS, nous avons constitué un jeu de test formé d'ensembles de connexions appartenant à la base de test KDD. L'avantage de ce jeu de test est que l'on connaît les caractéristiques de ces connexions et cela nous permet d'évaluer la capacité de notre IDS.

Le tableau V.1 présente pour chaque ensemble, le nombre de connexions totales, le nombre de connexions normales et le nombre de connexions anormales.

BASES	***Connexions normales***	***attaques***	***Le nombre total de connexions***
Basetest1	***0***	***5000***	***5000***
Basetest2	***0***	***3060***	***3060***
Basetest3	***0***	***8120***	***8120***
Basetest4	***9675***	***0***	***9675***
Basetest5	***4460***	***1000***	***5460***

Basetest6	***5528***	***3380***	***8908***

Tableau V.1 : Bases de test

Les tableaux V.2 et V.3 présente le résultat obtenu par notre IDS ***IDSAC*** sur le jeu de test qui vient d'être défini.

BASES	***Capacité de notre IDS***
Basetest1	***75.86%***
Basetest2	***85.94%***
Basetest3	***99.71%***
Basetest4	***96.20%***

Tableau V.2 : Résultat sur un jeu de test(avec un seul type de connexions)

Bases	***Capacité de détecter les connexions normales***	***Capacité de détecter les attaques***
Basetest5	**87.24%**	**100%**
Basetest6	**96 .03%**	**64.91%**

Tableau V.3: Résultat sur un jeu de test (avec deux types de connexions)

Les résultats obtenus sur ces bases de données, montrent que le système qu'on vient de réaliser est efficace, plus de 75% d'attaques sont détectées par notre système. Par exemple, pour la ***base1***, ***IDSAC*** détecte 3793 attaques. Cependant, il possède deux autres comportements indésirables qui sont : les faux négatifs et les faux positifs. En effet, ***IDSAC*** ne détecte pas toutes les intrusions existantes dans les bases de données utilisées et il émit aussi des fausses alertes.

Nous pouvons diminuer ces deux comportements indésirables en donnant plus d'itérations aux fourmis(T >10000) et on augmentant le nombre maximum t'itérations de la méthode des centres mobiles (*MAXITERATIONS* >45), certes au détriment du temps de calculs (le temps nécessaire pour générer le profil normal du système à surveiller) .

Conclusion

Durant ce chapitre, nous avons présenté l'implémentation et la mise en oeuvre de notre Premier IDS, en utilisant une méthode de classification non supervisée à base des fourmis *AntClass*. Dans cet IDS, les connexions de la base KDD sont manipulées par les fourmis d'une façon analogue au tri du couvain chez les fourmis réelles .Cela, nous a permis de concevoir un système de détection d'intrusions, qui traite toutes les fonctionnalités et les mécanismes de sécurité définis auparavant. Pour tester le système ***IDSAC***, nous avons utilisé six bases différentes contenant les connexions de la base de *test KDD*.

.

Conclusion générale

La sécurité réseau est un des problèmes les plus sérieux que connaissent les entreprises dotées d'un réseau informatique.

Quoi qu'il en soit, un réseau totalement sécurisé est une utopie, un réseau cent pour cent sécurisé est un réseau fermé auquel personne n'a accès que se soit par voie informatique ou par voie physique.

Il ne sera jamais possible de sécuriser totalement un système d'information, car il y'aura toujours des hackers pour découvrir des nouvelles failles dans le système, mais on peut toujours rendre une intrusion plus difficile en appliquant des nouvelles approches, de ce fait, nous avons proposé dans ce mémoire, une nouvelle solution pour la détection d'intrusions

basée un algorithme à base des fourmis, en effet, nous avons proposé un système de détection d'intrusions réseaux (NIDS) comportemental nommé ***IDSAC « système de détection d'intrusions à base de AntClass »*** est basé sur la méthode non supervisée *AntClass* .

Le système ***IDSAC*** proposé vise à répondre aux objectifs de la sécurité réseau, en s'appuyant sur le tri du couvain chez les fourmis réelles. ***IDSAC*** nécessite une phase d'apprentissage, pour cela, nous avons appliqué l'algorithme *AntClass* sur la base d'apprentissage et de test *KDD* pour construire le profil normal du système à surveiller. Afin de tester ***IDSAC*** , nous avons utilisé une base de test appelée base de test KDD contenant des connexions normales ,et des connexions considérées comme étant des attaques de type (DOS, Probing, R2L, URL) . Cette base de test est très appropriée pour évaluer un système de détection d'intrusions de type comportemental, puisqu'elle contient des attaques qui ne figurent pas dans la base d'apprentissage KDD.

Nous avons implémenté ***IDSAC*** en utilisant le langage Java (threads) *sous Windows*.

Ce projet nous a été d'un grand apport pédagogique, puisqu'il nous a permis de découvrir la sécurité informatique, ses fonctionnalités et ses mécanismes, de nous familiariser avec les threads de Java et de savoir comment résoudre les problèmes de classification en s'appuyant sur le comportements des fourmis.

Perspectives

Il semble que, quelles que soit les démarches suivies pour aborder un problème, on ne pourra atteindre complètement les objectifs initiaux, qui sont très variés et qu'ils ne peuvent pas être tous satisfaits dans notre travail, C'est pourquoi, nous proposons :

- Améliorer les systèmes pour récupérer les vrais paquets réseau, car nos systèmes fonctionnent seulement avec des données synthétiques.
- Réaliser et tester d'autres IDS en utilisant d'autres méthodes de classification basée sur des méthodes d'Intelligence Artificielle (algorithmes génétiques, etc...).

- Coupler le système ***IDSAC*** avec un IDS de type scénario pour pouvoir détecter les attaques inconnues.

ANNEXE A.

Classification

A.1. Différentes classification possibles [40]

Classifier est un processus qui permet de rassembler des objets dans des sous ensembles tout en conservant un sens dans le contexte d'un problème particulier. Les sous ensembles obtenus représentent une organisation. Ou encore une représentation de l'ensemble des objets. Les relations disponibles entre les objets sont rassemblées dans une *matrice de dissimilarite* dans la quelle les lignes et les colonnes correspondent aux objets. Comme les objets peuvent être représentés par des points dans un espace numérique à M dimensions, la dissimilarite entre deux objets peut être définie comme une distance entre les deux point correspondants . Cette matrice de dissimilarite est la principale entrée nécessaire a la phase de classification. La figure1 présente les différentes méthodes de classification (jain and dubes,1988).Voici une description de ces méthodes :

A.1.1. Classification non exclusive [40]

Une classification non exclusive autorise qu'un objet appartienne à plusieurs classes simultanément. Les classes peuvent alors se recouvrir.

A.1.2 Classification exclusive

Une classification exclusive est un partitionnement des objets : un objet n'appartient qu'a une classe et une seule.

A.1.2.1. Classification supervisé

Les objets sont étiquetés tout en connaissant leurs disimilarités. Le problème consiste alors à construire des hyper plans séparant les objets selon leur classe.

A.1.2.2 Classification non supervisé

Une méthode de classification non supervisé (en anglais clustering) n'utilise que la matrice de dissimilarité. Aucune information sur la classe d'un objet n'est fournie à la méthode (on dit que les objets ne sont pas étiquetés).L'objectif de la classification non supervisé est opposé au cas supervisé. Dans le premier cas , il s'agit de découvrir des groupes d'objets alors que dans le deuxième cas il s'agit d'utiliser les groupes connus pour découvrir ce qui les différencie a fin de pouvoir classer de nouveaux objets dont la classe n'est pas connue. Prenons l'exemple d'une population d'individus dont on dispose de renseignement tels que l'age, le sexe, le poids, la taille et le taux de cholestérol. Une méthode de classification non supervisée permettra par exemple de construire deux groupes : un groupe végétariens et un groupe de carnivores. Une méthode de classification supervisée nécessite de savoir pour chaque individu s'il est végétarien ou non et permettra d'établir des critères permettant de différencier un individu selon ses attributs (age, sexe,…).

- **Classification hiérarchique** : une méthode de classification hiérarchique construit une séquence de partitions imbriquée, que l'on visualise par exemple par un dendogramme (Figure 2).

- **Partitionnement :** un partitionnement ne construit qu'une seule partition des données.

Pour concevoir notre IDS, nous nous intéressons qu'au problème du partitionnement en laissant de côte la classification hiérarchique puisque nous ne la traitons pas avec les fourmis artificielles. Alors que la classification hiérarchique est surtout utilisée pour construire des taxonomies dans le domaine de la biologie, des sciences comportementales ou sociales. Les techniques de partitionnement sont surtout utilisées quand l'obtention

d'une partition unique est primordiale, par exemple pour la représentation et la compression de bases de données importantes[3].

Le problème de partitionnement peut être formulé de la façon suivante. Etant donnés N points dans un espace métrique à M dimensions, on cherche une partition de ces points en K classe (ou groupe) de telle sorte que les points d'un même groupe soient plus similaires entre eux qu'avec les points les points des autres groupes. La valeur de K peut ne pas être connue. Comme pour toute méthode non supervisée , la qualité d'une méthode de partitionnement ne peut être jugée qu' a partir de ses résultats : c'est l'expert du domaine qui peut déterminer la pertinence des résultats obtenus.

A.2 Distance

Etant donnée un ensemble $O= \{o_1,\ldots,o_N\}$ de N objets correspondant chacun à un point d'un espace métrique à M dimensions dont les coordonnées sont notées par le vecteur $x_i=(x_{i1},\ldots,x_{iM})$ pour l'objet o_i, on peut définir la distance conne suit :

Une distance est assimilable à une mesure de dissimilarité. Etant donné que nous nous intéressons à des espaces métriques. L'indice de dissimilarité le plus communément utilisé est **la métrique de Minkowski** :

$$\mathbf{d_r(o_i,o_j)=d_r(x_i,x_j)=\left(\sum_{k=1}^{M} w_k |x_{ik}-x_{jk}|^r\right)^{1/r}}$$

$\mathbf{w_k}$ est le facteur de pondération , dans notre thèse nous considérons w_k égal a 1. Suivant la valeur de r ($r\geq 1$) on obtient les mesures suivantes :

- r=1 : distance de Manhattan.
- r =2 : distance euclidienne.
- $r =\infty$: $d_\infty(xi,xj)=\text{Max}_{1\leq k\leq M} |x_{ik}-x_{jk}|$.

Ces mesures sont souvent utilisées pour des données numériques. Dans le cas de données symboliques, d'autres distances doivent être utilisées. La plus connue étant la distance de **Hamming**, à l'origine définie pour mesurer la distance entre des chaînes binaires, et qui correspond au nombre de bits différents dans les deux chaînes pour une même position.

Dans notre thèse, nous noterons la mesure d2(.) par d(.) pour alléger les notations.

[3] La représentation d'une classification hiérarchique par un dendogramme est particulièrement interdite quant la base de
données se compose de plusieurs centaines d'objets.

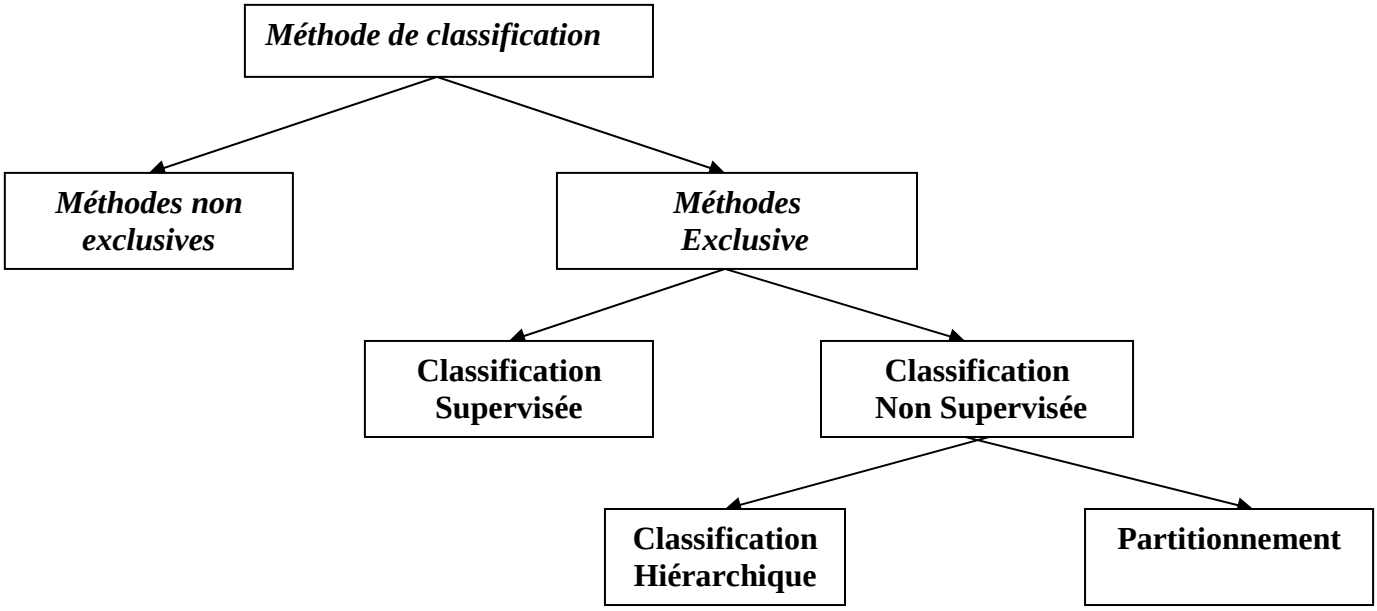


Figure 1. Les méthodes de classification [40]

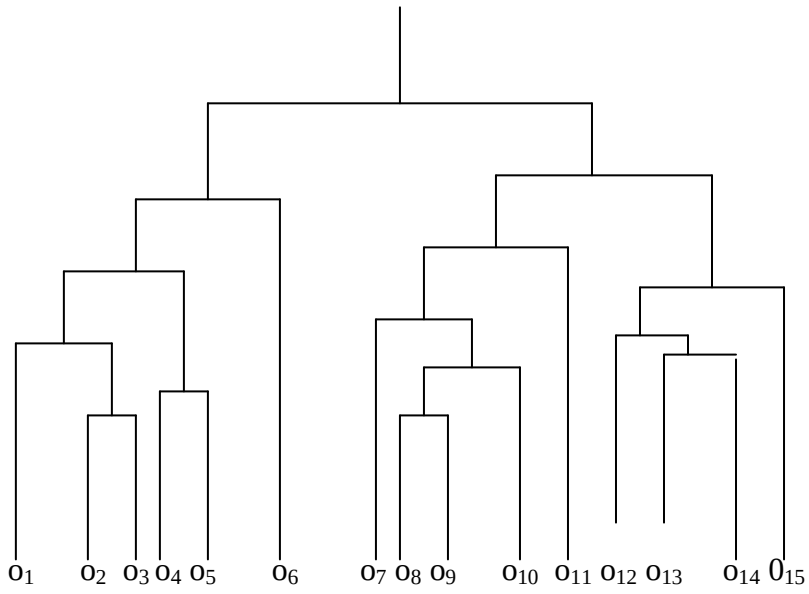


Figure 2. Exemple de dendrogramme [40].

ANNEXE B.

Les threads

B.1 introduction

Un thread est une unité d'exécution faisant partie d'un programme. Cette unité fonctionne de façon autonome et parallèlement à d'autres threads. En fait, sur une machine mono processeur, chaque unité se voit attribuer des intervalles de temps au cours desquels elles ont le droit d'utiliser le processeur pour accomplir leurs traitements.

La gestion de ces unités de temps par le système d'exploitation est appellée scheduling. Il existe deux grands types de scheduller:

- le découpage de temps : utilisé par Windows et Macintosh OS jusqu'à la version 9. Ce système attribue un intervalle de temps prédéfini quelque soit le thread et la priorité qu'il peut avoir
- la préemption : utilisé par les systèmes de type Unix. Ce système attribut les intervalles de temps en tenant compte de la priorité d'exécution de chaque thread. Les threads possédant une priorité plus élevée s'exécutent avant ceux possédant une priorité plus faible.

B.2. Création de threads avec Java [30]

Threads gérés dans différents langages. Gestion lourde avec ***C*** et simplifiée avec ***Java*** qui propose deux solutions :

- Objet héritant de la classe **`java.lang.Thread`** Deux objets de cette classe peuvent s'exécuter directement en concurrence.

- Objet implémentant l'interface **`java.lang.Runnable`**

  – On doit définir une méthode `run()` ;

  – On range ces objets dans des objets Threads (enveloppes) qui s'exécutent concurremment.

B.2.1 La dérivation de la classe Thread

Le plus simple pour définir un thread est de créer une classe qui hérite de la classe java.lang.Thread. Il suffit alors simplement de redéfinir la méthode run() pour y inclure les traitements à exécuter par le thread.

Exemple:

```
public class MonThread2 extends Thread {

  public void run() {
    int i = 0;
    for (i = 0; i < 10; i++) {
      System.out.println("" + i);
    }
  }

}
```

Pour créer et exécuter un tel thread, il faut instancier un objet et appeler sa méthode start(). Il est obligatoire d'appeler la méthode start() qui va créer le thread et elle-même appeler la méthode run().

Exemple

```
public class MonThread2 extends Thread {

  public static void main(String[] args) {
            Thread t = new MonThread2();
            t.start();
  }
```

```
  public void run() {
    int i = 0;
    for (i = 0; i < 10; i++) {
      System.out.println("" + i);
    }
  }

}
```

Implémentation de l'interface Runnable

Si on utilise l'interface Runnable , il faut uniquement redéfinir sa seule et unique méthode run() pour y inclure les traitements à exécuter dans le thread.

Exemple

```
public class MonThread3 implements Runnable {

  public void run() {
    int i = 0;
    for (i = 0; i < 10; i++) {
      System.out.println("" + i);
    }
  }
}
```

Pour pouvoir utiliser cette classe dans un thread, il faut l'associer à un objet de la classe Thread. Ceci ce fait en utilisant un des constructeurs de la classe Thread qui accepte un objet implémentant l'interface Runnable en paramètre.

Exemple

```
public class LancerDeMonThread3 {

  public static void main(String[] args) {
    Thread t = new Thread(new MonThread3());
    t.start();
  }
}
```

Il ne reste plus alors qu'à appeler la méthode start() du nouvel objet.

B.3. Les méthodes de gestion des threads [30]

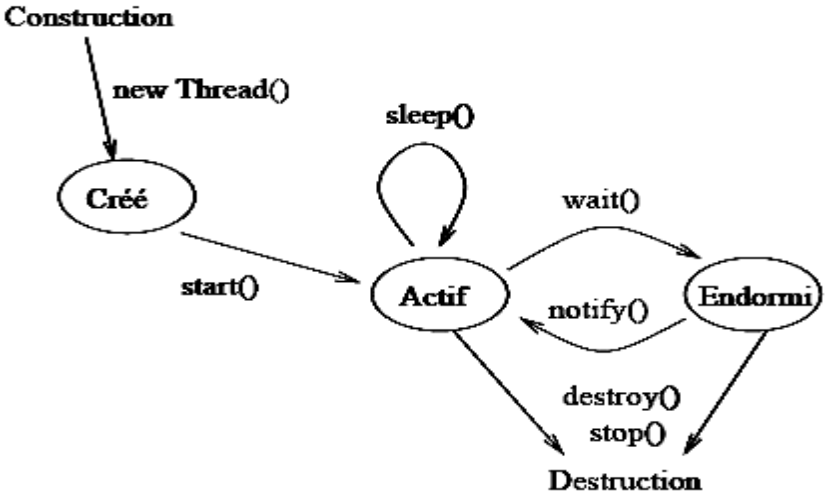


Figure 1. : Cycle de vie d'un thread

B.3 Thread en tâche de fond (démon) [30]

Il existe une catégorie de threads qualifiés de démons : leur exécution peut se poursuivre même après l'arrêt de l'application qui les a lancés.

Une application dans laquelle les seuls threads actifs sont des démons est automatiquement fermée.

Le thread doit d'abord être créé comme thread standard puis transformé en demon par un appel à la méthode setDaemon() avec le paramètre true. Cet appel se fait avant le lancement du thread, sinon une exception de type IllegalThreadStateException est levée.

B.4 Exclusion mutuelle [30]

Chaque fois que plusieurs threads s'exécutent en même temps, il faut prendre des précautions concernant leur bonne exécution. Par exemple, si deux threads veulent accéder à la même variable, il ne faut pas qu'ils le fassent en même temps.

Java offre un système simple et efficace pour réaliser cette tache. Si une méthode déclarée avec le mot clé synchronized est déjà en cours d'exécution, alors les threads qui en auraient également besoin doivent attendre leur tour.

Le mécanisme d'exclusion mutuelle en Java est basé sur le moniteur. Pour définir une méthode protégée, afin de s'assurer de la cohérence des données, il faut utiliser le mot clé synchronized. Cela crée à l'exécution, un moniteur associé à l'objet qui empêche les méthodes déclarées synchronized d'être utilisées par d'autres objets dès lors qu'un objet utilise déjà une des méthodes synchronisées de cet objet. Dès l'appel d'une méthode synchronisée, le moniteur vérouille tous les autres appels de méthodes synchronisées de l'objet. L'accès est de nouveau automatiquement possible dès la fin de l'exécution de la méthode.

Ce procédé peut bien évidemment dégrader les performances lors de l'exécution mais il garantit, dès lors qu'il est correctement utilisé, la cohérence des données.

B.4.1 Sécurisation d'une méthode [30]

Lorsque l'on crée une instance d'une classe, on crée également un moniteur qui lui est associé. Le modificateur synchronized place la méthode (le bloc de code) dans ce moniteur, ce qui assure l'exclusion mutuelle.

Le méthode ainsi déclarée ne peut être exécutée par plusieurs processus simultanement. Si le moniteur est occupé, les autres processus seront mis en attente. L'ordre de réveil des processus pour accéder à la méthode n'est pas prévisible.

Si un objet dispose de plusieurs méthodes synchronized, ces dernières ne peuvent être appelées que par le thread possédant le verrou sur l'objet.

B.4.2 Sécurisation d'un bloc [30]

L'utilisation de méthodes synchronisées trop longues à exécuter peut entrainer une baisse d'efficacité lors de l'exécution. Avec java, il est possible de placer n'importe quel bloc de code dans un moniteur pour permettre de réduire la longueur des sections de code sensibles.

Exemple

```
synchronized  void methode1() {
  // bloc de code sensible
  ...
}

void methode2(Object obj) {
  ...
  synchronized (obj) {
```

```
    // bloc de code sensible
    ...
  }
}
```

ANNEXE C.

Système de détection d'intrusions IDSAC

Pour faciliter l'utilisation de notre IDS, nous avons réalisé une interface graphique via *Microsoft visual studio version 2005*, cette interface est composée des fenêtres suivantes :

- ***Première fenêtre*** : elle contient deux boutons :

  Quitter : permet de quitter l'application.

  Lancer : permettant de lancer notre IDS, il fait appel à « la fenêtre principale ».

- ***Fenêtre principale :*** contient le bouton **phase d'apprentissage** et le bouton **phase de test**. Le premier bouton fait appel à la fenêtre « profil normal » par contre, le deuxième fait appel à la fenêtre « phase de test »

- ***Phase de test :*** elle contient une barre de titre et une barre des menus :

1. ***Fichier*** : contient les commandes suivantes :

   - Nouveau : permet de créer un nouveau fichier de connexions pour tester le système.

- Ouvrir : permet de consulter le fichier crée.
- Fermer : permet de fermer la fenêtre « phase de test ».

2. ***Consuler*** : ce menu permet de consulter les différentes bases utilisées dans notre mémoire pour évaluer la capacité de notre système de détection d'intrusions.
3. ***analyser*** : permet d'analyser les connexions appartenants aux bases.

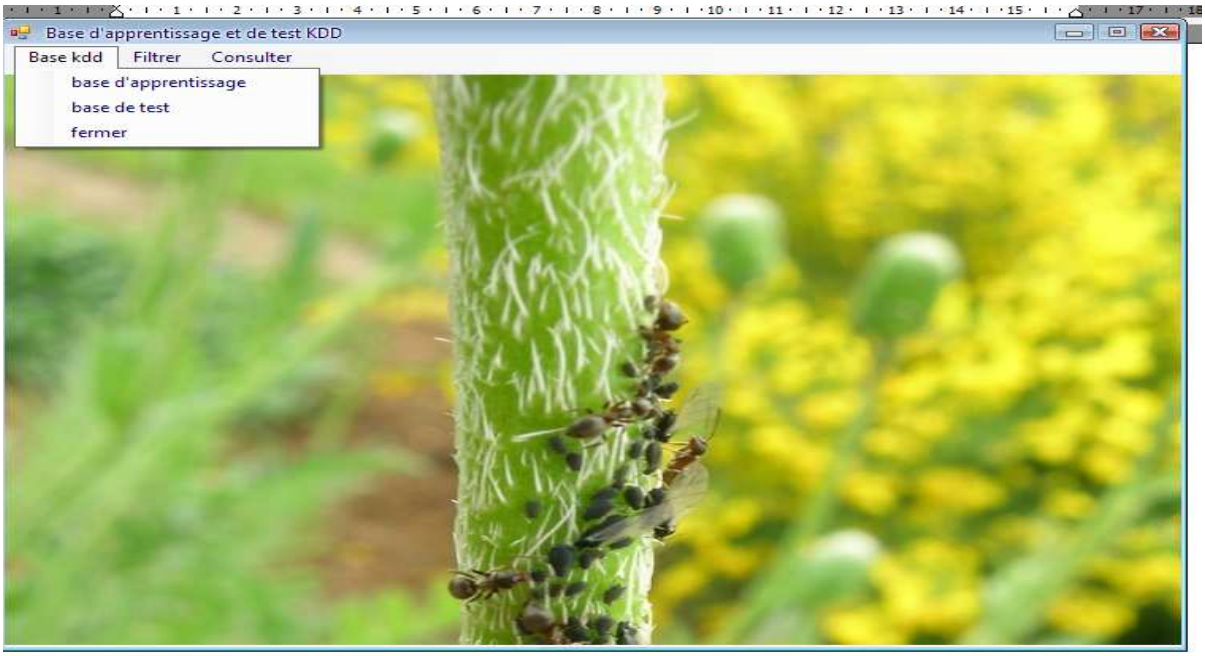
- ***phase d'apprentissage*** : cette fenêtre permet de générer le profil normal du système à surveiller pour cela, l'utilisateur doit posséder un mot de passe valide pour pouvoir y accéder.

Cette fenêtre est composée de trois boutons principaux :

- Générer le profil normal
- Consulter le profil normal
- Base d'apprentissage et de test : permet d'activer la fenêtre « base d'apprentissage et de test KDD »

- ***Base d'apprentissage et de test KDD :***: permet de :

  - Consulter la base d'apprentissage KDD.
  - Consulter la base de test KDD.
  - Filtrer les connexions normales de KDD.
  - Filtrer les connexions anormales de KDD.


Base d'apprentissage et de test KDD
Base kdd
Filtrer
Consulter
base d'apprentissage
base de test
fermer

Bibliographie

[1] **:** Pascal Nicolas. « Cours de réseaux maîtrise d'informatique université d'Angers »

[2] **:**Douglas E. Comer. Prentice-Hall.(1993). « Internetworking with TCP/IP Volume I».

[3] :Pierre Erny.(1998). « les réseaux informatiques d'entreprise ».

[4] :Kenza Tayeb.(2006) « *Détection d'intrusion coopérative basée sur la fusion de données* ». Thèse de Magister de l'INI ,2006.

[5] :Karim Tamine, « *Sécurité dans les réseaux* ». Cours MASTER2 (recherche) –Informatique- Décembre 2004.

[6] **:** Karima Boudaoud.(2002) « *Détection d'intrusions : Une nouvelle approche par système multiagents* ». Thèse de doctorat de l'Université de Genève. 2002.

[7] **:** Introduction et Initiation a la sécurité informatique. « SecuriteInfo.com ».

[8] **:** S.Northcut, J.Novak, D.Mclachlan. (2001). « Détection des intrusions réseaux ».

[9] **:** Tarek Abbes.(2004) « Classification du trafic et optimisation des règles de filtrage pour la détection d'intrusions ». Thèse de doctorat de l'université Henri Poincaré.Nancy1 .2004.

[10] : ***http://www.e-atlantide.com/securite/*** puis cliquer sur détection intrusions.

[11] **:** Yacine Bouzida.(2006). « Application de l'analyse en composante principale pour la détection d'intrusion et détection de nouvelle attaques par apprentissage supervisé » Thèse de doctorat de l'Université de rennes. 2002.

[12] **:** Yacine Bouzida and Sylvain Gombault « Profils Propres pour la Détection d'Intrusion ».

[13] **:** Ghenima Bourkache(2006/2007). « Un IDS réparti basé sur une société d'agents intelligents » .thèse de magister Université de Boumerdes (2006/2007).

[14] **:** G. Zémor.(2000) . « Cours de cryptographie ».

[15] **:** Douglas Stinson. (2003) *« Cryptographie, théorie et pratique »*. (Présentation claire des mathématiques de la cryptographie).

[16] **:** Introduction à la Cryptographies Printed in the United States of America.
[Traduction française : news :fr.misc.cryptologie, 1998] Copyright 1990-1998 Network Associates, Inc. and its Affiliated Companies. All Rights Reserved. PGP*, Version 6.0.2.

[17] **:** Article écrit par Jean-François : « politique d'un intrus ».

[18] **:** L. Spitzner. Honeypots : Catching the insider threat. In Proceedings of the 19th Annual Computer Security Applications Conference, page 170. IEEE Computer Society, 2003.

[19] **:** J. Justen. Nessus 2.0.8. Technical report, Network and Systems Professionals Association Inc., Novembre 2003.

[20] **:** R. F. Puppy. A look at whisker's anti-ids tactics,
Url : http ***://www.wiretrip.net/rfp/pages/whitepapers/whiskerids.html*** ,1999

[21] **:** Saint Corporation. Saint documentation contents,
Url ***: http ://www.saintcorporation.com/.***

[22] **:** Prévention et détection d'intrusion issu de comment sa marche.
http ://www. CommentCaMarche/.

[23] :Benjamin Morin .(2004). « Corrélation d'alertes issues d'outils de détection d'intrusions avec priseen compte d'informations sur le système surveillé ». ». Thèse de doctorat de l'Université de Rennes. 2004.

[24] : J. Anderson, « *Computer security threat monitoring and surveillance* ». 1980.

[25] : Dorothy E. Denning. « *An intrusion detection model* ». IEEE Transactions on software engeneering, SE-13 :222–232, 1987.

[26] :Philipe Biondi : « *Architecture expérimentale pour la détection d'intrusions dans un système informatique* ». Avril-Septembre 2001.

[27] :R.Heady, G.Luger, A.Maccabe et M.Sevilla. « *The architecture of a network level intrusion detection system* ». Août 1990.

[28] : H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion-detection systems.Annales des Télécommunications, 55(7_8) :361_378, 2000.

[29] :Ludovic Mé.2002. « Audit de sécurité par algorithmes génétiques" Thèse de doctorat de l'Université Rennes 1. 2002.

[30] : Bruce Eckel « penser en java » . ***http ://www.bruceeckel.com*** .2001.

[31] **:** David Burgermeister, Jonathan Krier « Les systèmes de détection d'intrusions ».

[32] : Klauss Muller, « *IDS : Système de détection d'intrusion, partie I* ». LinuxFocus article number 292. Url : ***http://linuxfocus.org.***

[33] : Ludovic Mé et Véronique Alanou, « *Détection d'intrusion dans un système informatique :*

Methodes et outils ».

[34] : ***http://www.snort.org***

[35] : *http://www.marlboro.edu/ ttoomey/benids*

[36] : *http://hank.sourcrforge.net/*

[37] : *http://www.prelude-ids.org*

[38] : *http://www.scaramangna.co.uk/restorm/.*

[39] : *http://www.icir.org/ver/bro-info.html*

[40] : Nicolas Monmarché.(2000) « *Algorithmes de fourmis artificielles : applications à la classification et à l'optimisation* ». Thèse de doctorat de l'université de Tour. Décembre 2000.

[41] : E.Lumer et B.Faieta, « *Diversity and Adaptation in Populations of Clustering Ants* ». In (Cliff et al., 1994), pages 501–508.

[42] : N. Ben Amor, S. Benferhat et Z. Elouedi, « *Réseaux bayésiens naïfs et arbres de décision dans les systèmes de détection d'intrusions* ».

[43] : *http://www.tripwire.com/products/index.cfml*

[44] : *http://freshmeat.net/redir/swatch/10125/url_homepage/swatch*

[45] : *http://www.enterasys.com/ids/squire/*

[46] : *http://freshmeat.net/redir/tiger-audit/30581/url_homepage/tiger*

[47] : *http://www.netiq.com/products/sm/default.asp*

[48] : réseaux et protocoles issu de comment sa marche,
http ://www.CommentCaMarche/.

[49] : Valdes, A., Skinner K.: Adaptive Model-based Monitoring for Cyber Attack Detection. In proceedings of Recent Advances in Intrusion Detection (RAID 2000), Toulouse, France, 80-92, 2000.

[50] **:** Quinlan, J. R.: Induction of decision trees. Machine Learning 1, 1-106, 1986.

[51] **:** Quinlan, J. R.: C4.5, Programs for machine learning. Morgan Kaufmann San Mateo Ca, 1993.

[52] ***:http://www.chimique.usherbrooke.ca/cours/gch445/neurones-intro.html***

[53] **:** Warren McCulloch & Walter Pitts, *A Logical Calculus of Ideas Immanent in Nervous Activity*, 1943, Bulletin of Mathematical Biophysics 5:115-133

[54] **:** ***http://labo.algo.free.fr/neuro/reseau_de_neurones_artificiel.html***

[55] **:** ***http://fr.wikipedia.org/wiki/Nmap***

[56] : Florian BOITREL Maxime CHAMBREUIL Tuteur : M. Philippe Leray (2003) « Détection d'intrusions réseaux par l'utilisation de cartes de Kohonen et d'un réseau de Neurones » . juin 2003.

[57] : Yacine Bouzida, Frédéric Cuppens, Sylvain Gombault. « Détection de nouvelles attaques ».

[58] **:** ***http://www.commentcamarche.net/contents/outils-reseau/ping.php3***

[59] **:** ***http://www.commentcamarche.net/contents/outils-reseau/traceroute.php3***

[60] ***:*** Ludovic Mé, Zakia Marrakchi, Cédric Michel « La détection d'intrusions : les outils doivent coopérer ».

[61] : Odile Papini . « détection d'intrusions ».

[62] :http://kdd.ccs.uci.edu/databases/kddcup99/task.html