Classification du trafic et optimisation des règles de filtrage pour la détection d'intrusions

Tarek Abbes

Département de formation doctorale en informatique
UFR STMIA

École doctorale IAEM Lorraine

Classification du trafic et optimisation des règles de filtrage pour la détection d'intrusions

THÈSE

présentée et soutenue publiquement le 14 décembre 2004

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Tarek ABBES

Composition du jury

Rapporteurs :	Marc DACIER	Professeur à l'Institut Eurécom, Nice
	Hervé DEBAR	Habilité à diriger des recherches, France Télécom, Caen
Examinateurs :	Adel BOUHOULA	Maitre de Conférences - Habilité, SUP'COM, Tunis
	Frédéric CUPPENS	Professeur à l'ENST-Bretagne
	Claude GODART	Professeur à l'Université Henri Poincaré, Nancy 1
	Michael RUSINOWITCH	Directeur de Recherche INRIA, Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Résumé

La cybercriminalité augmente de plus en plus ces dernières années ce qui pousse les administrateurs à sécuriser convenablement leurs réseaux informatiques. Ils ont recours à la détection d'intrusions pour détecter les attaques à l'entrée du réseau ou déceler les intrusions passées inaperçues à travers les pare-feux et les routeurs filtrants.

Dans cette thèse, nous nous intéressons à des problèmes sensibles rencontrés par la détection d'intrusions à savoir le haut débit, les techniques d'évasion et les fausses alertes. Afin de soutenir le haut débit, nous procédons à une division du trafic réseau ce qui permet de répartir la charge d'analyse sur plusieurs systèmes de détection d'intrusions et de sélectionner pour chaque classe du trafic la meilleure méthode de détection. Par ailleurs nous réduisons le temps de traitement de chaque paquet en organisant convenablement les règles de détection d'attaques stockées sur les systèmes de détection d'intrusions. Au cours de cette analyse nous proposons un filtrage à la volée de plusieurs signatures d'attaques. Ainsi, nous évitons le réassemblage du trafic qui était auparavant nécessaire pour résister aux techniques d'évasion. Enfin nous assurons une analyse protocolaire avec des arbres de décisions pour accélérer la détection des attaques et éviter les inconvénients du filtrage brut de motifs tels que la génération abondante des faux positifs.

Mot clés : Détection d'intrusions, division du trafic réseau, organisation des règles, filtrage à la volée des motifs, analyse protocolaire.

Abstract

Today's, because of the fast cyber-criminality development, administrators show an increasing interest towards network security. Specially they are interested by the intrusion detection in order to complement the firewall supervision and thus discover more undetected attacks.

In this dissertation we are interested by some bottlenecks that the intrusion detection faces, namely the high load traffic, the evasion techniques and the false alerts generation. In order to ensure the supervision of overloaded networks, we classify the traffic using Intrusion Detection Systems characteristics and traffic properties. Therefore each IDS supervises less IP traffic and uses less detection rules (with respect to traffics it analyses). In addition we reduce the packet processing time by a wise attack detection rules application. During this analysis we rely on a fly pattern matching strategy of several attack signatures. Thus we avoid the traffic reassembly previously used to deceive evasion techniques. Besides, we employ a protocol analysis with decision tree in order to accelerate the intrusion detection and reduce the number of false positives noticed when using a raw pattern matching method.

Keywords : Intrusion detection, network traffic division, rules organisation, on the fly pattern matching, protocol analysis.

Remerciements

Ce travail a été réalisé au sein de l'équipe CASSIS du LORIA - INRIA Lorraine. Il n'aurait pas pu voir le jour sans le soutien de nombreuses personnes que je tiens à remercier.

Je tiens tout d'abord à remercier mes deux directeurs de thèse Michaël Rusinowitch et Adel Bouhoula. A Michaël Rusinowitch qui m'a permis d'intégrer l'équipe CASSIS. Mais surtout à tous les deux, qui grâce à leur disponibilité et rigoureux conseils, j'ai pu entamer, développer et mener à terme ce travail. Qu'ils trouvent ici l'expression de toute ma gratitude.

Je remercie sincèrement tous ceux qui ont bien voulu prendre part à ce jury:

- *A Frédérique Cuppens qui m'a fait l'honneur de présider le jury. Je le remercie pour ses commentaires et ses nombreuses questions.*

- *A Marc Dacier et Hervé Debar qui ont accepté d'être les rapporteurs de ma thèse. Je les remercie pour le temps consacré à ce travail ainsi qu'à leurs remarques et suggestions qui ont contribuées à améliorer le rapport.*

- *A Claude Godart qui a accepté d'examiner cette thèse. Je le remercie pour tout l'intérêt qu'il a manifesté pour ce travail.*

Je remercie également les membres de l'équipe CASSIS. A Sophie pour son assistance et sa patience, et à mes collègues de bureau présents et passés avec qui j'ai passé de si bons moments: Mehdi, Judson, Julien, Silvio et Yanick.

Je tiens à remercier aussi Gregory Kucherov qui m'a fourni les sources de son outil Grappe, à Gabrielle Feltin qui m'a aidé à effectuer des tests sur le réseau du laboratoire et à Mehdi Bouallagui par son soutien et ses discussions fructueuses lors de mes premiers pas avec le logiciel Snort.

Je remercie également tous mes amis Tunisiens et Français pour leur soutien moral et les moments agréables que nous avons passés ensemble.

Enfin, c'est avec beaucoup d'émotion que je remercie maman, papa et tous mes proches.

À ma mère, mon père

Table des matières

3
Division du trafic appliquée à la détection d'intrusion

4
Optimisation des règles de détection d'attaques

Table des figures

Liste des tableaux

Introduction générale

1 Introduction

De nos jours les entreprises s'appuient de plus en plus sur Internet pour fournir leurs services et coopérer avec d'autres compagnies. Les réseaux sont devenus alors des ressources vitales et déterminantes pour le bon fonctionnement des entreprises. Mais l'ouverture facile au monde extérieur via des réseaux connectés à Internet rend l'entreprise plus vulnérable aux attaques. Ces attaques peuvent avoir de graves conséquences comme en témoignent ces dernières années. Par exemple en mars 1997 un pirate suédois a désactivé le système 911 en Floride [143]. Onze états ont été touchés. Le pirate s'est amusé à connecter les opérateurs 911 les uns aux autres. Un autre événement marquant est l'attaque par déni de services répartie (DDosS) qui a paralysé en février 2000 plusieurs sites web populaires dont CNN, Amazon.com, Yahoo ! et eBay [67]. Un jeune pirate canadien de 16 ans en était à l'origine et a causé des dommages évalués à 1,6 Milliards de Dollars.

Par ailleurs, les possibilités d'accéder aux systèmes informatiques s'accroissent avec l'impulsion de nouvelles technologies telles que les réseaux privés virtuels (VPN) et les réseaux Ad-hoc. Ces techniques offrent à des utilisateurs détachés physiquement l'opportunité de se connecter au réseau de l'entreprise. Cependant les attaquants peuvent en profiter pour accéder au réseau local et perturber les communications internes. Ils disposent de plusieurs outils à emploie facile pour réussir leurs exploits. D'après une étude faite à l'Université de Carnegie Mellon [148], les programmes d'attaques deviennent de plus en plus dangereux et nécessitent moins d'expertise ce qui expose les entreprises à des menaces d'intrusions supplémentaires. La Figure 1 montre cette évolution rapide qui aux années 80 concerne des outils développés par des attaquants programmeurs alors que de nos jours repose sur des outils automatiques à la porté des utilisateurs ordinaires. Ceci encourage les gamins scripteurs (Script-kiddies) à employer aveuglement les outils d'attaques sans pour autant avoir la maîtrise des programmes et la conscience des dégâts causés sur le réseau cible.

En outre, les attaquants profitent des logiciels point-à-point en pleine expansion tels que Kazaa, Napster et eMule. Ces utilitaires ouvrent des chemins directs pour accéder à des machines distantes appartenant au réseau cible. De plus elles contribuent à la propagation rapide des virus et des vers sur les réseaux d'entreprises. Enfin elles constituent des moyens faciles d'échange de connaissances et d'outils d'attaques entre les intrus.

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

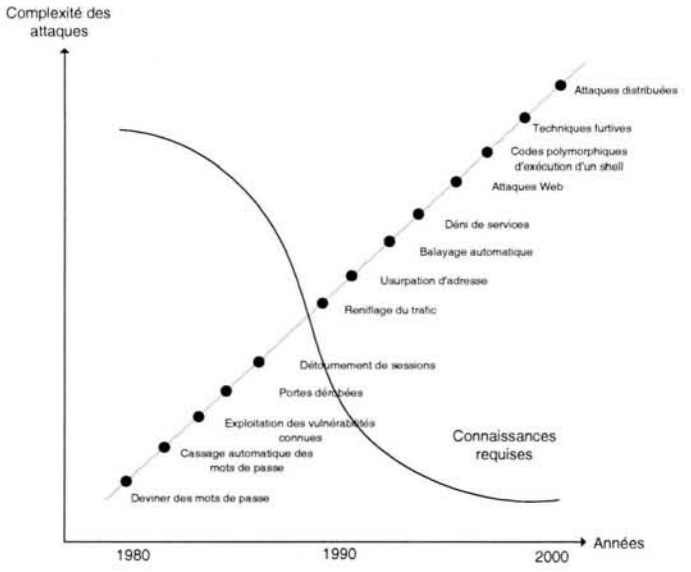


FIG. 1 – Evolution des techniques d'attaques [148]

les scénarios d'attaques et les exploitations non conformes du système informatique. Cette surveillance contribue à éviter le renouvellement des attaques en bloquant les sources d'attaque et en corrigeant les vulnérabilités du système. De plus une détection précoce d'un scénario d'attaque permet de stopper rapidement son développement et par suite éviter des dégâts plus graves.

Suite à leur grand intérêt, les systèmes de détection d'intrusions (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-feux et détecter les attaques passées inaperçues. Considérés comme une dernière barrière de sécurité, les IDS sont capables de comprendre la nature et les caractéristiques du trafic réseau afin de mieux détecter les intrusions. Les administrateurs adaptent les IDS aux services déployés sur leurs réseaux. Ils les configurent pour détecter par exemple les attaques sur les services Web et découvrir la présence des portes dérobées et les balayages de ports. Par conséquent, ils réduisent les risques d'intrusions et accentuent la sécurité du réseau. Cette protection exige néanmoins la bonne configuration de l'IDS et la définition et le respect d'une politique de sécurité réseau qui inclut entre autres la gestion des incidents informatiques.

2 Problématique

L'évolution des dispositifs de sécurité et en particulier des systèmes de détection d'intrusions (IDS) forcent les attaquants à compliquer leurs techniques d'attaques. Ils modifient alors leurs procédés pour compromettre un système informatique en fonction des lignes de sécurité établies sur le réseau. Ils utilisent de plus en plus des techniques d'évasion capables de tromper les IDS. Par exemple si les IDS cherchent des signatures d'attaques dans le contenu des paquets IP, les attaquants distribuent ces motifs sur plusieurs segments du trafic. Ils peuvent désordonner l'envoi des segments, ajouter des données invalides et varier la durée de vie des paquets. De telles techniques forcent les IDS à réviser leurs méthodes de détection afin de filtrer efficacement le trafic réseau.

Outre les attaques d'évasion, les systèmes de détection d'intrusions sont gênés par les conséquences d'une évolution rapide de l'infrastructure réseau. Ainsi pour protéger les nouveaux réseaux, les IDS d'aujourd'hui doivent supporter le haut débit et ceci en menant une analyse rapide du trafic. Un court délai d'inter-arrivée entre les paquets force les IDS à analyser rapidement les données sous peine de rejeter des paquets sans les inspecter. De leur coté, les attaquants essaient de surcharger les IDS en effectuant des attaques par déni de services. Ils envoient un gigantesque trafic pour alourdir le processus de détection, perturber le fonctionnement interne des IDS et les désactiver. La tendance des IDS à analyser passivement le trafic sans contrôler la connectivité réseau encourage les pirates à réaliser ce type d'attaques. En effet, en arrêtant le système de surveillance, les attaquants disposent toujours de la connexion réseau nécessaire pour lancer les attaques avec un risque minime de détection. Par conséquent les systèmes de détection d'intrusions doivent d'une part accélérer leur analyse pour supporter le haut débit et d'autre part se protéger convenablement des attaques afin d'assurer une surveillance continue des systèmes informatiques.

Par ailleurs, l'inspection rapide du contenu des paquets se complique d'avantage à cause du nombre croissant des attaques découvertes chaque jour. Une attaque se présente sous forme d'une règle de détection d'attaque et engendre un ensemble de tests à effectuer sur les paquets reçus. L'évolution du nombre des attaques implique plusieurs tests à réaliser sur chaque paquet ce qui alourdit le processus de supervision du trafic. Le Tableau 1 montre l'évolution rapide des règles de détection d'attaques sur le système de détection d'intrusions *Snort*. L'augmentation du nombre de règles pousse les IDS à optimiser la recherche des attaques en sélectionnant intelligemment les règles candidates de détection qui peuvent s'appliquer sur le trafic en cours.

Version Snort	1.6	1.7	1.8.7	1.9	2.0	2.1	2.2
Réalisation (mois/année)	03/00	01/01	07/02	10/02	04/03	12/03	08/04
Nombre de règles	220	701	1634	1747	1875	2166	2508

TAB. 1 – Evolution du nombre de règles de détection d'attaques de Snort

Enfin les IDS génèrent un grand nombre de faux positifs. Par conséquent les administrateurs se trouvent submergés par d'énormes fichiers journaux de sécurité. Le problème est difficile à gérer d'autant plus qu'une politique trop restrictive de déclenchement des alertes peut manquer de véritables attaques ce qui mène au problème inverse de faux négatif. Par suite, une analyse améliorée du trafic est indispensable pour augmenter les possibilités de détection et réduire le nombre de fausses alertes. De plus une gestion efficace des fichiers de sécurité contribue à discerner rapidement les menaces réelles des fausses alarmes.

Afin de remédier aux différents problèmes évoqués, la détection d'intrusions doit s'orienter vers de nouvelles stratégies pour mieux assurer la sécurité des réseaux. **Nous proposons dans cette thèse une division du trafic qui précède le processus de détection d'intrusions. Cette division tient compte des propriétés du trafic à analyser et des capacités des IDS afin d'associer à chaque classe du trafic la meilleure méthode de détection. A l'issue de cette étape de prétraitement, nous étudions deux mécanismes de détection d'intrusions : le filtrage brut de motifs et l'analyse protocolaire au niveau applicatif. Nous proposons de nouveaux mécanismes qui accélèrent la détection d'intrusions et résistent mieux aux attaques d'évasions.**

3 Contribution

Nous étudions les limites actuelles des systèmes de détection d'intrusions basés réseau et nous proposons des approches complémentaires qui améliorent le mécanisme de détection. Nos principales contributions se résument ainsi :

1. Division du trafic réseau : Nous divisons le trafic réseau afin d'une part supporter le haut débit et d'autre part adapter l'analyse du trafic aux caractéristiques des IDS. D'abord, nous définissons des règles de division de trafic qui décrivent ces propriétés. Ensuite, nous assurons la division du trafic en deux étapes. La première phase est une recherche géométrique des intervalles de ports définis par les règles de division. La deuxième phase est un parcours d'un graphe acyclique orienté construit lors du déploiement du diviseur du trafic à partir des adresses IP définis par ces règles. A l'issue de cette division, nous définissons divers types d'actions comme le rejet du trafic ou sa redirection vers un pot de miel ou un système de détection d'intrusions spécifique.
2. Organisation des règles de détection d'attaques : La détection d'intrusions par scénarios représente les signatures d'attaques sous forme de règles de détection. Cependant, le nombre de ces règles ne cesse de s'accroître en fonction des différentes attaques décelées chaque jour. Une application séquentielle des règles constitue une stratégie inefficace surtout dans le contexte du haut débit. Nous proposons deux méthodes d'application de règles. La première méthode construit un graphe acyclique orienté afin de regrouper les règles de détection d'attaques dans les nœuds finaux du graphe. Nous employons une structure de données similaire à celle utilisée pour diviser le trafic. Cependant, afin de réduire la consommation mémoire nous proposons une deuxième méthode qui ordonne les règles de détection d'attaques dans une liste. La nouvelle organisation optimise l'application des règles en évitant les tests similaires et ceux voués à l'échec.
3. Filtrage à la volée des signatures d'attaques : Afin de résister aux techniques d'évasions, les systèmes de détection d'intrusions s'appuient sur un processus de prétraitement qui regroupe et réordonne les paquets IP avant de déclencher les opérations de filtrage. Nous nous basons sur des structures de données adéquates afin d'assurer un filtrage à la volée du trafic. La méthode tient compte de plusieurs signatures d'attaques et une arrivée désordonnée des paquets. Elle évite ainsi le rassemblement du trafic ce qui économise les ressources mémoires de l'IDS et accélère la détection des attaques. Le travail a été publié en [1].
4. Analyse protocolaire avec des arbres de décisions : la division du trafic permet d'adapter l'analyse des paquets aux propriétés du trafic et à la politique de detection de l'entreprise. Une telle politique exige parfois une supervision détaillée des services fournis par l'entreprise qui ne se restreint pas à un simple filtrage des motifs d'attaques. Pour prendre en compte cette politique, nous proposons une analyse protocolaire via des arbres de décisions. La construction de l'arbre se base sur un système d'inférence combiné avec des méthodes de la théorie d'information pour adapter le processus de détection au trafic habituel transitant sur le réseau supervisé. Le travail fait l'objet d'une publication en [2].

Toutes nos propositions sont implantées et donnent lieu à des expérimentations que nous décrivons au cours de cette thèse. Nous divisons le trafic sur une topologie de réseau virtuel (VLAN). Ensuite nous vérifions le contenu des paquets reçus avec un des plusieurs instances de Snort 1.9 déployées sur les diverses interfaces du réseau virtuel. Par ailleurs, l'organisation des règles et le filtrage à la volé des signatures d'attaques sont testés sur Snort 1.9. Enfin l'analyse protocolaire est développée à part, et est introduite ensuite comme un pré-processeur de Snort 1.9.

4 Organisation de la thèse

Nous présentons dans le Chapitre 1 un scénario possible pour mener une attaque sur le réseau informatique. Cette analyse permet de comprendre les signatures d'attaques et par suite déterminer les différents tests réalisés par les systèmes de détection d'intrusions pour inspecter le contenu des paquets. Nous évoquons également dans la deuxième partie du Chapitre 1 les solutions de sécurité actuelles telles que les pare feux et les pots de miels. Nous soulignons pour chacune ces limites afin de déterminer l'intérêt des systèmes de détection d'intrusions.

Nous exposons dans le Chapitre 2 un état de l'art sur la détection d'intrusions. Nous nous focalisons sur le filtrage de motifs d'attaques qui représente notre axe de recherche. Nous soulignons également les limites actuelles des systèmes de détection d'intrusions réseaux avant de présenter notre proposition d'une architecture pour un système de détection d'intrusions. Il s'agit d'une solution générale qui se base sur un diviseur de trafic et de plusieurs modules de détection qui seront explicités dans les prochains chapitres de la thèse.

Le Chapitre 3 formalise le problème de division du trafic puis présente notre méthode de classification. Nous nous intéressons d'abord à la construction d'une matrice et de graphes de classification. Ensuite nous présentons l'algorithme du parcours du graphe et les résultats expérimentaux obtenus.

Le graphe de classification du trafic peut être adapté pour regrouper des règles de détection d'attaques stockées sur un système de détection d'intrusions par scénarios. Nous expliquons dans le Chapitre 4 ce procédé qui permet de sélectionner rapidement des règles candidates de détection d'attaques pouvant s'appliquer sur le paquet en cours d'analyse. Nous comparons cette méthode avec une nouvelle technique que nous proposons et qui organise les règles de détection d'attaques afin d'accélérer la recherche des règles candidates.

Le Chapitre 5 traite le filtrage de motifs, une opération vitale pour la plupart des systèmes de détection d'intrusions. Nous présentons une nouvelle approche pour le filtrage à la volée des motifs d'attaques. Cette méthode qui est implantée dans le système de détection d'intrusions Snort, évite le rassemblement du trafic ce qui permet d'accélérer le processus de détection d'attaques et protège mieux l'IDS en sauvegardant ses ressources mémoires.

Ensuite nous consacrons le Chapitre 6 à étudier l'impact de l'analyse protocolaire sur la détection d'intrusions. Ayant souligné son intérêt, nous proposons un algorithme pour construire des arbres de décisions qui implémentent des règles de détection d'attaques. Chaque arbre représente un seul protocole applicatif et considère la distribution des tests par rapport à deux classes *attaque* et *non attaque* pour adapter le processus de détection au trafic transitant sur le réseau à surveiller.

Nous concluons notre travail au Chapitre 7 puis nous présentons nos différentes perspectives. Par ailleurs, nous étudions dans l'Appendice 1 une approche décentralisée pour détecter les balayages de ports. La solution repose sur divers agents qui coopèrent afin de détecter les signatures d'attaques. La méthode s'applique suite à une division du trafic ou dans l'environnement des réseaux commutés.

1

Menaces et techniques de protection contre les attaques réseaux

Sommaire

1.1 Introduction

La sécurité informatique constitue la réplique des administrateurs pour contrarier les attaques couramment menées sur les réseaux informatiques. Les systèmes de sécurité essaient de prévenir de telles attaques et de corriger les vulnérabilités exploitées lors des dernières intrusions. Il est alors nécessaire pour sécuriser le réseau d'entreprise d'identifier les menaces potentielles et de connaître les procédés des attaquants. La compréhension de l'activité intrusive et des techniques utilisées permet de définir des méthodes efficaces de surveillance et de détection des attaques. Nous consacrons la première partie de ce chapitre à expliquer un scénario possible pour attaquer le réseau de l'entreprise. Ensuite nous présentons les solutions possibles de protection du système informatique. Nous analysons les limites de ces techniques ce qui permet de souligner la complexité de la prévention complète des attaques. D'ou l'intérêt de la détection d'intrusions pour retrouver les mauvaises exploitations du système à la recherche de nouvelles brèches qui mènent à la violation de la politique de sécurité de l'entreprise.

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

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

Les différents types d'attaquants cherchent à découvrir les propriétés du réseau cible avant de lancer les attaques. On parle généralement de la reconnaissance qui peut être passive ou active. Ayant récolté les informations nécessaires, ils lancent leurs vraies attaques pour exploiter le système. Ensuite ils créent des portes dérobées pour garantir des futurs accès faciles au système compromis. Enfin ils effacent leurs traces des journaux de sécurité. Nous détaillons dans la suite ces différentes étapes en les illustrant par des vrais scénarios d'attaques.

1.2.1 Reconnaissance passive

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

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

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

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

1.2.2 Reconnaissance active

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

Système d'exploitation	WS sans TS	WS avec TS
Linux 2.4.0	5840	5792
Windows NT 4.0	64240	65160
MAC OS 10.1	32768	33000
Open BSD 3.3	64240	65160
Free BSD 9.2	16384	17520
Free BSD 4.6	57344	57344

TAB. 1.1 – Options TS et WS en fonction des systèmes d'exploitation

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

1.2.3 Exploitation du système

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

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

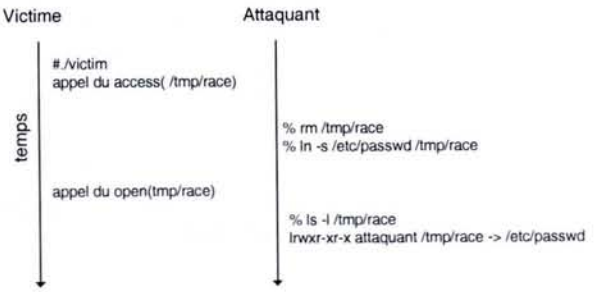


FIG. 1.1 – Accès concurrents au contenu du fichier

Par ailleurs un attaquant peut introduire des données imprévues pour tromper les applications mal conçues. Il réussit ainsi des dépassements de tampon, altère des requêtes SQL et détourne des mécanismes d'authentification. Nous présentons dans la Figure 1.2 le mécanisme de débordement de tampon. Il s'agit de copier dans la pile une grande quantité de données dans un espace mémoire assez petit, prévu pour contenir les variables locales des fonctions. Les données en excès contiennent du code malveillant et pénètrent dans des espaces mémoires voisins. De plus, le pointeur retour de la fonction se voie écrasé par une autre adresse qui pointe directement vers le code malveillant. Par conséquent, l'attaquant exploite le privilège de l'application pour créer de nouveaux comptes utilisateurs, élever son privilège ou copier des données sensibles du système.

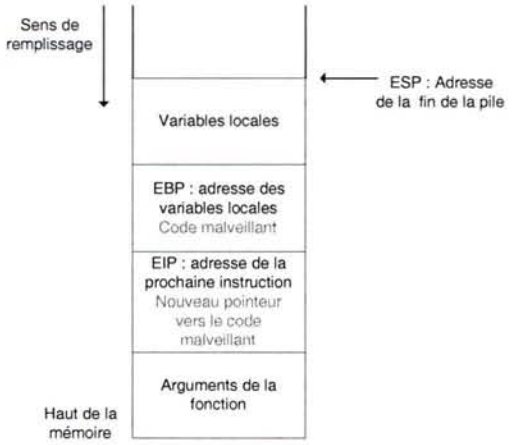


FIG. 1.2 – Débordement de tampon

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

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

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

En outre, un intrus utilise les attaques par déni de services pour exploiter les systèmes informatiques. Il empêche l'exécution normale des services par la saturation de la bande passante ou l'épuisement des ressources système. La Figure 1.3 montre le détournement de session réalisé par Mitnick. Ce dernier a pu, via une inondation SYN, isoler la machine A qui maintient des relations d'approbation avec la machine B. Ensuite il accède à la machine B en détournant la session de A.

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

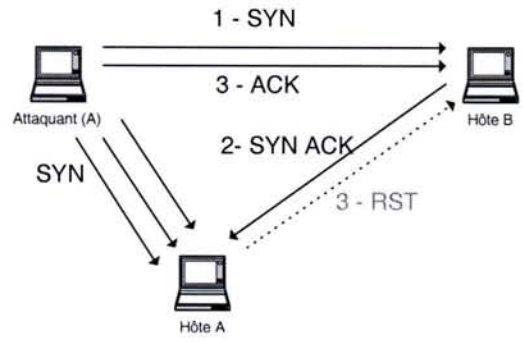


FIG. 1.3 – Attaque de détournement de session

1.2.4 Préservation de l'accès

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

Un attaquant prudent utilise des chevaux de Troie qu'il télécharge aussitôt qu'il gagne son premier accès. Pour ce faire, il dispose de plusieurs sources d'informations comme l'IRC, les pages Web, le partage point à point et les dépôts FTP. La trace présentée dans [156] montre un intrus qui installe le rootkit "lrk4". Il s'agit de plusieurs programmes modifiés dont l'un remplace le binaire "/bin/login". Le nouvel exécutable accepte par défaut le compte root "rewt" avec le mot de passe "saton". Ainsi les futures connexions TELNET utilisant ce compte réussissent.

1.2.5 Effacement des traces

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

1.3 Protection du système d'information

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

1.3.1 Pare feux

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

- pare feu avec filtrage des paquets : ce pare feu filtre les paquets en utilisant des règles statiques qui testent les champs des protocoles jusqu'au niveau transport. Ces fonctionnalités sont généralement incorporées dans un routeur appelé routeur à écran,
- pare feu à filtrage des paquets avec mémoire d'états : ce modèle conserve les informations des services utilisés et des connexions ouvertes dans une table d'états. Il détecte alors les situations anormales suite à des violations des standards protocolaires,
- pare feu proxy : ce pare feu joue le rôle d'une passerelle applicative. En analysant les données jusqu'au niveau applicatif, il est capable de valider les requêtes et les réponses lors de l'exécution des services réseaux.

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

Les scripts constituent aussi des sources d'intrusion que les pare feux échouent à détecter. Par exemple la vulnérabilité du RDS (Remote Data Service) sur les serveurs web IIS (Internet Information Server) de Microsoft permet aux intrus d'exécuter des commandes à distance sur des stations Serveur NT [28]. Le script "msadc.pl" de Rain Forest Puppy (RFP) exploite cette vulnérabilité [1]. Il emploie des méthodes valides du protocole HTTP telles que GET et POST pour pouvoir passer inaperçu à travers un pare feu proxy.

1.3.2 Scanners de vulnérabilités

Les scanners de vulnérabilités automatisent la découverte des failles de sécurité. Ils sont utilisés par les attaquants pour localiser les faiblesses du réseau cible. De plus les administrateurs peuvent en tirer profit pour corriger les vulnérabilités de leurs systèmes informatique. Nous citons à titre d'exemple Nessus [75], Whisker [133] et Saint [29].

Cependant les scanners présentent quelques limites qui peuvent être résumées en 3 points : l'exhaustivité, la mise à jour et l'exactitude. En effet, malgré le grand nombre de vulnérabilités détectées, les scanners d'aujourd'hui sont inaptes à déterminer toutes les faiblesses possibles. De plus, la mise à jour de ces produits ne suit pas le rythme de la découverte des nouvelles vulnérabilités. Enfin, la modification des bannières des services scannés permet de dissuader facilement le scanner ce qui entraîne parfois un responsable de sécurité à chasser des vulnérabilités fantômes.

1.3.3 Outils d'archivage

La plupart des systèmes d'exploitation fournissent des utilitaires d'archivage. Par exemple le daemon syslogd d'Unix enregistre dans des fichiers journaux de sécurité les opérations intéressantes exécutées sur le système. Parmi les fichiers log créés, trois sont susceptibles d'être manipulés par les attaquants à savoir wtmp, utmp et lastlog [114].

[1] http ://www.exploits.rg3.net/

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

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

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

1.3.4 Cryptographie

La cryptographie garantit la confidentialité, l'intégrité, la non répudiation et l'authenticité des données. Elle est fréquemment utilisée dans diverses applications réseaux telles que la messagerie, les connexions à distance, les réseaux privés et les serveurs web. Les administrateurs l'utilisent pour sécuriser leurs systèmes informatiques mais elle ne constitue pas une solution unique et suffisante. Effectivement, diverses implémentations des protocoles de sécurité se sont révélées vulnérables. De plus la sécurité peut être rompue via plusieurs types d'attaques. Par exemple l'homme du milieu (MITM) constitue une menace lors des créations des clés. Par ailleurs les courts et les simples mots de passes utilisés comme des clés de sécurité par les algorithmes symétriques sont facilement cassables via des attaques par dictionnaires ou de recherche exhaustive. Il y a aussi le risque de vol des clés privées suite à un mauvais stockage de ces informations ou la cryptanalyse appliquée sur des algorithmes à faible encodage tels que le XOR et la Base 64.

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

1.3.5 Pots de miels

Un pot de miel est une machine qui présente ou simule des failles de sécurité très répandues [157]. Disposant des moyens renforcés de surveillance, la machine peut servir d'appât pour apprendre la stratégie des attaquants et construire des signatures exactes d'attaques. Par ailleurs la simulation du comportement d'une machine doit être aussi réaliste pour ne pas éveiller les soupçons des attaquants.

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

1.3.6 Systèmes de détection d'intrusions

Une intrusion est toute activité qui menace la politique de sécurité de l'entreprise et mène à sa violation [19]. L'origine de l'intrusion est multiple et peut être due à un espionnage industriel ou des attaques lancées pour des gamins scripteurs. Ainsi un système de détection d'intrusions (IDS) tente d'identifier les menaces dirigées contre le réseau de l'entreprise. Il s'appuie sur plusieurs sources d'informations comme les fichiers d'audit, les journaux de sécurité et le trafic réseau. Nous avons étudié dans les sous sections précédentes diverses solutions pour sécuriser le réseau informatique et mentionné leurs intérêts et leurs limites. Il s'est avéré que ces outils ne peuvent pas prévenir toutes les attaques et par suite assurer seuls une sécurité idéale du réseau. Etant donnée l'impossibilité de stopper toutes les attaques, les systèmes de détection d'intrusions constituent une bonne solution pour détecter celles qui passent inaperçues. Placés après les pare feux, les IDS constituent la dernière barrière de sécurité. Ils analysent le trafic qui passe à travers les pare feux et supervisent les activités des utilisateurs sur le réseau local. Par ailleurs, placés avant les pare feux, les IDS découvrent les attaques à l'entrée du réseau.

Les IDS s'appuient généralement sur deux sources d'information : les paquets transitant sur le réseau et les informations collectées sur les machines. On parle alors de deux types de systèmes de détection d'intrusions : les IDS basés réseau et les IDS basés hôte. Ces deux catégories d'IDS emploient généralement deux principes de détection : l'approche comportementale et l'approche basée sur la connaissance. La détection par la connaissance définit des signatures d'attaques qui décrivent les intrusions. Ces signatures ne sont autres que les empreintes laissées par les intrus au cours de leurs exploits. La deuxième approche de détection, c'est-à-dire comportementale, se réfère au comportement normal et habituel des différents acteurs du système à protéger (application, utilisateur, etc.). Ensuite une déviation importante par rapport au normal représente une activité suspecte et révèle éventuellement une attaque. Nous détaillons les deux approches de détection ainsi que leurs limites dans le Chapitre 2.

1.4 Conclusion

Les attaquants suivent une stratégie d'attaque pour réussir leurs exploits. Ils disposent de plusieurs sources d'information et de divers outils pour compromettre le système informatique. Par conséquent, les administrateurs déploient des solutions de sécurité efficaces capables de protéger le réseau de l'entreprise. Dans ce contexte, les systèmes de détection d'intrusions constituent une bonne alternative pour mieux sécuriser le réseau informatique. Nous détaillons dans le Chapitre 2 les qualités requises des systèmes de détection d'intrusions. Nous discutons aussi les approches proposées dans la littérature et ceci en se basant sur les deux principes de détection à savoir la détection comportementale et la détection par la connaissances.

2

Détection d'intrusions : approches et limites

Sommaire

2.1 Introduction

Nous avons présenté au Chapitre 1 un scénario possible d'attaque sur le réseau de l'entreprise. Ce plan d'attaque souligne la nécessité et la complexité de sécuriser un système informatique. De nos jours, diverses méthodes de prévention existent. Cependant, malgré leur grand intérêt, elles présentent des lacunes dues à la nature des données qu'elles traitent d'une part et aux techniques évoluées pour mener et dissimuler les attaques d'autre part. Ainsi et dans l'impossibilité de stopper toutes les attaques, la détection d'intrusions tente de déceler celles qui passent inaperçues à travers les autres systèmes de sécurité. Par ailleurs déployés à l'entrée du réseau, les systèmes de détection d'intrusions détectent les différentes attaques lancées contre le réseau de l'entreprise. La compréhension des origines des intrusions et des intentions des pirates permet de développer les moyens efficaces de prévention des attaques.

Après la courte introduction des systèmes de détection d'intrusions présentée au Chapitre 1, nous détaillons dans ce chapitre les qualités requises des IDS. Nous présentons un état de l'art des approches proposées dans la littérature et des limites actuelles de détection. Nous introduisons ensuite notre procédé de détection qui se base sur une division du trafic puis une application rapide des règles de détection d'attaques et un filtrage à la volée du trafic. Ces techniques d'analyse du contenu des paquets seront développées dans les prochains chapitres du rapport.

2.2 Les systèmes de détections d'intrusions

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

2.2.1 Classification des systèmes de détection d'intrusions

La détection d'intrusions a commencé en 1980 avec le travail d'Anderson [8]. Il a proposé la surveillance des échecs d'authentification pour détecter les tentatives de pénétration au réseau local. De plus, l'observation des fichiers et des programmes manipulés permet de détecter des attaques internes. Enfin l'usurpation d'une identité est repérable en comparant l'activité courante au comportement habituel d'un utilisateur. Ensuite, Denning [40] a proposé en 1987 le premier modèle de la détection d'intrusions. Son approche statistique construit des profils qui relient des sujets à des objets. Un sujet est l'initiateur des actions. Il peut être un utilisateur ou un processus qui manipule des objets c'est-à-dire des programmes ou des fichiers. Suite à ce premier modèle, plusieurs approches de détection d'intrusions ont été proposées. Elles reposent sur diverses techniques de détection. Etant donnée la diversité de ces méthodes, des schémas de classification ont été proposés par Lunt [105], Sundavar [168], Debar [36, 37], Axelsson [10] et Bace [12]. Debar [37] utilise cinq critères pour classer les systèmes de détection d'intrusions (Figure 2.1) :

- Méthode de détection : deux principes de détections existent. L'approche comportementale modélise le comportement normal des utilisateurs, du système informatique et de l'activité réseau. Ensuite, toute déviation par rapport à la normale constitue un 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.
- Source d'information : les données analysées partagent les systèmes de détection d'intrusions en deux catégories : les systèmes de détection d'intrusions réseaux et les systèmes de détection d'intrusions hôtes. La première catégorie des IDS filtre le trafic réseau et se déploie généralement dans des endroits précis du réseau, par exemple dans la zone démilitarisée, un brin réseau contenant des serveurs internes ou juste avant ou/et après un pare feu. La deuxième catégorie des IDS analyse les données des journaux de sécurité établis par les systèmes d'exploitation et les applications qui tournent sur les machines. Ces IDS sont déployés directement sur les hôtes du réseau.
- Réponses des IDS : les systèmes de détection d'intrusions émettent des réponses actives qui influent directement la source d'attaque, comme ils peuvent se restreindre à des réponses passives qui inscrivent l'événement suspect. Une liste de réponses actives et passives est présentée dans le Tableau 2.1 [20]. Par ailleurs et afin de sélectionner la meilleure réponse, Cuppens introduit la notion d'anti-corrélation et l'applique sur le but de l'intrusion [33].
- Paradigme de détection : la détection d'intrusions s'effectue en analysant l'état courant du système ou en supervisant les transitions des états normaux aux états dangereux. Durant

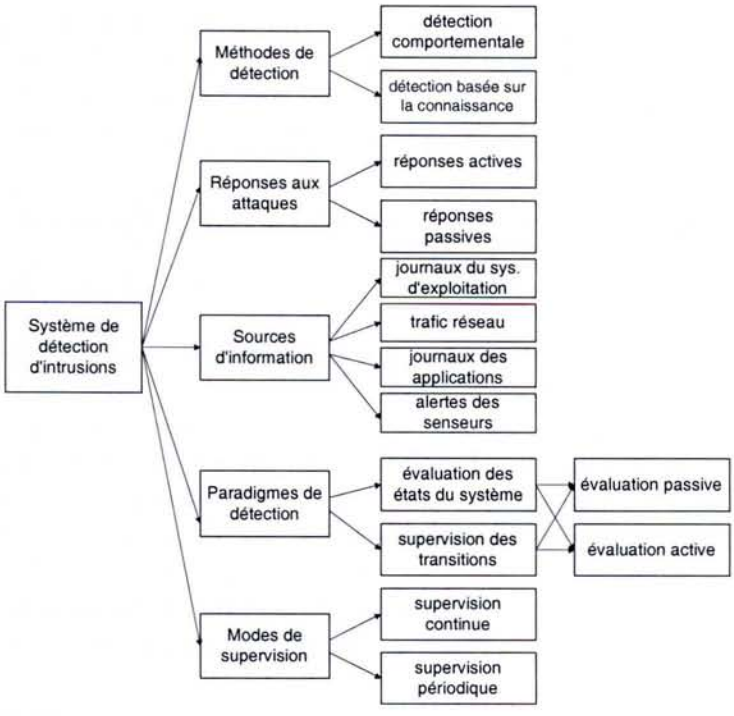


FIG. 2.1 – Classification des systèmes de détection d'intrusions [37]

ces deux types d'inspection, l'IDS récupère les informations en interrogeant directement le système ou en écoutant passivement les événements.

- Mode de supervision : l'analyse assurée par un système de détection d'intrusions peut être continue ou périodique dans le temps.

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

TAB. 2.1 – Réponses aux attaques des systèmes de détection d'intrusions

Les systèmes de détection d'intrusions utilisent diverses techniques de détection et définissent plusieurs modules qui effectuent des tâches distinctes comme la collecte des informations, l'analyse des données, la corrélation des événements et la génération des alarmes. Afin d'homogénéiser les communications entre les différents équipements, des travaux de standardisation ont été développés et d'autres sont en cours de développement. On note en particulier le CIDF [166] qui développait des protocoles et des API permettant la réutilisation facile des composants des IDS.

Par ailleurs l'IETF a créé un groupe de travail IDWG (Intrusion Detection Working Group) [47] pour définir le format des données et des échanges entre les systèmes de détection et de réponse aux intrusions. Le but du travail est de partager facilement l'information entre les systèmes de détection d'intrusions et de contrôler tout système interagissant avec eux [34, 49, 187].

2.2.2 Qualités requises des systèmes de détection d'intrusions

Les systèmes de détection d'intrusions actuels tendent à garantir les cinq propriétés suivantes [37] :

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

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

2.3 Méthodes de détection d'intrusions

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

2.3.1 Détection d'intrusions comportementale

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

2.3.1.1 Approche statistique

L'analyse statistique du comportement normal du système est l'une des premières approches adoptées en détection d'intrusions. Denning [40] présente un modèle dans lequel un profil relie via une variable aléatoire un sujet (utilisateur, processus) à un objet (ressources). Si après la création du profil, la valeur de la variable aléatoire dépasse le seuil toléré alors le comportement est considéré anormal. Divers systèmes de détection d'intrusions utilisent ce concept. NIDES [74] calcule des valeurs d'anomalie de plusieurs activités (temps CPU, bande passante, nombre et nature des services sollicités, etc). Il effectue ensuite la pondération des carrés de ces valeurs afin

de calculer un score d'anomalie global S (Eq. 2.1). Le score S est toujours positif et s'il dépasse le seuil toléré M, alors il s'agit d'un événement suspect.

$$S = a_1 S_1^2 + a_2 S_2^2 + ... + a_n S_n^2 \tag{2.1}$$

Haystack [151] utilise également l'approche statistique. Il effectue un ensemble de mesures $X = (x_1, x_2, ..., x_n)$ puis représente sous forme d'un vecteur de Bernoulli v toute mesure x_i qui dépasse un seuil prédéfini $< t_{i,min}, t_{i,max} >$. Ensuite le vecteur v est multiplié par un vecteur poids p spécifique à chaque intrusion. Le produit obtenu représente le score d'anomalie et il est considéré élevé si la probabilité d'obtenir des scores plus petits est grande.

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

2.3.1.2 Apprentissage automatique

Une autre technique de la détection d'intrusions comportementale consiste à prévoir la séquence normale des événements audités. Teng [172] propose une méthode inductive de génération de règles décrivant la séquence normale des activités des utilisateurs. L'équation 2.2 représente un exemple de règles :

$$E_1,\ E_2,\ *\ \rightarrow\ (E_4 = 95\ \%,\ E_5 = 5\%) \tag{2.2}$$

La règle 2.2 utilise le caractère * pour remplacer un événement quelconque et indique que la séquence (E_1, E_2, *) est suivie par l'événement E_4 dans 95% des cas et par E_5 dans 5% des situations possibles. Une déviation par rapport au comportement normal se caractérise par une apparition d'événements autres que E_4 et E_5 ou une disproportion des événements E_4 et E_5. La technique est capable de modéliser une variété de comportements et résiste bien à l'apprentissage progressif et malveillant des intrus. Néanmoins, elle est trop liée à la séquence d'événements ce qui pose des problèmes dans le cas où un utilisateur effectue simultanément deux ou plusieurs tâches.

Les deux méthodes décrites précédemment souffrent des données bruit lors de la phase d'apprentissage. Pour remédier à ce problème, Fox [56] propose l'utilisation des réseaux de neurones pour prévoir la prochaine commande dans une fenêtre de taille w. Les réseaux de neurones sont ensuite utilisés dans divers travaux [35, 141, 171]. Ghost et ses co-auteurs [59] emploient le modèle d'Elman des réseaux de neurones pour tenir compte des séquences récurrentes dans les exécutions des programmes. Ces approches sont capables de corréler automatiquement les différentes mesures. Néanmoins la durée nécessaire d'apprentissage constitue souvent une paramètre cruciale à définir.

Récemment, quelques travaux proposent l'utilisation des cartes auto-organisatrices de Kohonen (SOM) pour assurer l'apprentissage automatique. Lichodzijewsk [127, 102] construit des profils de connexions réseaux et utilise les multi SOM pour assurer une détection d'intrusions basée hôte. Par ailleurs Ramadas [135] applique le même concept mais en distinguant les services réseaux. Les réseaux de Kohonen évitent l'apprentissage supervisé et présentent une forte sensibilité aux données fréquentes. Ils réduisent également la complexité calculatoire. En revanche ils peuvent présenter un taux d'erreur élevé et ne garantissent pas la convergence sur des réseaux de grande dimension.

2.3.1.3 Approche immunologique

Forrest [52] était la première à utiliser l'approche immunologique pour modéliser les processus sur une machine. Sa méthode consiste à décrire le comportement normal ou le "soi" via une séquence finie d'appels systèmes. Les séquences appelées N-gram servent de base pour comparer les appels systèmes des processus lors d'une phase de surveillance. Cette comparaison énumère les différences entre les paires dans une fenêtre de taille k (tide) [52] ou utilise des règles de r bits contiguës (stide) [68]. Wespi, Dacier et Debar [184] considèrent un cas plus général en analysant les événements d'audit. Ils génèrent des séquences d'événements de tailles variables pour modéliser l'état normal du système. Ensuite un motif est sélectionné s'il existe d motifs qui le suivent directement sinon le score d'anomalie est incrémenté de 1 et une alarme est déclenchée lorsque le score dépasse le seuil toléré.

Marceau [112] optimise la représentation des N-gram sous forme de graphes acycliques orientés (DAG) ce qui permet de réduire la base de profils définie par Forrest. De plus, elle utilise le mécanisme de fenêtre glissante pour comparer les motifs. Kosoresow [79] étudie les caractéristiques des traces des appels systèmes et remarque que les différences entre les motifs apparaissent dans des régions de tailles fixes. En divisant la trace en 3 parties : début, corps et fin, il réussit à générer de nouvelles séquences de motifs représentées en des machines à états finis. La méthode permet de réduire le nombre de séquences. Par exemple, 26 descriptions du processus sendmail suffisent au lieu de 147. Cependant l'auteur propose une construction manuelle de l'automate pour traduire ces motifs.

Warrender et Forest comparent dans [183] quatre approches immunologiques : la séquence simple d'événements (stide), la séquence d'événements menue des fréquences d'apparition (t-stide), la génération automatique des règles inductives via RIPPER et le modèle caché de Markov (HMM). Ils concluent qu'en moyenne la modélisation HMM présente des meilleurs performances. Mais il ne s'agit pas d'une supériorité absolue puisque les résultats des expériences dépendent des programmes testés.

La lenteur d'apprentissage constitue le principal inconvénient de la modélisation HMM. Afin de résoudre ce problème, Cho [24] s'intéresse uniquement aux événements qui sont en relation avec le changement de privilège. Il réduit ainsi le nombre d'appels systèmes audités de l'ordre de 75% (de 267 à 80). Par ailleurs et afin de diminuer les faux positifs très répandus dans le cadre d'une détection d'intrusions comportementale, l'auteur combine plusieurs HMM et utilise diverses mesures dont les informations utiles sont raffinées via une méthode d'auto-organisation de données (SOM).

2.3.1.4 Spécification des programmes

La spécification des exécutions normales des processus constitue une autre technique de la détection d'intrusions comportementale. Ko, Fink et Levittet [78] s'intéressent aux programmes avec privilège "super utilisateur" et développent un langage de spécification qui se base sur la logique des prédicats et les expressions régulières. Cependant la réalisation de ces spécifications est une tâche assez difficile. Les auteurs proposent une méthode utilisant une logique inductive pour synthétiser directement les spécifications à partir des traces valides [77].

Dans ce contexte, Nuansri [126] construit un diagramme de transition d'états pour représenter les changements de privilège. Il définit ainsi un ensemble de règles dont la violation révèle la présence d'une attaque.

Enfin Sekar et Uppurili [177] utilisent un langage spécifique BMSL (Behavioral Monitoring Specification Language) pour modéliser les propriétés importantes des événements à analyser.

Ils expriment via ce langage des règles de la forme $motif \rightarrow action$. Les motifs sont des expressions régulières d'événements qui seront traduites sous forme d'une machine à états finis quasi déterministe. Les actions sont des opérations d'affectation à des variables d'états ou de simples appels à des routines extérieures pour stopper les attaques en cours.

2.3.1.5 Fouilles de données et théorie d'information

Lee et Xiang [101] utilisent la théorie d'information pour comprendre la nature des données auditées et par suite construire des modèles de détection d'intrusions comportementale. Les techniques de fouilles de données (Data Mining) permettent également de construire des modèles de détection adaptatifs. Les algorithmes utilisées par Lee [100, 97] divisent les données en deux catégories : des données normales et des données anormales. Cette classification permet de construire des règles qui expriment des relations entre les enregistrements des fichiers de sécurité. Par exemple, pour un utilisateur particulier, l'éditeur Xemacs est le plus souvent associé à des fichiers ".c". Lee souligne que l'extraction des événements fréquents permet de mieux analyser les traces d'événements. De plus une méta-classification des analyses de plusieurs IDS garantie une meilleure détection avec moins de faux positifs. Ces différentes techniques sont implantées dans le système de détection d'intrusions JAM [167]. De plus l'analyse de données porte sur des traces normales pour assurer une détection comportementale ou bien sur des traces d'intrusions. Elle contribue donc à construire des règles de détection d'attaques utilisables lors d'une détection d'intrusions par connaissances.

ADAM est autre système de détection d'intrusions qui utilise la fouille de données. Il est basé sur les travaux de Barbara [14, 13] et effectue deux étapes d'apprentissage. La première étape utilise des données hors ligne pour construire des règles d'association modélisant les profils normaux. La deuxième étape considère des données en ligne et emploie les règles d'association déjà construites pour créer un classificateur d'événements suspects. L'objectif de cette phase est de rendre le système de détection d'intrusions plus apte à distinguer les vraies attaques des faux positifs.

2.3.2 Détection d'intrusions basée sur la connaissance

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

2.3.2.1 Systèmes experts

Les premiers prototypes des systèmes de détection d'intrusions utilisent les systèmes experts pour détecter les scénarios d'attaques. La base de règles traduit les connaissances des experts ce qui permet de transformer les faits extraits des sources d'audit et d'en déduire la présence des attaques. Plusieurs langages de définition de règles sont proposés. Par exemple, P-BEST (Production Based Expert System Toolset) a été initialement développé pour le système MIDAS [144] puis employé par IDES [106, 107], NIDES [74] et EMERALD [121]. Par ailleurs, Le Charlier définit le langage RUSSEL (Rule based Sequence Evaluation Language) pour décrire les signatures d'attaques dans le système ASAX [65]. Néanmoins l'utilisation de ces deux langages nécessite une certaine expertise. De plus, définir de nouvelles signatures d'attaques nécessite du temps surtout lorsque la description d'une attaque requiert l'ajout de plusieurs règles.

Afin d'améliorer la détection des attaques en tenant compte des activités courantes sur le système, Lunt et Garvey [58] ajoutent au système expert le concept du raisonnement sur les modèles. En effet, à chaque modèle d'attaque est associé des indicateurs d'activités. Ainsi, la supervision des activités permet d'activer seulement quelques modèles et par suite de tenir compte uniquement des scénarios d'attaques susceptibles d'être vérifiés.

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

2.3.2.2 Automate et logique temporelle

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

2.3.2.2.1 Machines à états finis

Porras [131] représente les signatures d'attaques sous forme de machines à états finis. Les états de la machine correspondent aux états du système en voie de compromission et ils contiennent des assertions qui doivent être vérifiées pour pouvoir transiter d'un état à un autre. Les arcs sont étiquetés par des événements qui forment le scénario d'attaque. Le système de détection d'intrusions USTAT [71] implante cette approche pour détecter les attaques sur le système Unix. Par ailleurs Kermmer et Vigna [180, 181] adaptent le système pour détecter des attaques réseaux. Leur modélisation transforme l'infrastructure réseau en un hypergraphe dont les nœuds sont des interfaces réseaux et les arcs sont des hôtes et des liens réseau.

Nous constatons que la description des scénarios d'attaques sous forme de machines à états finis est un moyen plus facile que la définition de règles dans un système expert. Cependant, la méthode ne permet d'exprimer que de simples relations qui portent sur des séquences d'événements.

2.3.2.2.2 Réseau de Pétri

Kummar [86, 87] utilise les réseaux de pétri pour représenter les scénarios d'attaques . Les transitions sont étiquetées par des appels systèmes alors que les jetons évoluent chaque fois qu'un événement permet de tirer une transition. Un jeton possède une couleur qui est une valuation des variables. De plus, les transitions sont gardées pour vérifier certaines conditions. Les réseaux de pétri offrent un bon pouvoir expressif puisqu'ils permettent de représenter l'existence et la séquence d'événements. De plus ils expriment des expressions régulières menées d'un opérateur de parallélisme. Néanmoins leur mise en ouvre pose quelques problèmes. En effet, à chaque commande on crée de nouveaux jetons, ce qui augmente le nombre de jetons présents et alourdit le processus de vérification.

2.3.2.2.3 Logique temporelle

Roger [138] transforme l'analyse des journaux d'audit en un problème de vérification des propriétés dans une logique temporelle. En effet, les scénarios d'attaques sont des formules de la

logique temporelle linéaire alors que le fichier d'audit est la trace sur laquelle les algorithmes de vérification s'exécutent. Le langage de définition des attaques devient assez expressif grace aux opérateurs de la logique temporelle (Next, Until). De plus la vérification traite des problèmes d'atteignabilité, de sûreté et d'équité.

2.3.2.3 Algorithmes génétiques

Le système de détection d'intrusions Gassata [110] utilise les algorithmes génétiques pour détecter les intrusions à posteriori . Cette méthode de détection vise particulièrement les attaques dont l'ordre temporel des événements importe peu dans le déroulement du scénario d'attaque. Mé définit une matrice d'attaques/événements notée $\mathcal{A}$ qui caractérise les événements associés à chaque attaque. De plus il construit un vecteur $\mathcal{R}$ du risque encouru par chaque attaque et de la liste d'événements $\mathcal{O}$ observés durant une période T. Le problème de détection se transforme en un problème d'optimisation qui est résolu via les algorithmes génétiques (Eq 2.3).

$$\begin{cases} \mathcal{A} * \mathcal{H} \leq \mathcal{O} \\ \mathcal{R} * \mathcal{H} = Maximum \end{cases} \tag{2.3}$$

La méthode de détection s'adapte bien lorsque les administrateurs s'intéressent uniquement à des attaques considérées intrusives et sévères sur leurs réseaux. De plus le temps d'analyse est satisfaisant. Cependant, la détection est fortement liée à la période T de collecte des événements. De plus, elle ne considère pas les événements communs entre plusieurs attaques. Enfin, la construction de la matrice attaques/événements exige une certaine expertise de la part de l'utilisateur.

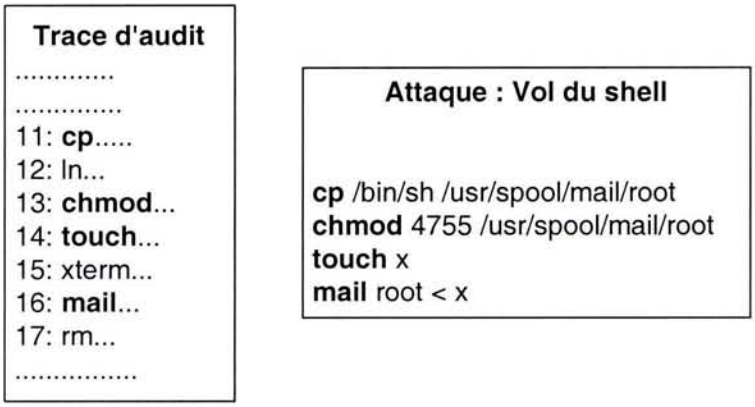
2.3.2.4 Fouille de données

Les techniques de la fouille de données (Data Mining) s'appliquent aussi bien pour une détection d'intrusions par anomalie que pour une détection d'intrusions basée sur la connaissance. Nous avons présenté cette méthode dans le cadre de la détection d'intrusion comportementale. Lee [98] utilise les données d'apprentissage du programme d'évaluation DARPA pour construire automatiquement des règles de détection d'attaques. Ces règles servent ensuite à la détection en ligne des scénarios d'attaques.

2.3.2.5 Filtrage de motifs

La description sémantique d'un scénario d'attaque peut se traduire en informations explicites caractérisant l'attaque. Pour une détection basée hôte, ces informations se présentent sous forme d'événements à retrouver à partir des journaux de sécurité. Cependant, un système de détection d'intrusions basé réseau examine les champs des protocoles réseaux et filtre le contenu des paquets à la recherche des chaînes de caractères qui sont des motifs d'attaques. Deux exemples de signatures d'attaques pour une détection d'intrusions basée hôte et réseau sont schématisés dans la Figure 2.2. Le premier scénario montre une attaque dont l'objectif est d'obtenir un terminal avec le privilège "super utilisateur". Ensuite, le deuxième scénario détecte une utilisation non conforme d'un bit réservé du protocole IP ce qui peut révéler un canal de communication secret. Nous constatons que la représentation via des motifs facilite l'interprétation des signatures d'attaques. De plus les signatures sont facilement réutilisables par d'autres systèmes de détection d'intrusions.

Scénario 1:



Scénario 2:

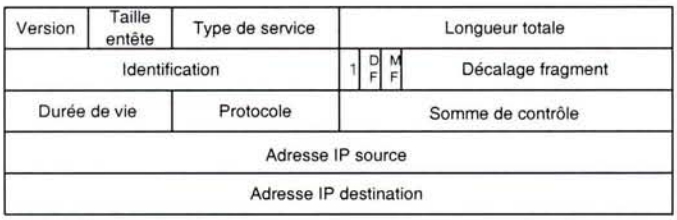


FIG. 2.2 – Détection d'intrusions avec filtrage de motifs

Dans le cadre de la détection d'intrusions basée hôte, Kuri, Navarro, Mé et Heye transforment un fichier d'audit en une chaîne de caractères où chaque lettre représente un événement particulier [88]. Par exemple ils remplacent chmod par le caractère a, ln par b, etc. La même transformation est réalisée sur les scénarios d'attaques et par suite le problème de détection se ramène en un problème de filtrage de motifs. Les auteurs adoptent une stratégie flexible de détection des scénarios d'attaques en autorisant l'insertion d'au plus k caractères. Ils proposent également deux algorithmes pour résoudre ce problème [89]. La première méthode utilise les opérations binaires pour implanter l'algorithme naïf de recherche des positions des motifs dans une trace T avec k insertions possibles [146]. Une démarche analytique limite le nombre d'insertion k à $m(\sigma/e-1)$ avec σ est la taille de l'alphabet et m est la taille du motif. La deuxième méthode de filtrage, plus rapide, compte le nombre d'apparitions de chaque caractère dans une fenêtre de taille $m+k$. L'idée repose sur le fait que pour trouver un motif dans une trace T, il faut avoir autant de caractères dans la trace que dans le motif. Cette technique élimine rapidement les zones de texte où la détection est impossible. Néanmoins elle n'assure pas le bon ordre des caractères ce qui nécessite l'utilisation d'un algorithme de filtrage exact. Par ailleurs, une démarche analytique montre que le nombre d'insertions doit être inférieure à $\sigma - m$ pour assurer une bonne qualité de filtrage.

Une méthode similaire de filtrage mais appliquée sur le trafic réseau a été proposée par Markatos et ses co-auteurs [113]. Leur algorithme appelé ExB cherche dans le contenu du paquet toutes les séquences de bits qui apparaissent dans le motif d'attaque. Afin d'optimiser la recherche, les auteurs proposent l'utilisation de l'opérateur binaire "ou" pour combiner deux séquences de bits [6]. De plus la nouvelle version ExB2 effectue le filtrage en ignorant la casse.

La méthode ExB n'affirme pas la présence d'un motif d'attaque puisque qu'elle ne vérifie pas le bon ordre des séquences de bits. Pour compléter l'opération de filtrage, les auteurs utilisent un algorithme standard de filtrage d'un seul motif. Cette méthode est efficace lors de la recherche d'une seule signature d'attaque. Seulement les règles de détection d'attaques définissent simultanément plusieurs signatures. De plus, le nombre de règles ne cesse de s'accroître à cause des nombreuses attaques découvertes chaque jour. Il en découle qu'exécuter plusieurs fois l'opération de filtrage sans tenir compte des similarités entre les motifs constitue une stratégie défaillante. Fisk et Varghese [51] puis Coit et ses co-auteurs [25] proposent des méthodes de filtrage simultané de plusieurs signatures d'attaques. Leurs stratégies permettent d'améliorer la détection d'un rapport qui varie entre 1.02 à 3.32. Néanmoins et afin de tenir compte de l'arrivée désordonnée des paquets, ces techniques procèdent à un rassemblement puis une organisation du trafic. Nous proposons dans le Chapitre 5 une technique de filtrage à la volée qui, en tenant compte d'une arrivée quelconque des paquets, cherche simultanément plusieurs signatures d'attaques.

2.3.3 Avantages et inconvénients des méthodes de détection

Nous avons présenté les deux principes de la détection d'intrusions : l'approche comportementale et l'approche basée sur la connaissance. D'une part l'approche comportementale détecte toute deviation importante dans le comportement habituel d'un utilisateur. Diverses techniques de détection existent à savoir l'approche statistique, l'apprentissage automatique, l'approche immunologique, la spécification des programmes et la théorie d'information. D'autre part, la détection basée sur la connaissance cherche directement les activités malveillantes en s'appuyant sur les descriptions des attaques connues. Parmi les techniques employées nous citons les systèmes experts, les automates, les algorithmes génétiques, la fouille de données et le filtrage de motifs. Nous résumons dans le Tableau 2.2 les avantages et les inconvénients de ces deux principes de détection.

2.4 Limites de la détection d'intrusions basée réseau avec filtrage de motifs

Nous avons présenté dans la Section 2.3 les deux principes de la détection d'intrusions puis montré les avantages et les inconvénients de chaque approche. Nous nous focalisons dans la suite aux techniques de filtrage et d'application rapide des règles de détection d'attaques. Nous nous intéressons dans la section 2.4 aux problèmes de filtrage des motifs dans le cadre d'une détection d'intrusions basée sur la connaissance. Ensuite, nous présentons les difficultés de la détection d'intrusions en ligne lorsqu'elle porte sur un trafic réseau.

2.4.1 Problèmes du filtrage de motifs

La détection d'intrusions via le filtrage des motifs d'attaques est largement déployée dans divers IDSs réseaux. Elle permet de détecter rapidement la présence d'une signature d'attaque dans le contenu d'un paquet. Seulement les attaquants essaient de contourner les systèmes de détection d'intrusions en s'appuyant sur des attaques d'évasion ou en générant un grand nombre de faux positifs. Nous discutons dans la suite ces deux problèmes.

	Détection par anomalie	**Détection basée sur la connaissance**
Avantages	– Détection de nouvelles attaques	– Explication facile des scénarios d'attaques – Génération minimale des faux positifs
Inconvénients	– Apprentissage inexacte : si ce dernier se base sur des données supposées être normales mais contenant des attaques inconnues – Evolution du comportement normal ce qui exige un apprentissage adaptatif $\Longrightarrow$ risque d'un apprentissage progressif initié par un intrus – Génération abondante des faux positifs – Sélection cruciale des paramètres utiles lors de la phase d'apprentissage puisque des paramètres en trop représentent un bruit alors que ceux en moins dégradent la qualité de détection. – Définition difficile des seuils exacts d'anomalie – Fixation difficile des durées nécessaires à l'apprentissage	– Non détection de nouvelles attaques – Expertise requise pour construire efficacement des signatures d'attaque – Mise à jour continue de la base de règles – Augmentation rapide du nombre de règles qui généralement manquent d'abstraction – Délais important entre la découverte des attaques et la définition des signatures [103].

TAB. 2.2 – Comparaison des deux principes de détection d'intrusions

2.4.1.1 Attaques d'évasion

Les attaquants tentent d'éviter les systèmes de détection d'intrusions qui se basent sur le filtrage de motifs. Ils emploient diverses techniques d'évasion (Figure 2.3) qui peuvent être classées en 3 catégories [132] :

- Evasion : elle consiste à envoyer des données qui seront ignorées par le système de détection d'intrusions mais prises en compte par la machine destinataire.
- Insertion : elle se base sur le principe inverse de l'évasion c'est à dire les données sont analysées par l'IDS mais ignorées par la machine.
- Déni de services : elle consiste à attaquer directement le système de détection d'intrusions

pour réduire ses performances ou le désactiver complètement. Le succès de cette attaque permet de suspendre l'analyse du trafic sans rompre la connectivité du réseau. Ainsi, les attaquants réduisent le risque de détection.

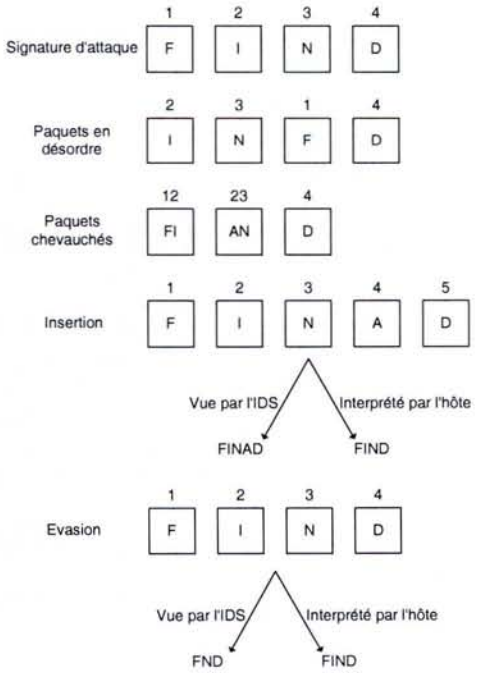


FIG. 2.3 – Techniques d'évasion

2.4.1.1.1 Techniques d'évasion

Les attaquants disposent de plusieurs moyens pour réussir des attaques d'évasion [132]. Ils forgent les champs des protocoles mis en œuvre lors des communications réseaux. Nous présentons dans le Tableau 2.3 ces diverses techniques en fonction du niveau protocolaire choisi pour mener l'attaque.

Les attaquants habiles forgent leurs propres paquets en combinant les différentes techniques d'évasion évoquées ci-dessus. Cette tâche est également à la portée des novices puisque il existe beaucoup d'outils qui automatisent les techniques d'évasion. Notons en particulier Fragrouter [153] et Whisker [133]. Fragrouter est un outil pour fuir les systèmes de détection d'intrusions. Il fragmente les paquets à différents niveaux protocolaires. De son coté, Whisker est un scanneur de vulnérabilités des serveurs web. Il s'appuie sur diverses tactiques d'évasion que nous résumons ainsi :

- Encodage des URL en hexadécimale ou unicode
- Utilisation des doubles slashes
- Utilisation du répertoire père
- Utilisation du répertoire courant
- Fin prématurée des requêtes
- Edition spéciale des messages HTTP (par exemple remplacer les espaces par des tabulations)

Niveau protocolaire	Techniques d'évasion	Connaissance
Tous	champs erronés : somme de contrôle, bits réservés, taille de la trame, valeurs non permises (version IP différente de 4 et 6), etc.	Système d'exploitation
Physique	Adresse physique inexistante	Topologie réseau
Réseau	TTL insuffisante pour que le paquet aboutisse à la machine destinataire	Topologie réseau
	Fragmentation et désorganisation des paquets IP	Réassemblage au niveau IDS
	Paquets à non fragmenter (DF) de tailles assez grandes pour pouvoir franchir les futurs liens réseaux	Topologie réseau
	Fragments IP qui se chevauchent	Système d'exploitation
	Insertion des options IP susceptibles d'être refusées. (exemple : routage par la source)	Système d'exploitation
Transport	Segmentation et désorganisation des segments TCP	Réassemblage niveau IDS
	Segments TCP qui se chevauchent	Système d'exploitation
	Insertions des options TCP non négociées durant la phase d'ouverture de connexion	Système d'exploitation
Application	Les techniques d'évasion sont spécifiques à chaque protocole applicatif. L'analyse protocolaire présentée au Chapitre 6 permet de traiter indépendamment chaque protocole	Implantation des services

TAB. 2.3 – Techniques d'évasion

- Emission de longues URL
- Utilisation des anti-slashes acceptés par les serveurs web déployés sur des systèmes windows
- Modification de la casse

Ayant présenté quelques techniques d'évasion, nous expliquons dans la sous section 2.4.1.1.2 les divers moyens utilisés pour empêcher ces attaques.

2.4.1.1.2 Solutions contre les attaques d'évasion

Les systèmes de détection d'intrusions intègrent de nouveaux procédés pour mieux résister aux attaques d'évasion. Pour se défendre contre la manipulation des protocoles, les IDS collectent et normalisent le trafic. Par exemple, Snort [137, 81] définit deux pré-processeurs Frag2 et Stream4 pour rassembler respectivement les fragments IP et les segments TCP. En outre, Paxson et ses co-auteurs [66] proposent l'utilisation d'un normalisateur de trafic. Il s'agit d'un système qui s'interpose directement sur le chemin du trafic, rassemble les différents fragments puis résout les ambiguïtés au niveau IP et TCP. Cette méthode, peu déployée, a l'inconvénient d'altérer le trafic et de se baser sur le normalisateur qui démarre un point de faille puisque en cas de

panne, il rompt la connectivité du réseau. Paxson et Shankar proposent dans [147] une stratégie active de balayage pour résoudre les ambiguïtés en analysant les paquets au niveau des IDS. La reconnaissance a pour but de découvrir la topologie du réseau et de récolter les caractéristiques des systèmes d'exploitation installés sur les machines. Cependant la phase de réassemblage est toujours nécessaire au niveau des IDS. Enfin, Talcek souligne la possibilité d'une reconnaissance passive des systèmes d'exploitation des machines [170]. La solution repose sur l'observation des premiers échanges entre les hôtes pour en déduire à partir du comportement par défaut, les systèmes d'exploitation installés sur ces machines.

Pour résoudre les ambiguïtés au niveau des protocoles applicatifs, les systèmes de détection d'intrusions normalisent le trafic avant de déclencher l'opération de filtrage. Par exemple la requête HTTP "Get /usr/./passwd" se transforme en "GET /usr/passwd". Paxson [128] souligne l'intérêt des expressions régulières pour décrire les signatures d'attaques puis les intègre dans le système de détection d'intrusions Bro. Les expressions régulières sont traduites en automates déterministes utilisés pour retrouver les signatures d'attaques. Ce mécanisme de recherche, linéaire en temps, remplace les algorithmes de filtrage de type Boyer Moore connus être sous linéaire. Les performances des deux méthodes dépendent de la phase supplémentaire de normalisation du trafic appliquée lors d'un filtrage de type Boyer Moore. Par ailleurs, la transformation de plusieurs expressions régulières en un automate fini pose des problèmes d'allocation de mémoire si des milliers d'états doivent être créés. Paxson propose une construction à la volée de l'automate. La méthode intègre un mécanisme de cache des derniers états visités et borne l'allocation de l'espace mémoire.

Si les expressions régulières résistent aux insertions des caractères pour cacher les signatures d'attaques, elles ne supportent pas les techniques qui répartissent les motifs d'attaques sur plusieurs paquets. Les systèmes de détection d'intrusions ont toujours recours au réassemblage des données avant de procéder au filtrage [41, 152]. Néanmoins cette technique nécessite du temps et de l'espace mémoire pour regrouper tous les paquets et les ordonner. Par ailleurs elle présente le risque d'être exploitée négativement par un attaquant malicieux. Ce dernier sature constamment l'IDS par des séquences inachevées de paquets pour bloquer l'opération de filtrage. Nous proposons au Chapitre 5 un filtrage à la volée qui évite le réassemblage. Cette nouvelle méthode s'applique sur plusieurs motifs d'attaques.

2.4.1.2 Génération de faux positifs

La détection d'intrusions qui se base sur le filtrage de motifs souffre du problème de faux positifs. En effet, les signatures d'attaques doivent décrire précisément l'attaque pour ne pas s'appliquer sur des trafics sains. Par exemple la règle Sid 1201 [2] de Snort comporte le motif "HTTP/1.1 403" pour détecter les accès non autorisés aux pages Web. Seulement la chaîne de caractères peut apparaître dans d'autres parties du trafic sans qu'il s'agisse d'une attaque. Snort spécifie l'espace de recherche des signatures d'attaques en définissant un décalage (Offset) et une profondeur de recherche (Depth). La méthode réduit le nombre de faux positifs mais peut provoquer des faux négatifs. En effet un attaquant peut manipuler les paquets en ajoutant ou en enlevant des champs afin de modifier les positions des motifs d'attaques. Par exemple la représentation XDR du protocole RPC représente les données sous forme de mots à quatre octets. L'introduction d'un champs nul est ignorée par le protocole mais permet de décaler la représentation des autres champs significatifs. Par conséquent la technique change la position des motifs d'attaques.

[2] http ://www.snort.org/snort-db/

Paxson [152] associe deux types de contextes aux opérations de filtrage des signatures d'attaques. Le premier contexte décrit les états de connexions et les protocoles mis en œuvre. Il s'agit d'une analyse protocolaire qui améliore la qualité de détection en supervisant les échanges protocolaires entre clients et serveurs. Le deuxième type de contextes est de haut niveau. Il exploite les connaissances acquises sur la topologie réseau et les caractéristiques des machines pour analyser les événements suspects et déterminer s'ils constituent réellement des vraies attaques. Ce procédé réduit efficacement les faux positifs [103].

Snort utilise également les contextes de connexions pour vérifier les trafics TCP. Cette fonctionnalité, incorporée via le pré-processeur Stream4, réduit les fausses alertes. Elle sert à appliquer les règles de détection d'attaques uniquement sur des connexions légitimes. Nous employons dans cette thèse le même concept de contexte afin de filtrer les signatures d'attaques (Chapitre 5) et assurer l'analyse protocolaire (Chapitre 6). De plus les paquets d'une même connexion se succèdent généralement dans la même période de temps ce qui nous encourage à stocker les contextes dans des structures d'arbres balancés.

2.4.2 Problèmes de la détection basée réseau

Nous avons présenté dans la Section 2.4.1 les problèmes de la détection d'intrusions avec le filtrage de motifs. Nous nous sommes focalisés sur les attaques d'évasion et la génération de faux positifs. Dans cette section nous nous intéressons aux problèmes de l'analyse en ligne du trafic réseau. Rappelons qu'un IDS réseau utilise les sondes pour écouter le trafic à analyser. Ainsi et contrairement aux systèmes de détection d'intrusions basés hôte, un nombre réduit de sondes suffit pour superviser l'intégralité du trafic. De plus, un IDS réseau s'appuie sur ses propres ressources (mémoire, CPU, etc) et donc, ne perturbe pas le fonctionnement des machines. Enfin, il opère généralement d'une façon furtive sur le réseau ce qui rend son exploitation une tâche difficile. Néanmoins et malgré ces avantages, les IDS basés réseau se heurtent à un ensemble de difficultés que nous développons dans la suite.

2.4.2.1 Haut débit

Les nouveaux réseaux utilisent de plus en plus des liaisons rapides en Gigabits. Cependant les systèmes de détection d'intrusions sont incapables de suivre le transfert rapide des données. En effet dans des conditions de haut débit, les sondes ne captent pas l'intégralité du trafic. La tâche de l'IDS devient plus complexe dans ces circonstances de haut débit surtout s'il est nécessaire de rassembler le trafic et de mener une analyse avec mémoire d'état. Nous montrons dans le Chapitre 3 l'apport de la division du trafic pour équilibrer la charge de détection sur plusieurs IDS.

2.4.2.2 Commutation et routage asymétrique

Les commutateurs préservent la bande passante en divisant logiquement le réseau physique de l'entreprise. Cette division diminue le risque de la reconnaissance passive ce qui élève le niveau de sécurité sur le réseau informatique. Cependant, elle réduit considérablement les données interceptées par les sondes des IDS ce qui restreint l'analyse des paquets. Le même problème est soulevé suite au routage asymétrique qui tend à optimiser les routes des paquets. En effet, en routant dynamiquement les données, les paquets d'une même connexion peuvent être collectées par différentes sondes. Ceci empêche l'analyse protocolaire et la vérification avec mémoire d'états des paquets et par suite diminue l'efficacité de la détection d'intrusions.

Afin de supporter la division logique des réseaux, les administrateurs utilisent le port miroir disponible sur les nouveaux commutateurs pour recopier l'intégralité du trafic. Néanmoins cette solution n'est pas infaillible puisque la copie des données n'est pas garantie lors des surcharges réseau. Une deuxième solution plus efficace consiste à utiliser les Taps [129]. Il s'agit d'un équipement physique qui calque le trafic dans un seul sens. Le déploiement des Taps ne perturbe pas la connectivité du réseau. Cependant, afin de reconstituer les deux sens du trafic il est nécessaire d'utiliser un équilibreur de charge [120] ou un commutateur avec un port miroir [118, 119].

2.4.2.3 Chiffrement des données

Nous avons discuté au Chapitre 1 les avantages de la cryptographie pour sécuriser les réseaux informatiques. Néanmoins un système de détection d'intrusions basé réseau n'est pas capable d'analyser le contenu des paquets. Ainsi, un intrus peut dissimuler son activité en ayant recours au chiffrement des données. Le déchiffrement et l"analyse du trafic au niveau hôte constitue une solution couramment déployée pour supporter ce type de trafic.

2.4.2.4 Suivie des attaques

Les systèmes de détection d'intrusions basés hôte sont capables de suivre le déroulement des attaques et de conclure leurs réussites. Cela permet de réduire les faux positifs et de distinguer les vraies attaques qui menacent le système informatique. Cependant le suivi des attaques devient plus complexe avec une détection d'intrusions basée réseau. Une minutieuse analyse protocolaire permet d'atteindre cet objectif.

2.4.2.5 Attaques d'évasion

Les attaques par déni de services lancées contre les IDS constituent des techniques d'évasion fréquemment utilisées par les intrus. Par conséquent et afin de protéger correctement le réseau, les systèmes de détection d'intrusions doivent se protéger en intégrant des solutions algorithmiques efficaces capables de s'adapter aux caractéristiques du trafic. De plus il faut éviter les pires exécutions initiées intentionnellement par les attaquants.

2.5 Stratégie pour une détection d'intrusions basée réseau

Afin de résoudre quelques problèmes de la détection d'intrusions basée réseau, nous proposons une nouvelle stratégie de vérification du trafic. Nous divisons le processus de détection en deux étapes. Une première étape de division du trafic suivie par une étape de détection d'intrusions. La première étape tient compte des propriétés du trafic et des caractéristiques des IDS pour partager la charge de l'analyse sur plusieurs IDS. Nous discutons dans la sous-section 2.5.1 les avantages de cette division du trafic. Ensuite nous présentons dans la sous-section 2.5.2 les différentes méthodes de détection d'intrusions que nous employons durant la deuxième étape de l'analyse du trafic réseau.

2.5.1 Phase de la division du trafic

La division du trafic réseau sert principalement à mieux gérer le haut débit et à choisir la méthode de détection convenable à chaque classe du trafic. D'autres avantages liés à la tolérance aux fautes, la réduction des faux positifs et la gestion des journaux de sécurité sont explorées au fur et à mesure dans cette sous-section.

Support du haut débit : les systèmes de détection d'intrusions doivent supporter le haut débit afin d'assurer la sécurité des futurs réseaux. La rapidité du trafic réduit le temps alloué au traitement des paquets sous peine de l'augmentation du taux des paquets rejetés. La tâche de détection est de plus en plus difficile pour un seul IDS surtout s'il faut mener une analyse avec mémoire d'états. La division de la charge d'analyse sur plusieurs IDS constitue une solution efficace pour résoudre ce problème. L'idée se fonde sur une division du trafic. Ensuite, chaque classe est traitée par un IDS spécialisé. Cet IDS contiendra moins de règles de détection d'attaques et assurera des analyses protocolaires détaillées d'un ensemble réduit de protocoles.

Tolérance aux pannes : les IDS sont des systèmes ouverts en cas de panne (fail-open devices) qui analysent une copie du trafic qui passe à travers le réseau. Les attaquants essaient de les désactiver en menant dessus des attaques par déni de services. Le succès de ces tentatives facilite les futures intrusions portées sur le réseau et minimise le risque de détection.

Un diviseur de trafic oriente chaque classe du trafic vers un IDS déterminé. Le contrôle permanent des performances de ces IDS permet de détecter rapidement la baisse de performance suite à des attaques réussies. Le diviseur du trafic redirige alors les classes dédiées initialement à des IDSs endommagés vers d'autres IDS indemnes. Par suite l'analyse des paquets se poursuit afin de détecter les futures intrusions.

Efficacité de détection : la classification des paquets tient compte non seulement de la nature du trafic, mais également des serveurs fournissant les services analysés. Par exemple un flux HTTP peut être inspecté différemment selon que ce trafic est généré par un serveur Web interne de l'entreprise ou par un autre serveur de l'extérieur. Un administrateur peut réaliser une analyse protocolaire détaillée pour mieux protéger les serveurs internes et se restreindre au filtrage brut du trafic s'il est généré par des serveurs externes. Ainsi chaque type du trafic présente sa propre méthode de détection et c'est grâce à la classification qu'il est possible d'opter pour le système de détection le plus adéquat.

Par ailleurs, la division du trafic résout les problèmes liés au routage asymétrique en envoyant chaque catégorie de trafic vers le même IDS. Elle permet conjointement de réduire le nombre de fausses alertes et de détecter plus d'attaques. Par exemple lors d'une analyse protocolaire, on évite les fausses alarmes si les paquets de deux phases consécutives d'un protocole empruntent des chemins distincts. Par ailleurs la classification assure une meilleure détection des attaques en résistant à des formes d'évasion. Un attaquant peut envoyer deux segments TCP malicieusement construits et routés sur deux chemins séparés. Par conséquent, la détection doit s'effectuer sur le même IDS pour retrouver la signature d'attaque.

Déploiement des pots de miels : la division du trafic est utile pour déployer efficacement les pots de miels. En effet, elle permet de sélectionner quelques classes du trafic pour les rediriger vers des pots de miels. Cette sélection s'effectue suite à l'étude du comportement actuel des IDS et de l'activité intrusive détectée sur le réseau. Ainsi et en cernant les sources potentielles d'attaques, la partie suspecte du trafic peut être directement routée vers un pot de miel afin de mieux connaître les techniques d'intrusions et les objectifs des attaquants.

Optimisation des fichiers de sécurité : l'élaboration des journaux de sécurité constitue une des dernières étapes du processus de détection d'intrusions et elle reflète la qualité de

l'analyse effectuée sur le trafic. Néanmoins, les administrateurs sont souvent submergés par de gigantesques fichiers journaux dans lesquels des vraies attaques sont noyées au milieu de nombreux faux positifs. L'amélioration de l'efficacité de la détection évoquée précédemment et exprimée via le ROC (Receiver Operator Characteristic [3]) contribue à optimiser ces fichiers de synthèse.

Par ailleurs, la division du trafic permet d'analyser séparément chaque classe. Par suite on crée des journaux de sécurité indépendants et spécifiques aux types étudiés du trafic. Le nombre de faux positifs contenus dans les fichiers résultats se trouve divisé offrant une exploration plus aisée et ciblée des caractéristiques des attaques détectées.

Après avoir expliqué les intérêts de la division du trafic, nous présentons dans la sous section 2.5.2 les actions possibles à effectuer sur chaque classe du trafic. Nous nous intéressons en particulier à deux méthodes de détection d'intrusions que nous développons au cours de cette thèse.

2.5.2 Phase de la détection d'intrusions

Suite à la division du trafic, nous décidions du traitement que subira chaque classe du trafic. Diverses actions sont envisageables durant cette deuxième étape d'analyse :

- Simple filtrage du trafic : afin d'accélérer l'analyse, nous sélectionnons les règles possibles de détection d'attaques. Ensuite, nous inspectons le contenu des paquets à la recherche des motifs d'attaques.
- Analyse protocolaire : nous supervisons quelques classes du trafic jusqu'au niveau application ce qui permet de mieux protéger les services fournis par l'entreprise (services web, mail, ftp...).
- Direction vers un pot de miel : cet action permet d'analyser le comportement des attaquants et par suite d'apprendre leur plan d'attaques pour mieux définir les signatures d'attaques.
- Ignorer l'analyse : cette opération s'avère intéressante pour gérer les situations de surcharge réseau. Par exemple lors des pics du trafic, l'analyse peut se restreindre à des classes plus prioritaires. Nous attribuons alors à chaque classe de trafic un ordre de priorité.
- Stopper les trafics nocifs : l'action implémente une stratégie de prévention contre les attaques ou de réponse aux attaques détectées.

Nous présentons dans la Figure 2.4 les deux phases d'analyse du trafic réseau. D'abord, la division du trafic s'appuie sur des règles de division définies par l'administrateur. Ces règles dépendent des propriétés du trafic circulant sur le réseau et des méthodes de détection d'intrusions déployées sur le site de l'entreprise. Ensuite divers traitements sont possibles à chaque classe du trafic. Les résultats de ces analyses permettent de mieux comprendre les propriétés du trafic ce qui contribue à mieux formuler les règles de division du trafic et par suite adapter le diviseur.

Dans cette thèse, nous nous intéressons d'abord à la division du trafic. Nous proposons dans le Chapitre 3 un nouvel algorithme de classification. Ensuite nous abordons la phase de détection d'intrusions. Nous optimisons le filtrage de motifs en sélectionnant dans un premier temps des règles candidates de détection d'attaques (Chapitre 4) puis en intégrant une stratégie de filtrage à la volée pour éviter le rassemblement du trafic (Chapitre 5). Enfin, nous étudions l'intérêt de l'analyse protocolaire et nous l'appliquons via des arbres de décisions pour accélérer la recherche des attaques (Chapitre 6).

[3] ROC=prob(vraie détection)/prob(fausses alertes)

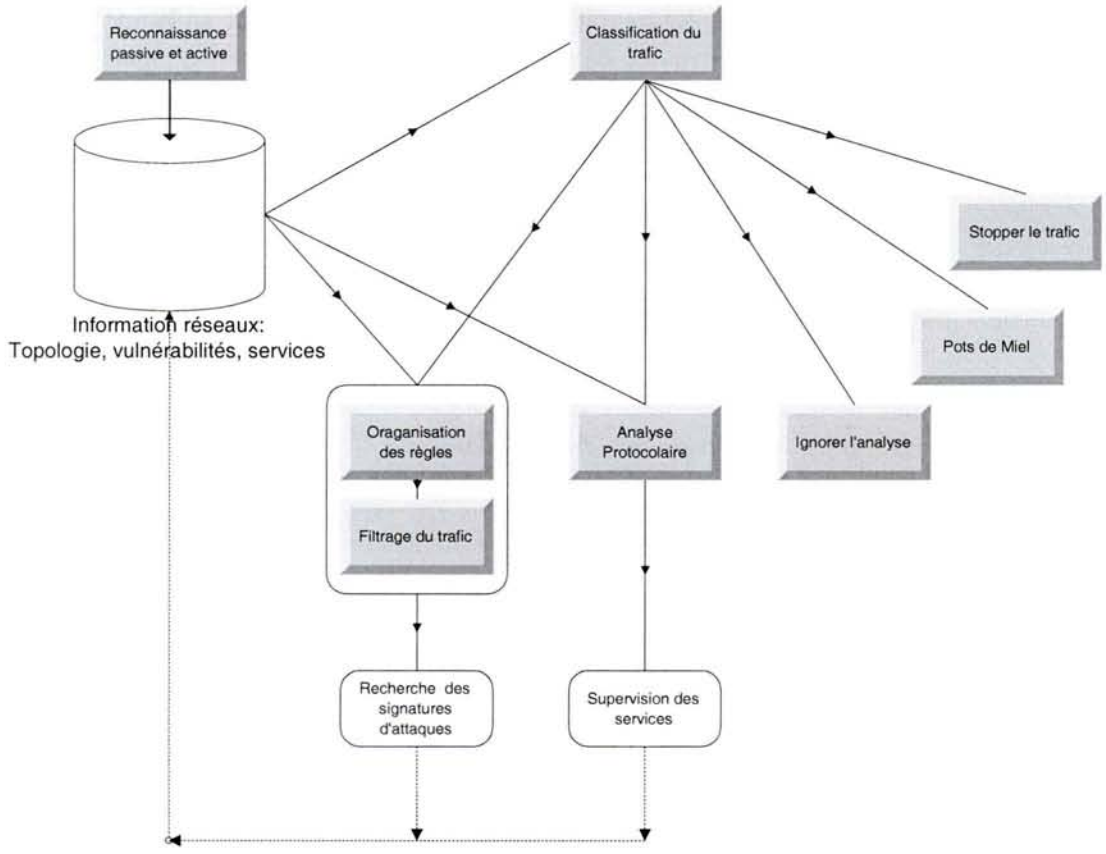


FIG. 2.4 – Stratégie d'analyse du trafic réseau

2.6 Conclusion

Nous avons présenté tout au long de ce chapitre les qualités requises des systèmes de détection d'intrusions. Afin de remplir ces objectifs, diverses méthodes de détection d'intrusions ont été proposées. Elles se basent principalement sur deux principes de détection : la détection par anomalie et la détection par la connaissance. Nous avons expliqué ces deux principes de détection et détaillé en particulier la détection d'intrusions qui emploie le filtrage de motifs. Par ailleurs nous avons souligné les limites des systèmes de détection d'intrusions basés réseau. Afin de résoudre ces limites, nous proposons une division du trafic qui redirige chaque classe du trafic vers le système de détection d'intrusions adéquat. Le prochain chapitre s'intéresse à la première phase du traitement du trafic c'est-à-dire sa division en fonction de la politique de détection de l'entreprise et des caractéristiques des IDS.

3

Division du trafic appliquée à la détection d'intrusion

Sommaire

3.1 Introduction

La classification est intéressante dans de nombreuses applications réseaux. Le routage IP, la différentiation de services et le filtrage du trafic au niveau des pares feux constituent de nos jours des domaines d'application connus. Les systèmes de détection d'intrusions peuvent tirer profit de la technique de classification afin de surmonter le handicap du haut débit. La charge du réseau sera divisée sur plusieurs IDS assurant ainsi une meilleure analyse du trafic. Par ailleurs, chaque IDS comporte uniquement des fonctionnalités s'accordant avec la classe du trafic qui lui est transmis. Il utilisera un nombre réduit de règles de détection et optimisera l'analyse selon la nature du trafic à traiter.

Nous avons montré dans la sous section 2.5.1 l'intérêt de la division du trafic pour détecter les intrusions réseaux. Cette opération s'effectue en inspectant quelques champs des paquets à savoir les adresses IP et les ports de connexions. Le problème est similaire au routage et au filtrage des paquets réalisés sur les routeurs et les pare-feux. Ainsi, nous présentons dans la Section 3.2

quelques algorithmes de classification couramment utilisées par ces outils. Ensuite la Section 3.3 montre les premiers travaux de division du trafic pour équilibrer la charge d'analyse sur plusieurs systèmes de détection d'intrusions. Ces différents travaux ne considèrent pas la politique de détection de l'entreprise ainsi que les capacités des IDS et les propriétés du trafic à analyser. Dans la Section 3.4 nous exprimons ces propriétés sous forme de règles de division du trafic puis nous soulignons le problème de chevauchement de règles. Afin de résoudre ce problème nous proposons dans la Section 3.5 un nouvel algorithme de classification qui retarde le traitement des cas de chevauchement. En effet, nous réalisons une recherche multi-bits en considérant les règles comme une suite d'octets. Durant cette analyse nous favorisons l'inspection des octets complets c'est à dire sans masque de bits par rapport aux octets masqués. La Section 3.6 présente les résultats expérimentaux de cette méthode et ceci en faisant varier le nombre de règles ainsi que le débit binaire en entrée.

3.2 Techniques de classification

La division du trafic peut être vue comme un problème de classification. Nous présentons dans cette section quelques algorithmes fréquemment utilisés dans ce domaine. Ces techniques essaient généralement de résoudre l'un des deux problèmes : trouver la règle la plus prioritaire qui s'applique ou celle ayant le plus long préfixe. Nous consacrons la première partie de cette section à présenter les algorithmes qui favorisent la règle la plus prioritaire. Ensuite nous présentons les méthodes qui cherchent la règle ayant le plus long préfixe vérifié.

3.2.1 Classification par la priorité des règles

Le filtrage des paquets au niveau des pares feux et la différentiation de services au niveau des routeurs se basent sur ce mécanisme de classification. Gupta et McKeown [64] ont étudié ces techniques de classification et ont divisé les solutions existantes en 4 catégories :

Algorithmes de recherche de base : ils utilisent les listes chaînées ou les arbres pour représenter les règles de classification. Parmi ces méthodes nous citons la recherche linéaire qui parcourt itérativement une liste chaînée de règles jusqu'à retrouver la règle la plus prioritaire. Par ailleurs la méthode Hierarchical Trie construit des arbres de préfixes (Trie) pour chaque dimension possible des règles [64]. Les arbres sont ensuite interconnectés d'ou le nom d'arbres hiérarchiques. Enfin l'algorithme "Set Pruning Trie" [175] optimise le temps de recherche de l'algorithme Hierarchical Trie. L'idée consiste à répliquer les règles pour éviter le parcours récursif des arbres d'une même dimension.

Algorithmes géométriques : selon cette famille, une règle de dimension d forme un hyper rectangle. De plus chaque paquet représente un point dans un hyper plan de dimension d et peut appartenir à un ou plusieurs hyper rectangles. Parmi les méthodes proposées dans la littérature nous citons Geometric Efficient Matching [139], Grid of Trie [163], Cross Producting [163], 2-dimentional Classifier [90], Area Based Quadtree [18] et Fat Inverted Segment Tree [50].

Méthodes heuristiques : étant donnée la grande complexité en mémoire des algorithmes géométriques, une autre catégorie d'algorithmes propose des solutions heuristiques pour sélectionner rapidement la meilleure règle de classification. Parmi ces techniques nous évoquons Recursive Flow Classification (RFC) [62], Hierarchical Intelligent Cutting (HiCut) [63] et Tuple Space Search [161].

Méthodes hardwares : ce sont des solutions de classification faciles à implanter sur des dispositifs matériels. Nous mentionnons par exemple Ternary CAM (TCAM), Bitmap Intersection [90] et Aggregated Bit Vector (ABV) [11].

Nous résumons dans le Tableau 3.1 les algorithmes de classification étudiés par Gupta et McKown et nous ajoutons quelques travaux plus récents du domaine. Nous présentons la complexité en temps et en espace de chaque méthode. Nous désignons par N le nombre total de règles, d la dimension des règles et W le nombre de bits nécessaires pour représenter les valeurs de chaque champs.

3.2.2 Classification par la longueur des préfixes des règles

La deuxième famille d'algorithmes de classification sélectionne la règle valide ayant le plus long préfixe. La technique est principalement utilisée pour le routage du trafic réseau. Deux types de recherche sont mis en œuvre : la recherche basée sur les valeurs et la recherche selon la taille des préfixes [140]. La première catégorie d'algorithmes effectue des recherches séquentielles simples ou des recherches binaires. Les structures de données employées sont généralement des arbres binaires (vérification par bit) ou des arbres N-aires (vérification par un ensemble de bits). De plus, plusieurs optimisations sont possibles et elles se résument en :

- la compression du chemin [116, 122, 150] : elle consiste à rassembler les nœuds appartenant à un même chemin et ne filtrant aucun préfixe. Ainsi on minimise le temps de recherche et l'espace mémoire en éliminant les nœuds intermédiaires.
- l'expansion des préfixes [162] : il s'agit d'assurer une longueur fixe des règles. La transformation offre l'opportunité de compresser efficacement les données. De plus, la définition d'une longueur unique des préfixes contourne le problème de recherche du plus long préfixe ce qui évite les chevauchements entre les règles.
- choix optimal des tailles des séquences multi-bits : les séquences peuvent être de taille fixe ou variable. Le choix tient compte de la représentation mémoire des mots et de la fréquence d'utilisation des préfixes. Ainsi il contribue à accélérer les accès mémoire [23].

La deuxième catégorie d'algorithmes assure une recherche sur la longueur des préfixes. La recherche du plus long préfixe peut se transformer en une recherche du plus petit intervalle valide [91]. Une autre technique consiste à stocker les préfixes de tailles égales dans la même table de hachage [182]. La recherche débute alors par rapport aux tables ayant les grands préfixes.

3.3 Division du trafic réseau pour la détection d'intrusions

Le problème de la classification a été largement exploré dans diverses applications réseau : routage, filtrage des paquets, limitation du trafic, facturation des services, etc. Nous nous intéressons dans ce travail au filtrage du trafic et à sa distribution pour détecter efficacement les intrusions réseau. Quelques travaux étudient la division du trafic réseau pour équilibrer la tâche d'analyse sur plusieurs IDS. Kruegel et ses co-auteurs [84] divisent le trafic selon les scénarios actifs d'attaques stockés sur chaque IDS. Un scénario est composé d'un ensemble d'événements qui sont décrits par des filtres au niveau du diviseur du trafic. Les auteurs soulignent que la division s'effectue également en tenant compte des adresses destinations. Il s'agit d'une stratégie simple qui maintient la possibilité d'une analyse avec sauvegarde d'état au sein de chaque IDS. Nous constatons que leur méthode tient compte des capacités des IDS. Néanmoins elle ne considère pas la politique de détection de l'entreprise, ni l'état courant des IDS. Une telle stratégie statique de division du trafic présente des limites étant donnée qu'un attaquant peut émettre un

Catégorie	**Algorithme**	**complexité en temps**	**complexité en espace**
Recherche de base	Recherche linéaire	N	N
	Hierarchical Trie	W^d	NdW
	Set Pruning Trie [175]	dW	N^d
Algorithmes géométriques	Geometric Efficient Matching [139]	$dlog_2(N)$	N^d
	Grid of Trie [163]	W^{d-1}	NdW
	Cross Producting [163]	dW	N^d
	2-dimensional classifier by Laskshaman [90]	$Wlog_2(N)$	NW
	Area Based QuadTree [18]	αW avec α est un paramètre	NW
	Fat Inverted Segment tree [50]	$(l+1)t_{RC}$ avec t_{RC} le temps de recherche sur la 1^{ere} dimension et l le nombre de niveaux de l'arbre FIS	$lN^{\frac{l+1}{l}}$
Méthodes heuristiques	RFC [62]	d	N^d
	Hierarchical Intelligent Cutting (HiCuts) [63]	d	N^d
	Tuple Space Search [161]	N	N
Méthodes Hardwares	Ternary CAM	1	N
	Bitmap Intersection [90]	$dW + \frac{N}{memWidth}$	dN^2

TAB. 3.1 – Algorithmes de classification [64]

trafic qui sera traité toujours par le même IDS. Par ailleurs les auteurs ne mentionnent pas la méthode employée pour classer le trafic. Cette opération est vitale dans des situations de haut débit mais complexe à cause des chevauchements de filtres.

Charitakis et ses co-auteurs [70] divisent le trafic pour équilibrer la charge d'analyse sur plusieurs IDS qui utilisent uniquement le filtrage brut de motifs. La division du trafic se déroule en 2 étapes. La première phase est un pré-filtrage qui élimine les paquets qui ne comportent pas de données à analyser ou qui vérifient des règles simples de détection d'attaques. La deuxième phase consiste à hacher quelques champs de chaque paquet pour choisir l'IDS qui traitera le contenu. La procédure de division empêche l'analyse avec mémoire d'état puisque les paquets d'une même connexion peuvent être routés vers des IDS distincts.

Par ailleurs TopLayer [45, 73] offre des produits commerciaux pour diviser le trafic. La documentation associée souligne l'importance de préserver l'intégrité des flux c'est à dire les paquets d'un même flux seront dirigés vers un IDS unique. Néanmoins aucune indication n'est donnée concernant les méthodes qui assurent l'intégrité des flux. De plus la division à pour but principal d'équilibrer la charge du trafic sans tenir compte des méthodes de détection appliquées par les IDS et de la politique de détection de l'entreprise. Ainsi pour appliquer cette stratégie, le diviseur du trafic emploie un algorithme de type tourniquet (Round Robin) sans définir de filtres pour classer le trafic réseau.

Nous proposons une division du trafic réseau qui respecte la politique de détection de l'entreprise, les caractéristiques des IDS et les propriétés du trafic à surveiller. Nous exprimons ces propriétés via des règles de division du trafic dont nous définissons le format dans la section suivante. Nous soulignons ensuite le problème de chevauchement de règles qui empêche une sélection aisée de la règle la plus prioritaire. Afin de résoudre ce problème, nous proposons un nouveau algorithme de classification que nous appliquons ensuite pour diviser le trafic.

3.4 Règles de division du trafic

Nous présentons dans cette section les formes possibles de division du trafic qui précèdent la détection d'intrusions. Le mécanisme de division est exprimé sous forme de règles, qui une fois satisfaites, renvoient le trafic vers un IDS spécifique ou abandonne l'analyse. Une règle peut être divisée en deux parties. La première partie contient un ensemble de paramètres (champs) à chercher dans les paquets. Le nombre de ces champs est variable, mais il est souvent restreint aux adresses IP et les ports utilisés. L'ensemble est indexé par un sens indiquant la direction du trafic par rapport au réseau interne de l'entreprise (entrant/sortant). En menant la détection d'intrusions avec une analyse protocolaire, les deux directions du flux, entrant ou sortant, doivent être distinguées pour superviser et mettre à jour l'état d'exécution des protocoles applicatifs. Dans la suite on suppose que l'absence du sens dans une règle exprime sa bidirectionnalité.

La deuxième partie d'une règle exprime la priorité de la règle et l'action à entreprendre pour traiter la classe du trafic. On peut diriger les paquets d'une classe vers un IDS particulier, un pot de miel, stopper le trafic ou ignorer tout simplement son analyse. La Règle 3.1 représente le format général d'une règle de classification.

$$\mathcal{R} \ : \ (champ_1, champ_2 ... champ_n)_{sens} \rightarrow \ \langle \ action, \ priorite \ \rangle \tag{3.1}$$

$$\text{avec} \ \ \text{sens} \in \{\text{entrant}, \ \text{sortant}\}$$

3.4.1 Catégories des règles de classification

Nous présentons dans la suite trois types de filtres à appliquer sur le trafic réseau. Les filtres sont ensuite composés pour construire des règles de division plus complexes.

3.4.1.1 Type (adresse, port)$_{sens}$

La première catégorie de règles se présente sous la forme $(adresse, port)_{sens} \rightarrow action, priorite$. Le modèle est fréquemment utilisé pour superviser les services fournis par un serveur interne de l'entreprise. Par conséquent, les IDS concernés sont généralement ceux qui sont capables de mener une analyse protocolaire détaillée du trafic réseau. Nous donnons dans la suite quelques exemples d'utilisation de ce filtre.

$R_1 : (adr\_serveurs\_http,\ ports\_http) \rightarrow IDS_1, priorite_1$ avec
adr_serveurs_http : ensemble d'adresses sur 32 bits des serveurs Web appartenant au réseau de l'entreprise,
ports_http : ensemble de ports HTTP,
IDS_1 : IDS dédié à analyser les flux HTTP.

$R_2 : (adr\_serveurs\_rpc,\ ports\_rps) \rightarrow IDS_2, priorite_2$ avec
adr_serveurs_rpc : ensemble d'adresses sur 32 bits appartenant au réseau de l'entreprise
ports_rpc : ensemble de ports utilisés par les services basés RPC
IDS_2 : IDS dédié à l'analyse du protocole RPC.

$R_3 : (adr\_machine\_ssh,\ port\_ssh) \rightarrow NUL, priorite_3$ avec
adr_machine_ssh : adresse d'une machine utilisant le ssh,
port_ssh : port utilisé lors du tunneling,
NUL : l'action choisie est d'ignorer l'analyse du flux. Les données de cette classe sont cryptées et donc il est inutile d'examiner le trafic avec un IDS basé réseau sans disposer des clés nécessaires de déchiffrement.

3.4.1.2 Type (adresse, *)$_{sens}$

La deuxième catégorie de règles présente la forme $(adresse, *)_{sens} \rightarrow action, priorite$ et s'emploie généralement pour adapter le degré d'analyse aux adresses des réseaux distants.

$R_4 : (adr\_reseau\_1, *) \rightarrow IDS_3, priorite_4$ avec
adr_reseau_1 : préfixe d'une adresse IP qui dénote l'adresse d'un réseau distant,
IDS_3 : le réseau adr_reseau_1 est un réseau de confiance. Une analyse superficielle de son trafic est confiée à l'IDS$_3$. De plus, lors des situations de surcharge, cette classe de trafic est moins prioritaire et sera ignorée en premier.

$R_5 : (adr\_reseau\_2, *) \rightarrow Pot\_de\_Miel, priorite_5$ avec
adr_reseau_2 : préfixe d'une adresse IP qui dénote l'adresse d'un réseau distant,
Pot_de_Miel : le réseau adr_reseau_2 est suspect et il a été trop souvent l'origine des attaques. La classe du trafic est alors dirigée vers un pot de miel, déployé volontairement pour acquérir les expériences des attaquants.

3.4.1.3 Type (*, port)$_{sens}$

La troisième catégorie de règles se présente sous la forme $(*, port)_{sens} \rightarrow action, priorite$. Elle sert à superviser quelques services réseaux. L'analyse s'effectue avec des signatures d'attaques

comportant un ensemble de motifs à filtrer. La classe de trafic ne s'appuie pas sur une analyse protocolaire et par suite elle concerne uniquement un sens bien déterminé du flux.

$R_6 : (*, port\_http) entrant \rightarrow IDS_4, priorite_6$ avec
port_http : ensemble de ports HTTP,
IDS_4 : l'IDS_4 emploie le filtrage de motifs pour superviser le flux HTTP entrant.

3.4.1.4 Généralisation

La composition des 3 modèles nous permet de dériver d'autres filtres plus compliqués. Par exemple afin de superviser tous les transferts de zones reçus par un serveur DNS secondaire, nous utilisons la règle suivante :

$R_7 : (adr\_1, port\_1)(*, port\_2) \rightarrow IDS_5, priorite_7$ avec
Adr_1 : adresse sur 32 bits du serveur DNS secondaire,
Port_1 : port 53 qui correspond au port utilisé pour échanger les données,
Port_2 : port 53 qui représente le port utilisé par les serveurs DNS primaires
IDS_5 : un système de détection d'intrusions capable d'analyser le protocole DNS.

3.4.2 Discussion

Les adresses IP utilisées par les règles de division du trafic sont des mots de 32 bits désignant des machines particulières du réseau. Par ailleurs, les préfixes d'adresses dénotent des adresses réseaux. Cet adressage ne se restreint pas au schéma classique définissant des classes de type A (réseau de taille 2^{24}), de type B (réseau de taille 2^{16}) et de type C (réseau de taille 2^8) mais utilise des tailles quelconques de préfixes (le concept CIDR : Classless InterDomain Routing). Chaque préfixe peut être vu comme un intervalle d'adresses et inversement un intervalle sur w bits peut être représenté par au plus $2w$-2 préfixes [64]. L'utilisation des préfixes crée alors des chevauchements entre les plages d'adresses et par suite complique la recherche de la règle adéquate de division du trafic. En fait, il ne s'agit pas de trouver la première règle qui s'applique mais de choisir la règle la plus prioritaire. Par exemple, la Figure 3.1 montre deux règles, qui malgré leurs différences apparentes, se chevauchent sur certains paquets.

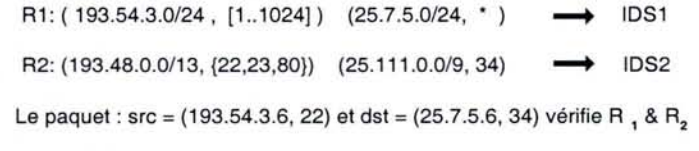


FIG. 3.1 – Chevauchements des règles de classification

Un problème similaire surgit en manipulant les ports de connexions. En effet, le champs "port" défini par une règle de division du trafic peut avoir une valeur unique sur 16 bits ou être un ensemble ou un intervalle d'entiers. Étant donnée la marge des valeurs prises par les deux ports, source et destination, diverses règles de division peuvent être satisfaites simultanément. Nous attribuons un ordre de priorité pour chaque règle de division et nous optons ensuite pour la règle la plus prioritaire si plusieurs d'entre d'elles sont vérifiées. Cette stratégie est analogue à celle des pare feux qui choisissent la première règle définie dans le fichier de configuration. Dans ce type de recherche, l'ordre d'apparition des règles fixe la priorité. Cependant notre problématique est différente de celle des routeurs qui cherchent la règle ayant le plus long préfixe pour router les paquets reçus.

Nous présentons dans la Section 3.5 une nouvelle méthode de division du trafic qui s'appuie sur des règles menues de différentes priorités. Nous appliquons une stratégie qui favorise la vérification des valeurs explicites par rapport aux valeurs masquées, vues comme des intervalles d'entiers. Ceci nous permet de réduire le nombre de chevauchements lors de la compilation des filtres et par suite de construire un graphe de classification simple garantissant une division rapide du trafic.

3.5 Notre algorithme de division du trafic réseau

Nous expliquons dans cette section notre méthode pour classer le trafic réseau. Nous considérons des règles de division du trafic avec 4 champs : adresse source, adresse destination, port source et port destination. Il s'agit d'un cas particulier d'application de notre algorithme puisque le nombre de paramètres peut être étendu en utilisant d'autres critères de classification. La division se déroule en 2 étapes. La première étape effectue une recherche géométrique dans un espace à 2 dimensions. On extrait alors les règles candidates qui définissent les mêmes ports source et destination que ceux du paquet reçu. Ces règles sont stockées dans un graphe dont le parcours permet de choisir la règle la plus appropriée pour rediriger le trafic ce qui constitue la deuxième phase de division du trafic (voir Figure 3.2). Nous expliquons dans les sous-sections 3.5.1 et 3.5.2 la construction de la matrice et du graphe de classification nécessaires pour diviser le trafic.

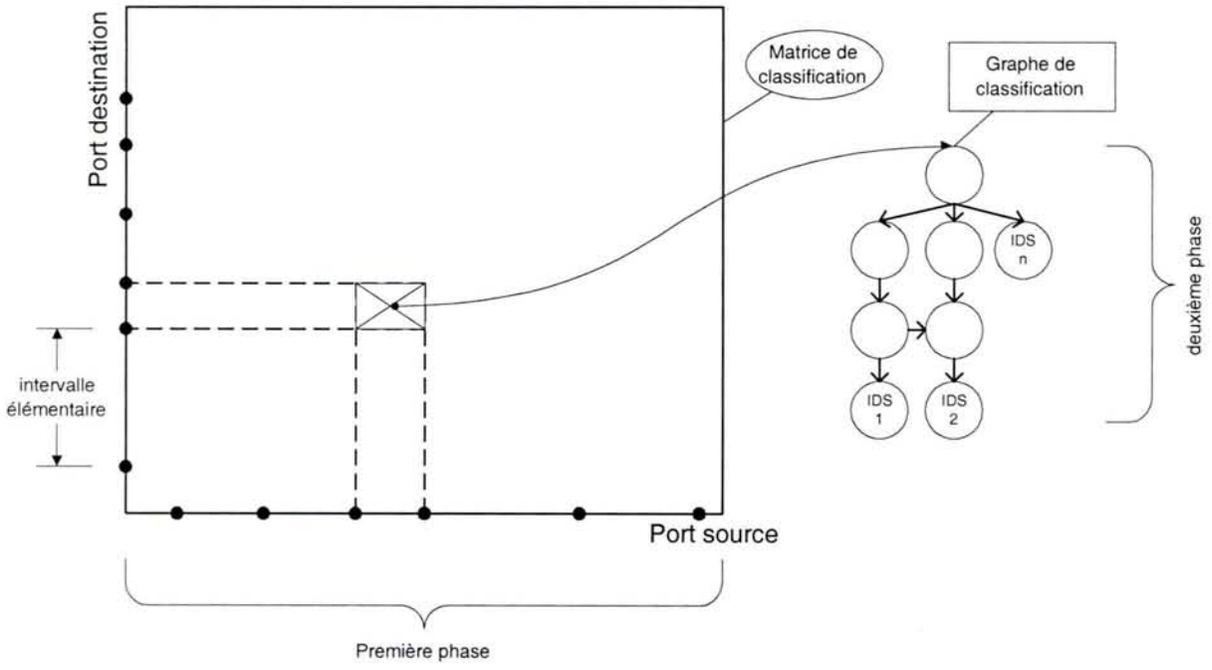


FIG. 3.2 – Classification du trafic réseau

Notre algorithme de classification s'applique à des paquets contenant des adresses et des ports de connexions. Cependant certains fragments IP ne comportent pas une entête TCP ou UDP ce qui empêche la sélection d'une règle de division du trafic. Afin de supporter ce genre de trafic, nous sauvegardons le résultat de la classification du premier fragment IP. Nous associons à cette décision les adresses de la connexion et l'identificateur du paquet. Nous nous basons ensuite sur ces paramètres pour rediriger les prochains fragments du même paquet.

3.5.1 Matrice de classification

La matrice de classification définit pour chaque couple de ports, source et destination, un ensemble de règles candidates qui vérifient ces paramètres. La construction de la matrice s'appuie sur les valeurs des ports spécifiées dans les règles de division du trafic. Ces valeurs peuvent être de simples entiers allant de 0 à 2^{16} ou des ensembles et des intervalles d'entiers. Les intervalles se chevauchent ce qui complique la construction de la matrice de classification (Figure 3.3). La solution que nous retenons consiste à diviser chacune des 2 dimensions (les ports source et destination) en intervalles élémentaires. Un intervalle est défini par ses bornes inférieure et supérieure. Durant la recherche on ne considère que l'une des 2 bornes, par exemple la borne inférieure. La méthode de recherche consiste à localiser la plus grande borne inférieure qui précède la valeur du port utilisé dans le paquet. Cette opération s'effectue en temps sous linéaire en nombre d'intervalles élémentaires. Ensuite ayant récupéré les coordonnées sur la matrice de classification, on accède directement à l'ensemble des règles candidates qui seront représentées sous la forme d'un graphe acyclique orienté.

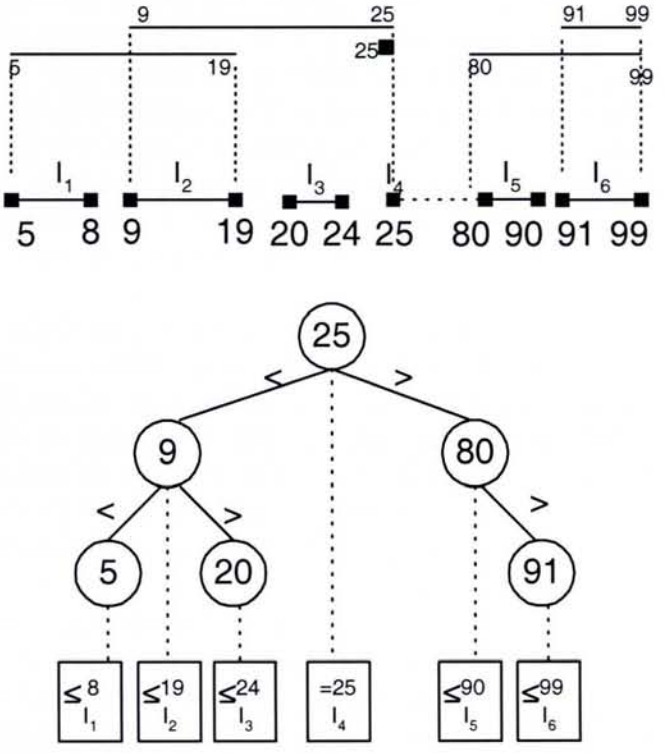


FIG. 3.3 – Recherche des intervalles de ports

3.5.2 Graphe de classification

La première étape de la classification utilise les deux champs, port source et port destination, pour initialiser la recherche de la règle convenable pour rediriger le trafic. A l'issue de cette étape nous sélectionnons un ensemble de règles candidates. La deuxième phase de la classification vérifie les autres champs du paquet pour choisir la règle la plus prioritaire parmi les règles candidates. Le nombre des paramètres inspectés durant cette étape est variable mais nous nous restreignons dans la pratique à deux critères : l'adresse IP source et l'adresse IP destination. Nous expliquons dans la suite la construction du graphe de classification. D'abord, nous étudions les champs adresses contenus dans les règles de division du trafic. Ensuite nous définissons les structures de données nécessaires pour construire le graphe. Enfin nous montrons la méthode de construction et la stratégie du parcours du graphe de classification.

3.5.2.1 Structures de données

Nous commençons cette partie par expliquer la représentation des champs utilisés pour classer le trafic réseau. Nous nous intéressons en particulier aux adresses IP contenues dans les règles de division du trafic. Les autres paramètres (s'ils existent) seront considérés comme des suites d'octets.

3.5.2.1.1 Représentation des adresses IP

Les adresses IP version 4 forment des mots sur 32 bits et désignent généralement des machines accessibles via le réseau. On utilise aussi la notation préfixe pour représenter un sous réseau ou un ensemble de machines. Ces préfixes sont considérés comme des intervalles d'adresses et par suite ils se chevauchent avec d'autres préfixes ou avec de simples adresses IP définies sur 32 bits. Par conséquent les règles de classification qui utilisent des adresses IP sources et destinations pour diviser le trafic interfèrent sur certains paquets. Ceci complique la tâche de la classification puisqu'il ne suffit pas de trouver une règle dont les paramètres vérifient le contenu du paquet mais de sélectionner la règle la plus prioritaire parmi toutes les règles possibles. Un parcours naïf de la liste des règles est une stratégie inefficace étant donné les parties communes entre les paramètres d'une part et d'autre part les différentes priorités des règles qui peuvent être exploitées pour optimiser la recherche.

Afin de représenter les adresses IP, fréquemment utilisées dans les règles de division du trafic, nous avons pris comme unité l'*octet*. En effet, la recherche multi-bits accélère la division puisque 4 opérations au maximum suffisent pour vérifier une adresse IPv4 au lieu de 32 tests nécessaires sur des arbres binaires. Néanmoins la concaténation des bits produit une réplication des règles et par suite une augmentation de la consommation mémoire. Par exemple si on utilise un bloc de 2 bits, le filtre 1* est représenté dans les deux branches 10 et 11 de l'arbre multi-bits. Afin de surmonter ce problème, nous distinguons entre deux types d'octets : les octets masqués et les octets complets (Figure 3.4) :

- Octet masqué : le nombre de bits fixés à 0 ou à 1 est inférieur à 8. On utilise alors un masque pour marquer les bits définis. Ce nombre varie entre 0 et 7.
- Octet complet : les 8 bits de l'octet prennent une valeur précise de 1 ou 0. L'octet complet est alors équivalent à un octet masqué avec un masque de taille égale à 8.

La complexité du problème de la classification est due essentiellement aux chevauchements des règles. Alors que les différents algorithmes proposés dans la littérature traitent successivement et indépendamment chaque dimension des règles, nous proposons une nouvelle stratégie

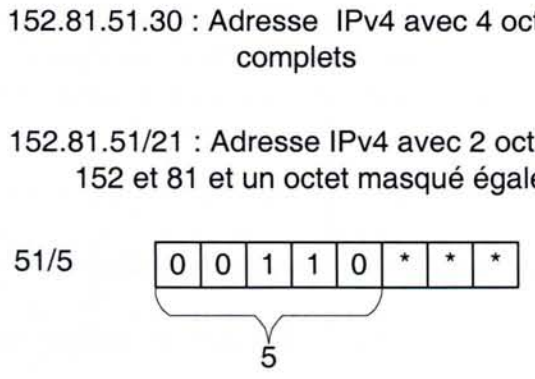


FIG. 3.4 – Format d'un octet masqué

qui, quelque soit l'ordre de la dimension, favorise l'analyse des octets complets aux octets masqués. Cette organisation simplifie le problème de la classification en réduisant le nombre de cas de chevauchement. La Figure 3.5 contient deux règles de division qui ne se chevauchent pas, bien que leurs premières dimensions interfèrent. Notre idée consiste alors à fusionner toutes les dimensions de recherche et à organiser différemment les octets pour inspecter les octets complets avant les octets masqués. En effet en vérifiant la suite des octets complets, le nombre de règles encore candidates sera minime ce qui réduit le nombre de chevauchements d'une part et diminue la réplication des règles d'autre part. Par exemple la figure 3.5 montre qu'en inspectant le premier octet de la deuxième dimension (de valeur 2 et 3) avant le deuxième octet de la première dimension, on évite la division en intervals élémentaires et la réplication des règles constatées dans la partie A de la même Figure.

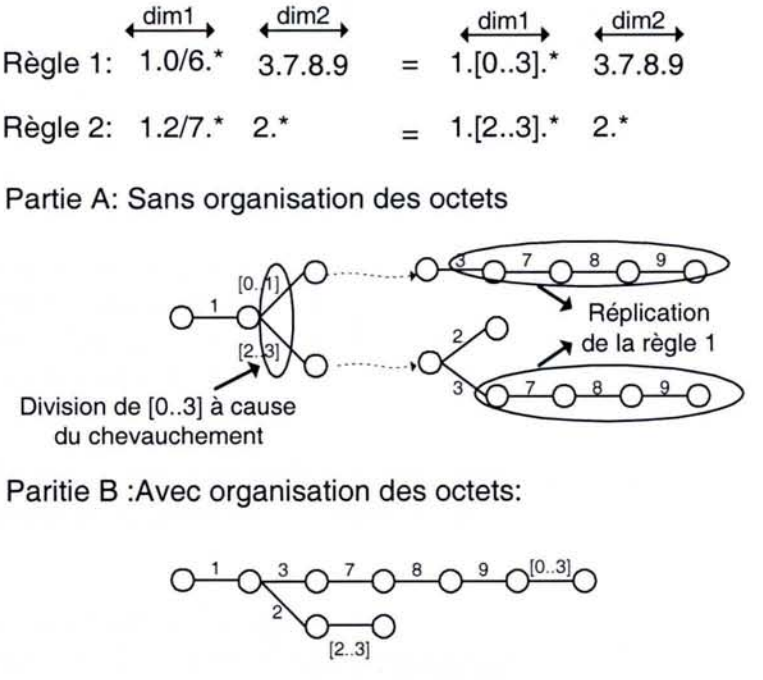


FIG. 3.5 – Organisation des octets complets et masqués

La réorganisation des octets impose l'introduction d'une information supplémentaire associée à chaque octet pour sauvegarder sa position initiale. Nous définissons alors la structure de données "info" sous forme d'un couple de couples d'entiers pour stocker simultanément la position et la valeur de chaque octet.

Définition 3.5.1 (Structure info) *La structure info associée à chaque octet est un couple (position, valeur) défini par :*

- *position= (champs, ordre) où champ est la dimension examinée de la règle et ordre est le numéro de l'octet vérifié.*
- *valeur=(octet, masque) où octet est la valeur de l'octet et masque est le nombre de bits masqués.*

Exemple : Dans la Figure 3.5, la première dimension de la règle 1 contient 2 octets ayant respectivement les structures infos ((1,1),(1,8)) et ((1,2),(0,6)).

L'organisation des octets des règles de classification revient à ordonner les structures infos associées à ces octets. Nous définissons alors la relation d'ordre $\prec$ sur les structures infos.

Définition 3.5.2 (Relation d'ordre $\prec$) *Soient $info_i$, $i \in \{1,2\}$ deux structures infos. On note $masq_i = info_i.valeur.masque$, $chp_i = info_i.position.champs$ et $ord_i = info_i.position.ordre$. La relation binaire $\prec$ est la relation d'ordre définie par*

$$info_1 \prec info_2 \text{ ssi } \begin{cases} masq_2 < masq_1 \text{ ou} \\ masq_1 = masq_2 \text{ et } chp_1 < chp_2 \text{ ou} \\ masq_1 = masq_2,\ chp_1 = chp_2 \text{ et } ord_1 < ord_2 \end{cases}$$

Exemple :

- ((1,1),(1,8)) $\prec$ ((1,2),(0,6))
- ((1,1),(1,8)) $\prec$ ((2,1),(1,8))
- ((1,1),(1,8)) $\prec$ ((1,2),(1,8))

3.5.2.1.2 Représentation des règles de division du trafic

Une règle de division du trafic est composée d'une suite d'octets complets et masqués. En représentant chaque octet par sa structure $info$, nous transformons la règle en une liste de structures $info$. Ensuite et afin d'organiser les octets, nous appliquons la relation d'ordre $\prec$ sur les éléments de cette liste. Nous obtenons une suite ordonnée de structures info qu'on appellera chemin d'une règle.

Définition 3.5.3 (Chemin d'une règle de classification) *Soit r une règle de division du trafic. Le chemin de la règle r est la liste ordonnée des structures infos selon la relation d'ordre $\prec$.*

Exemple : la liste des structures infos associée à la règle de classification 1 de la Figure 3.5 est ((1,1),(1,8)) : ((1,2),(0,6)) :((2,1),(1,8)) :((2,2),(2,8)) :((2,3),(3,8)) :((2,4),(4,8)). Le chemin de la règle est alors ((1,1),(1,8)) : ((2,1),(1,8)) :((2,2),(2,8)) :((2,3),(3,8)) :((2,4),(4,8)) :((1,2),(0,6)).

3.5.2.1.3 Représentation du graphe de classification

Nous introduisons dans cette sous section les structures Nœud, Etiquette et Graphe nécessaires pour construire le graphe de classification. De plus nous définissions le chemin d'un nœud qui permet de distinguer entre deux catégories de règles candidates, les règles candidates primaires et les règles candidates secondaires. Afin d'expliquer le contenu de chaque structure, nous donnons un apercu sur la construction du graphe de classification. En effet, nous considérons d'abord les chemins des règles candidates pour former un arbre de préfixes des chemins. Nous utilisons à ce stade la terminologie des arbres pour exprimer les relations entre les nœuds de l'arbre. En particulier nous nous intéressons aux notions de chemin et ancêtre. Ensuite, nous complétons l'arbre par des liens d'échec qui relient des nœuds appartenant à des chemins distincts de l'arbre. Nous transformons ainsi l'arbre initial de préfixes des chemins en un graphe de classification. La Figure 3.6 montre le début du graphe. Les nœuds indiquent les prochaines positions à inspecter sur une dimension particulière du paquet. De plus ils précisent la nature des futurs octets à vérifier c'est-à-dire des octets complets ou masqués. Enfin ils contiennent deux listes de règles candidates. Ainsi la structure nœud possède les quatre champs suivants :

- Type : un nœud de type C indique que le prochain octet est complet tandis qu'un nœud de type M signifie que la prochaine inspection concerne des octets masqués. Enfin un nœud de type F est final et contient la règle gagnante de division du trafic,
- Dimension : cette variable indique la dimension à considérer sur les règles de division du trafic,
- Enregistrement de pointeurs : c'est un tuple d'entiers dont la taille est égale à la dimension des règles de division. Il indique la progression de la recherche sur les diverses dimensions des règles,
- Listes des règles candidates : elles contiennent les règles de division vérifiées sur ce nœud.

Définition 3.5.4 (structure nœud) *Un nœud du graphe indique une position de vérification possible sur une dimension déterminée des règles de division. Il est représenté via l'enregistrement* $(Type, Dim, Rec, (l_1, l_2))$ *où Type est le type du nœud, Dim est la dimension de la prochaine inspection, Rec est l'enregistrement des positions déjà vérifiées et l1 et l2 sont deux listes de règles candidates. Dans la suite on notera* $\mathcal{N}$ *l'ensemble des nœuds du graphe.*

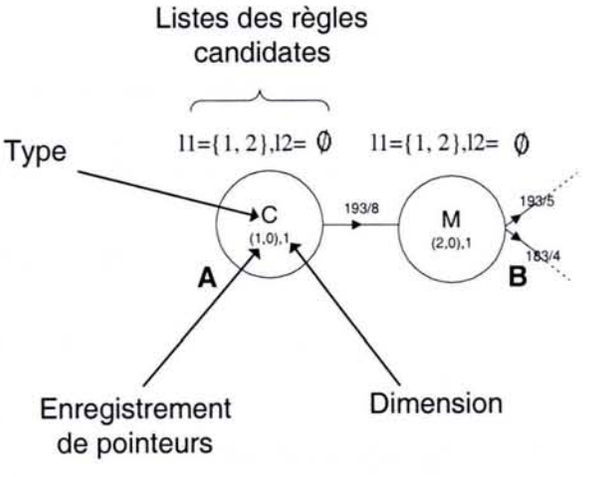


FIG. 3.6 – Structure nœud du graphe de classification

Chaque nœud du graphe contient deux listes de règles candidates. Les octets à la prochaine position de vérification définis par ces règles, forment les liens issus du nœud. L'étiquette d'un lien contient la valeur de l'octet et le nombre de bits non masqués. En particulier un octet complet aura 8 bits non masqués. Dans la suite nous appelons ρ la fonction qui retourne pour chaque couple de nœuds, l'étiquette de leur lien s'ils sont reliés.

Définition 3.5.5 (structure étiquette) *L'étiquette d'un lien du graphe est un couple label=(l_valeur, l_masque) où l_valeur est une valeur possible d'un octet (0..2^8-1) et l_masque est le nombre de bits non masqués (1..8). Une étiquette spéciale ϵ est introduite pour exprimer un lien qui peut être toujours emprunté lors du parcours du graphe. Dans la suite on notera $\mathcal{L}$ l'ensemble des étiquettes possibles.*

Ayant défini les structures nœud et étiquette, nous présentons dans la suite la structure "graphe" utilisé pour diviser le trafic réseau.

Définition 3.5.6 (structure graphe) *Soit ρ la fonction partielle de $\mathcal{N} \times \mathcal{N}$ dans $\mathcal{L}$ définissant les liens du graphe. Le graphe de classification $G_{(\mathcal{N},\rho)}$ est un graphe acyclique orienté dont les liens sont étiquetés par $\mathcal{L}$. Le graphe admet un seul nœud racine N_0 (sans prédécesseur) minimum pour $\prec$. Par ailleurs les nœuds finaux sont les nœuds de type F : ils contiennent la règle de division la plus prioritaire parmi les règles candidates.*

Nous définissons dans la suite le chemin d'un nœud du graphe qui, comme pour le chemin d'une règle, est composé d'une suite de structures infos.

Définition 3.5.7 (Chemin d'un nœud) *Le chemin d'un nœud N du graphe de classification est une liste de structures infos dont les positions sont celles des nœuds intermédiaires entre le nœud racine et le nœud N et les valeurs sont les étiquettes des transitions prises.*

Exemple : Dans la Figure 3.6, le chemin du nœud A est la liste vide [ ] quant au chemin du nœud B est ((1,1),(193,8)).

Les chemins des règles de division du trafic et des nœuds du graphe sont composés par des structures infos. Nous présentons dans la Définition 3.5.8 des opérations possibles sur les listes d'infos. Ces opérations permettent ensuite de distinguer entre deux types de règles candidates sur un nœud du graphe.

Définition 3.5.8 (Opération sur les listes d'infos) *Soient $liste_1$ et $liste_2$ deux listes d'infos alors :*

1. $(info_1 : liste_1) =_c (info_2 : liste_2)$ *ssi* $info_1 = info_2$ *et* $liste_1 =_c liste_2$,
2. • $[\,] \prec_c (info_1 : liste_1)$
   • $(info_1 : liste_1) \prec_c (info_2 : liste_2)$ *ssi* $info_1 \prec info_2$ *ou* $info_1 = info_2$ *et* $liste_1 \prec_c liste_2$,
3. $liste_2$ *est une sous séquence de* $liste_1$ *ssi pour tout élément* $info_2$ *de* $liste_2$, *il existe un élément* $info_1$ *de* $liste_1$ *tel que :*
   - $info_1.position = info_2.position$ *et*
   - $info_1.valeur \subseteq info_2.valeur$

   Notons que l'inclusion des $info_i.valeur$ s'applique naturellement puisque le champs valeur représente un couple (octet, masque) ce qui correspond à un intervalle d'entiers.

Exemple :

- ((1,1),(1,8)) :((1,2),(0,6)) $=_c$ ((1,1),(1,8)) :((1,2),(0,6))
- ((1,1),(1,8)) :((2,1),(1,8)) $\prec_c$ ((1,1),(1,8)) :((1,2),(2,8))
- ((1,1),(1,8)) :((1,2),(1,7)) *sous sequence de* ((1,1),(1,8)) :((1,2),(1,8))

Le chemin d'un nœud du graphe permet de diviser ses règles candidates en deux catégories : les règles primaires et les règles secondaires. Intuitivement, une règle est primaire dans un nœud si le chemin de ce nœud est un début du chemin de la règle. Par contre la règle est dite secondaire si le chemin du nœud est un cas particulier de celui de la règle. En utilisant les opérations sur les listes d'infos, nous caractérisons dans la Définition 3.5.9 les règles primaires et secondaires d'un nœud du graphe. Nous stockons ensuite la première catégorie de règles dans la liste l_1 des règles candidates du nœud et le deuxième ensemble de règles dans la liste l_2.

Définition 3.5.9 *Soit r une règle candidate appartenant à un nœud N du graphe de classification alors :*

- *r est une règle primaire dans N ssi le chemin de N est un préfixe du chemin de r,*
- *r est une règle secondaire dans N ssi r n'est pas primaire dans N et le chemin de N est une sous sequence du chemin de r.*

La distinction entre les règles primaires et secondaires évite la réplication des règles. En effet, chaque règle candidate possède au moins un chemin sur le graphe sur lequel elle est primaire (Preuve 3.1). Ainsi, la construction des liens qui se base uniquement sur les règles primaires est possible puisque en cas d'échec sur ces chemins, on peut transiter via des pointeurs d'échec vers les chemins primaires des règles secondaires. Les liens d'échec sont étiquetés par ϵ. Nous expliquons dans la section suivante la méthode de construction.

Preuve 3.1 : Soit R_2 une règle candidate secondaire sur un chemin L_1. Notons que R_2 est primaire sur le nœud racine N_0 du graphe puisque le chemin de N_0 (liste vide) est un préfixe du chemin de R_2. Donc il existe un nœud sur L_1 sur lequel R_2 devient secondaire.

Soit N le dernier nœud de L_1 dans lequel R_2 est primaire. Puisque R_2 devient secondaire alors à cette position de vérification, R_2 contient un octet masqué. Par construction, on favorise à cette position l'octet complet ayant la plus petite structure info selon $\prec$. De plus on ajoute un lien étiqueté ϵ pour vérifier le prochain octet complet de R_2 (ca sera alors sur une autre dimension de R_2). Ce nouveau chemin construit et qu'on nomme L_2 constitue le chemin primaire de R_2.

Exemple : La Figure 3.7 montre le graphe de classification de 3 règles de division du trafic. Nous représentons à l'intérieur de chaque nœud le type, la dimension et l'enregistrement des positions visitées. Par ailleurs, nous schématisons au dessus de chaque nœud les listes des règles candidates primaires et secondaires. Les liens sont étiquetés par les valeurs des octets et le nombre de bits masqués. Notons que si le lien représente un octet complet alors le nombre de bits masqués est 8 que nous évitons de montrer sur le graphe.

Le chemin du nœud A est ((1,1),(193,8)) :((1,2),(54,8)). Ce nœud possède uniquement la règle 2 comme primaire. Les deux règles 1 et 3 sont aussi satisfaites dans A et appartiennent à la liste des règles candidates secondaires.

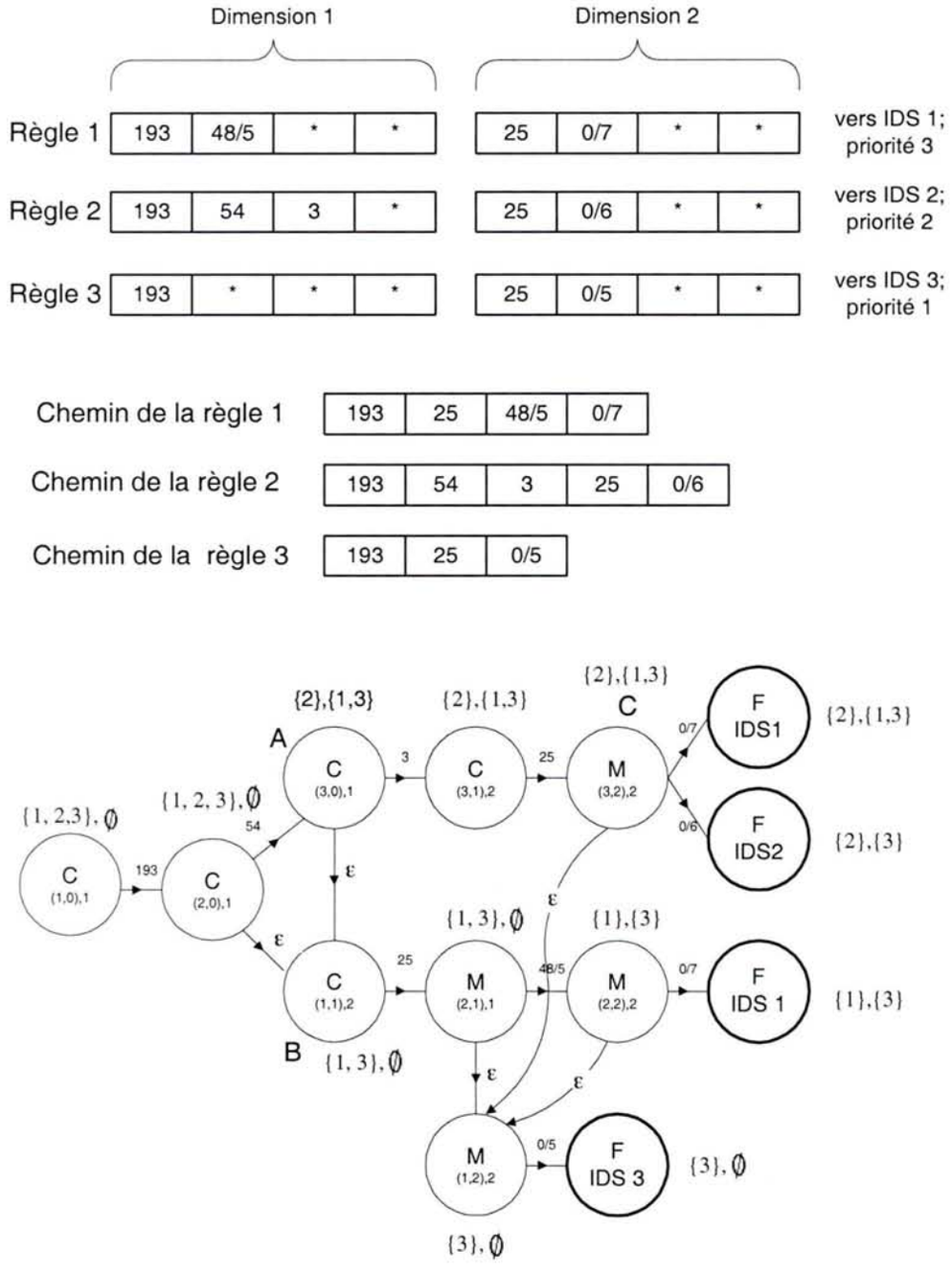


FIG. 3.7 – Graphe de classification du trafic

3.5.2.2 Algorithmes de construction du graphe de classification

Nous avons présenté les structures de données nécessaires pour diviser le trafic réseau. Nous nous intéressons dans la suite à la construction effective du graphe de classification. Cette étape se déroule pendant l'initialisation du diviseur du trafic et donc elle n'influe pas le traitement des paquets lors de la détection intrusions. Nous détenons un ensemble de règles candidates issues de la matrice de classification. Notre but est de construire le graphe de classification qui se présente comme un arbre de préfixes des chemins des règles candidates, complété par des liens d'échec.

La première étape consiste à construire l'arbre de préfixes. Nous considérons les chemins des règles candidates ordonnés via la relation d'ordre $\prec$. Nous disposons de deux méthodes de construction. La première méthode est itérative et elle consiste à construire le chemin de chaque règle, considérée a part. Il s'agit de la même stratégie adoptée par l'algorithme d'Aho-Corasick [5] pour le filtrage de motifs (à la différence prés qu'on traite une séquence d'intervalles ordonnée via la relation $\prec$ et non pas une suite de caractères). Néanmoins cette méthode est inefficace étant donnée la priorité des règles. Par exemple la construction du chemin de la règle 193.82.* de priorité 5 est inutile s'il existe une règle égale à 193.* de priorité 10. Nous optons alors pour une construction récursive du graphe de classification qui traite position par position, toutes les règles candidates.

D'abord, nous considérons le nœud racine N_0 dans lequel toutes les règles candidates issues de la matrice de classification sont primaires. En effet le chemin de N_0 est la liste vide, et donc il est préfixe des chemins des règles candidates. Nous cherchons ensuite la structure info minimum selon $\prec$ extraite des chemins des règles primaires. Nous déterminons alors les valeurs des champs *Dim* et *Rec* du nœud N_0. La connaissance de la position de vérification permet de typer N_0. En effet, si la règle la plus prioritaire primaire ou secondaire au nœud N_0 est vérifiée alors ce dernier est de type F. Sinon, si à la position définie par N_0, il existe une règle candidate primaire qui contient un octet complet alors le nœud est de type C, sinon il est de type M. Ensuite, nous construisons à partir du nœud N_0 de nouveaux liens étiquetés par les octets des règles candidates primaires. De plus nous ajoutons un lien étiqueté ϵ si il existe une règle candidate primaire qui définit une nouvelle position différente de celle choisie au nœud N_0. L'ensemble des liens construits permet de créer de nouveaux nœuds auxquels nous appliquons la même procédure de construction réalisée sur N_0.

Nous présentons dans Algorithme 1 la fonction récursive *Construire* qui appliquée à un nœud N déjà typé du graphe, permet de générer de nouveaux nœuds, de les typer, puis de leur appliquer de nouveau la fonction *Construire*. La construction commence par le nœud racine N_0 et s'arrête lorsque nous obtenons des nœuds finaux. Nous créons ainsi l'arbre de préfixes des chemins des règles candidates en tenant compte des priorités des règles.

Notre stratégie de construction se base sur deux hypothèses, considérer uniquement les structures infos des règles candidates primaires puis vérifier la faisabilité des règles secondaires sur les chemins primaires. Deux contraintes surgissent à ce niveau :

1. Le lien construit en se basant sur une règle primaire (par exemple étiqueté 0/6=[0..4]) est plus général que la structure info d'une règle secondaire (par exemple avec une valeur 0/7=[0..2]). On ne peut pas vérifier à ce stade la faisabilité de la règle secondaire.
2. Le parcours sur un chemin du graphe échoue car les données à classer ne correspondent pas aux structures infos des règles primaires. Néanmoins, ces données vérifient des règles secondaires du chemin considéré.

Ces deux contraintes nous force à compléter l'arbre des préfixes déjà construit, ce qui constitue la deuxième étape de la construction du graphe de classification. La première contrainte ne concerne que les nœuds de type M car si le lien défini par la règle primaire est plus général alors nécessairement il représente un octet masqué. Afin de résoudre ce problème, nous divisons les intervalles des règles primaires en intervalles élémentaires par rapport aux octets des règles secondaires. Ainsi nous vérifions sur chaque intervalle élémentaire la faisabilité des règles secondaires. Par exemple, le nœud C de la Figure 3.7 divise l'octet masqué 0/6 ([0..4]) de la règle candidate primaire 2 en deux intervalles 0/7 ([0..2]) et 0/6 ([0..4]) en vérifiant d'abord l'intervalle 0/7 ([0..2]).

Construire(nœud) :
début
  si *nœud.type* $==$ *C* **alors**
    pour chaque *octet complet* $\mathcal{O}$ *des règles primaires* **faire**
      Ajouter un lien étiqueté $\mathcal{O}$ vers un nouveau nœud $\mathcal{S}'$
      Ajouter les candidats primaires et secondaires à $\mathcal{S}'$
      Typer $\mathcal{S}'$
      `Construire` ($\mathcal{S}'$)
    si *il existe un octet masqué* **alors**
      Ajouter un lien étiqueté ϵ vers un nouveau nœud $\mathcal{S}'$
      Ajouter les candidats primaires et secondaires à $\mathcal{S}'$
      Typer $\mathcal{S}'$
      `Construire` ($\mathcal{S}'$)
  sinon si *nœud.type* $==$ *M* **alors**
    pour *taille du masque allant de 1 à 7* **faire**
      pour chaque *octet masqué* $\overline{\mathcal{O}}$ *des règles primaires* **faire**
        Diviser $\overline{\mathcal{O}}$ en intervalles élémentaires I_i par rapport aux octets des règles secondaires
        pour chaque *Intervalle* I_i **faire**
          Ajouter un lien étiqueté I_i vers un nouveau nœud $\mathcal{S}'$
          Ajouter les candidats primaires et secondaires à $\mathcal{S}'$
          Ajouter un liens étiqueté ϵ vers le dernier état qui interfère avec $\mathcal{S}'$
          Typer $\mathcal{S}'$
          `Construire` ($\mathcal{S}'$)
  sinon si *nœud.type* $==$ *F* **alors**
    Affecter au nœud la règle la plus prioritaire
fin

Algorithme 1: Construction du graphe de classification : fonction Construire

La deuxième contrainte exige la complétion du graphe de classification par des liens d'échec entre certains nœuds. Les pointeurs d'échec représentent en fait d'autres alternatives pour diviser le trafic en suivant d'autres chemins plus généraux que les premiers chemins empruntés sur le graphe. Nous utilisons les informations des premiers chemins pour retrouver les chemins possibles sur le graphe. Par ailleurs, nous relions deux nœuds par un lien d'échec que si ils n'appartiennent pas au même chemin. Cette condition évite la création de cycles sur le graphe de classification. Par exemple, le lien d'échec entre les deux nœuds N et M_1 de la Figure 3.8 est interdit. Enfin, et pour empêcher le même type d'échec, nous nous assurons que le nœud pointé par le lien d'échec possède des liens supplémentaires par rapport au nœud d'échec. Ainsi, le lien entre les nœuds N et M_2 de la Figure 3.8 est irréalisable. Nous introduisons dans la Définition 3.5.10 la fonction $fail$ qui tient compte de ces propriétés pour calculer les pointeurs d'échec convenables entre les nœuds du graphe. Nous utilisons dans cette définition la terminologie des arbres (nœud ancêtre) car la base de notre graphe de classification est un arbre de préfixes c'est à dire un cas particulier des arbres.

Définition 3.5.10 (Fonction *fail*) *Soit N un nœud du graphe. La fonction fail appliquée à N retourne un nœud M ayant le plus petit chemin selon* $\prec_c$ *vérifiant les propriétés suivantes :*

1. *le chemin de M est une sous séquence du chemin de N*
2. $M \notin$ *ancêtres de N*
3. *les liens issus de* $M \not\subseteq$ *liens issus de N*

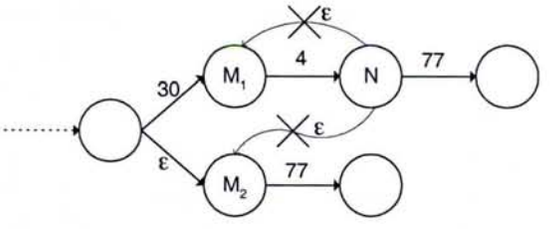


FIG. 3.8 – Pointeurs d'échec

Exemple : Dans la Figure 3.7, fail(A)=B car le chemin de B est plus petit chemin selon $\prec_c$ vérifiant :

- chemin de B = ((1,1),(193,8)) est sous séquence du chemin de A = ((1,1),(193,8)) ; ((1,2),(54,8)) ;
- B $\notin$ ancêtres de A ;
- liens issus de B={((1,3),(3,8))} $\not\subseteq$ liens issus de A= {((2,1),(25,8))}.

Nous présentons dans Algorithme 2 la fonction *Completer* qui exécute la fonction *fail* sur chaque nœud du graphe ce qui permet de compléter le graphe classification.

```
Completer() :
début
    pour chaque nœud N du graphe de classification faire
        construire un lien étiqueté ϵ de N vers fail (N)
fin
```

Algorithme 2: Complétion du graphe de classification : fonction Completer

3.5.2.3 Parcours du graphe de classification

Nous résumons dans cette sous section la construction du graphe de classification pour montrer sa complétude et extraire ensuite la stratégie adéquate pour parcourir le graphe.

Notre idée de base consiste à favoriser les octets complets sur les octets masqués. Ceci correspond à ordonner les positions de vérification selon la relation d'ordre $\prec$. Cette construction se poursuit jusqu'à verifier la règle la plus prioritaire ou terminer toutes les positions de vérification. Ainsi en inspectant toutes les positions possibles de vérification, nous assurons que toutes les règles de classification sont analysées et que leurs chemins sont représentés sur le graphe de classification. Ensuite notre méthode de complétion du graphe utilise la relation $\prec_c$ sur les chemins. Elle consiste à relier via des pointeurs d'échec des nœuds des chemins spécifiques (formés par des octets complets et des octets masqués à grands masques) à des chemins plus généraux (comportant des octets masqués à petits masques). Cette construction nous impose de commencer la vérification par les chemins spécifiques pour pourvoir transiter ensuite vers les chemins

généraux. Par conséquent, nous favorisons durant le parcours du graphe les liens étiquetés par les valeurs explicites aux liens étiquetés par ϵ. Par ailleurs, nous empruntons les liens avec les longs préfixes (grand masque=petit intervalle) avant de considérer les petits préfixes. Nous optons pour ce choix puisque seulement la vérification d'un petit intervalle permet d'induire la vérification d'un intervalle plus grand qui le contienne. Nous présentons dans Algorithme 3 la fonction *Parcourir* qui permet de traverser le graphe de classification.

```
Parcourir() :
début
    nœud = nœud racine du graphe
    si nœud.type == C alors
        si un lien d'un octet complet est vérifié alors
            Faire la transition
            nœud = nouveau nœud
        sinon si il existe un lien étiqueté ϵ alors
            Faire la transition
            nœud = nouveau nœud
        sinon
            Rejeter le paquet
    sinon si nœud.type == M alors
        pour taille du masque décroissant de 7 à 1 faire
            si un lien d'un octet masqué est vérifié alors
                Faire la transition
                nœud = nouveau nœud
                Continuer
        si il existe un lien étiqueté ϵ alors
            Faire la transition
            nœud = nouveau nœud
        sinon
            Rejeter le paquet
    sinon si nœud.type == F alors
        Rediriger le paquet
fin
```

Algorithme 3: Parcours du graphe de classification : fonction Parcourir

Notre méthode de classification bénéficie d'un temps de réponse proportionnel dans le pire des cas à la taille de la plus grande règle en octets complets et masqués. On note par d la dimension des règles, chp_i le i^{eme} champs d'une règle et N le nombre total des règles de classification. De plus r désigne une règle de division du trafic et R constitue l'ensemble de toutes les règles. On utilise $|\ |_o$ pour exprimer la longueur d'un champs en octet. La complexité moyenne en temps pour retrouver la règle de division du trafic la plus prioritaire est évaluée à $\sum_{r \in R} \sum_{i=1}^{d} |\ r.chp_i\ |_o\ /N$.

La complexité en espace est égale dans le pire des cas à $\sum_{r \in R} \sum_{i=1}^{d} |\ r.chp_i\ |_o$. Cette complexité n'est

jamais atteinte en pratique vu la présence des octets en commun entre plusieurs règles de division du trafic. Par ailleurs l'utilisation des préfixes d'adresses réduit le nombre d'octets complets ce qui engendre moins de nœuds sur le graphe. Notons que nous donnons une complexité moyenne en temps car sa valeur exacte dépend des priorités des règles. Avec un bloc de bits qui correspond à un octet, la complexité de notre algorithme est plus petite d'un facteur de 8 que celle offerte par l'algorithme *Set Pruning Trie*. Nous évitons également la réplication des règles engendrée par ce dernier algorithme en favorisant les valeurs explicites aux valeurs masqués d'une part, et en distinguant entre les règles primaires et les règles secondaires d'autres part. Cette dernière optimisation nous force à compléter le graphe par des pointeurs d'échec, ce qui ressemble au principe de l'algorithme *Grid of Trie*. Néanmoins cet algorithme tient compte uniquement de 2 dimensions afin de connecter les arbres de la deuxième dimension de recherche.

3.6 Résultats expérimentaux

Nous présentons dans cette section les résultats expérimentaux obtenus lors de la division du trafic. Nous nous intéressons dans un premier temps à la phase de construction de la matrice et des graphes de classification. Ensuite nous testons les performances du diviseur en fonction du nombre de règles et du débit binaire en entrée.

Les différentes expériences sont réalisées sur le réseau de notre laboratoire le LORIA. Nous montrons dans la Figure 3.9 les propriétés de ce trafic. Nous remarquons que la majeure partie du trafic concerne des données HTTP, FTP, SSH, DNS et NNTP. Nous nous basons alors sur ces flux pour créer des règles de division du trafic. Ces règles sont stockées dans un fichier de configuration puis traitées par le diviseur du trafic pendant sa phase d'initialisation.

Par ailleurs nous déployons notre diviseur du trafic sur une machine Linux sur laquelle nous réalisons une architecture de réseaux virtuels (VLAN). Nous nous appuyons sur le protocole 802.1Q qui permet le marquage des trames Ethernets. De plus nous utilisons l'utilitaire vconfig pour configurer les réseaux virtuels. Ainsi nous créons sur la même machine 4 interfaces virtuelles. Nous assignons à chaque interface une adresse IP et nous déployons dessus une instance du système de détection d'intrusions Snort 1.9. Chaque IDS est configuré différemment en fonction des règles de classification qui dirigent le trafic à analyser. Le premier IDS tient compte des attaques Web et s'intéresse particulièrement aux flux HTTP émis par des serveurs Web du LORIA. Le deuxième IDS analyse le trafic HTTP généré par des serveurs Web de l'extérieur. Nous distinguons la supervision des deux flux afin de détailler plus l'inspection du trafic reçu par les serveurs internes. Cette analyse se traduit par l'activation d'un nombre supérieure de règles de détection d'attaques et peut évoluer vers une analyse protocolaire avec un autre IDS. Le Troisième IDS analyse les trafic FTP, TELNET, X11, DNS et NNTP alors que le quatrième IDS traite les autres flux non classés.

L'implémenation actuelle du diviseur du trafic se base sur la librairie Libpcap pour capter les paquets. Nous utilisons ensuite la matrice et les graphes de classification pour choisir la règle de division du trafic la plus prioritaire. Nous déterminons alors l'IDS qui analysera le paquet puis nous employons la bibliothèque Libnet pour renvoyer le trafic sur son interface virtuelle. La solution Libpacp/Libnet est retenue pour tester le comportement du diviseur de trafic. Néanmoins, nous envisageons améliorer le système en s'appuyant directement sur Netfilter [48]. Ce dernier maintient des accroches (hooks) sur le trajet du paquet à travers la pile des protocoles. Sur chacun de ces points, Netfilter définit diverses actions pour traiter les paquets. Nous pouvons associer à ces actions, notre algorithme de classification.

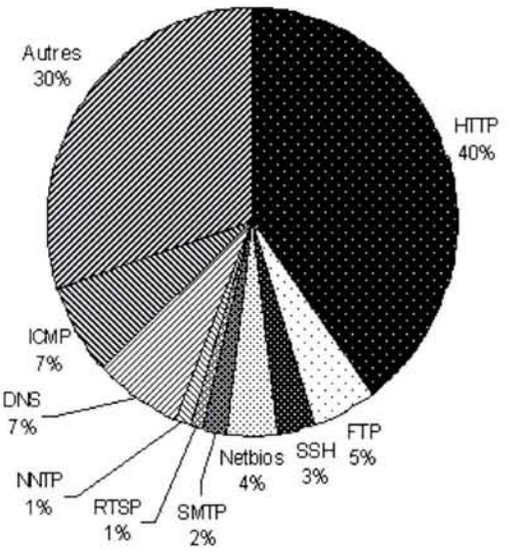


FIG. 3.9 – Composition du trafic

3.6.1 Déploiement du classificateur

L'objectif de la première série d'expériences est de mesurer le temps nécessaire pour initialiser le diviseur du trafic. Les temps obtenus correspondent à la construction de la matrice et des graphes de classification. Nous varions le nombre de règles de classification et nous mesurons le temps et la mémoire consommée dans chaque cas. Les règles utilisées traitent principalement des trafics HTTP, FTP, TELNET, SMTP, X11, DNS et NNTP que nous stockons dans un fichier de configuration en entrée du classificateur. De plus et afin de créer des chevauchements entre les règles, nous introduisons des règles qui définissent des intervalles de ports. Par exemple, on s'intéresse à l'ensemble des ports réservés (c'est-à-dire ceux inférieurs à 1024) et aux canaux IRC du chat (ports compris entre 6666 et 7000). Nous insérons également des règles avec des valeurs génériques de port source et destination ce qui correspond à des intervalles [0..65535]. Cette configuration assure la création d'une large matrice de classification qui divise les intervalles de ports en intervalles élémentaires pour résoudre les cas de chevauchement.

La construction d'une grande matrice de classification engendre une large consommation mémoire. Afin d'optimiser l'allocation mémoire, nous implantons la matrice sous forme d'une liste à deux dimensions (Figure 3.10). L'allocation mémoire sur une dimension donnée s'effectue en une seule étape ce qui permet de simuler des tableaux et d'effectuer des recherches dichotomiques rapides. Nous appliquons cette méthode de construction dans un cas général qui tient compte de plusieurs dimensions puis nous considérons lors de nos expériences le cas particulier de 2 dimensions c'est-à-dire des ports sources et destinations.

Afin d'augmenter le nombre de règles, nous divisons les adresses IP source et destination en différentes plages. Nous utilisions ensuite ces différentes valeurs pour construire le graphe de classification. Notre implémentation du graphe considère une dimension quelconque des règles. Tous les champs sont traités comme des suites d'octets complets suivies par un octet masqué. Ensuite, nous nous restreindrons dans le cadre de l'expérience à deux champs. Le Tableau 3.2 montre le temps du déploiement du classificateur alors que la Figure 3.11 présente la quantité de mémoire consommée en fonction du nombre de règles.

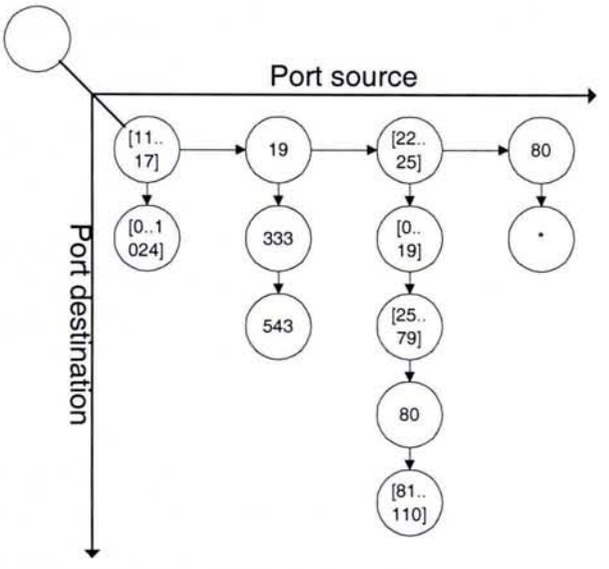


FIG. 3.10 – Matrice de classification

Nombre de règles	Nombre de cellules dans la matrice	Nombre de nœuds du DAG	Temps de Construction (en secondes)
2	5	28	0,1
10	46	411	0,1
20	74	986	0,3
50	108	2975	0,4
100	192	7615	0,51
1000	243	8702	0,681

TAB. 3.2 – Déploiement du classificateur

Le Tableau 3.2 indique que le temps d'initialisation du classificateur dépend du nombre de règles. Toutefois il reste assez court en présence d'un grand nombre de règles. Par ailleurs la consommation mémoire augmente au départ rapidement en fonction des règles. Ensuite elle tend à se stabiliser au fur et à mesure qu'on augmente le nombre de règles. Ceci s'explique par l'augmentation du nombre des octets en commun entre les règles ce qui évite la création de nouveaux nœuds dans le graphe. De plus les nouvelles règles peuvent avoir de grandes priorités et définir des préfixes d'adresses plus petits ce qui les favorisent par rapport aux anciennes règles et diminuent le nombre de nœuds dans le graphe de classification.

3.6.2 Classification du trafic réseau

Nous présentons dans cette partie notre deuxième série d'expériences qui évaluent le comportement du classificateur. D'abord nous considérons un trafic en ligne et nous calculons le temps nécessaire pour classer 10^8 paquets. Nous limitons la capture des paquets à 10^3 puis nous classons chacun 10^5 fois. A chaque exécution nous varions le nombre de règles de division du trafic et nous calculons le temps nécessaire pour la classification. La Figure 3.12 montre les différents résultats obtenus. Nous remarquons que le temps total du parcours de la matrice et du graphe de classification est assez court ce qui permet de supporter un haut débit. En effet, si la taille

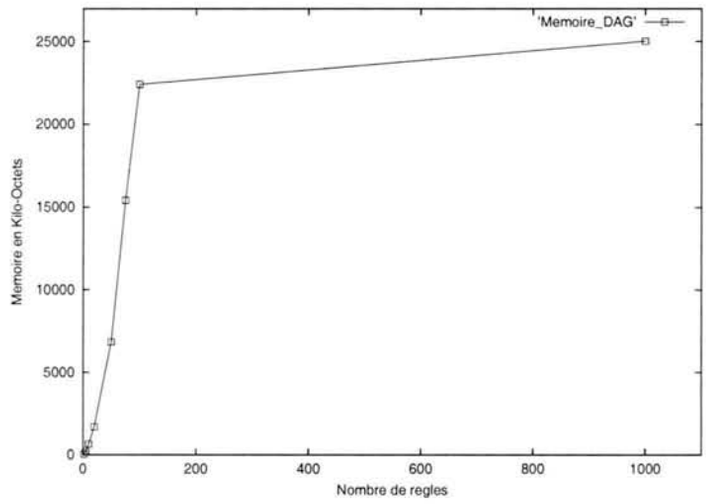


FIG. 3.11 – Consommation mémoire du classificateur

moyenne d'un paquet est de 500 Octets, on atteint avec 100 règles un débit binaire classé égale à 14,8 Gb/s. Par ailleurs, nous constatons que le temps de classification augmente légèrement en ajoutant les règles de division du trafic. Cette augmentation est due à la recherche dans la matrice de classification puisque le temps mis pour parcourir le graphe de classification est tributaire de la longueur en octet de la règle la plus prioritaire. Il est donc indépendant du nombre de règles. Ceci nous encourage à modifier notre méthode de classification en considérant les champs des ports sources et destinations comme des suites de deux octets.

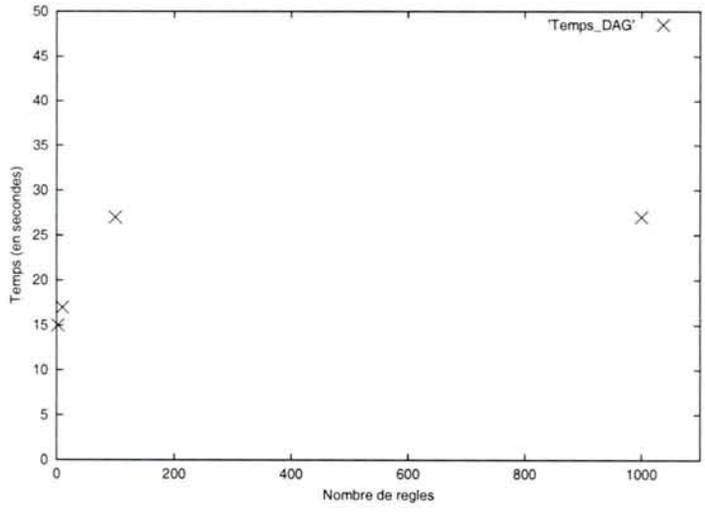


FIG. 3.12 – Temps de classification de 10^8 paquets

Nos différentes expériences sont réalisées avec une carte réseau de 100Mb/s. Cette interface constitue un handicap pour tester des débits binaires importants. Pour contourner le problème, nous enregistrons une partie du trafic dans un fichier log de taille 222 Méga-octets, puis nous régénérons le trafic à l'aide de l'utilitaire Tcpreplay. Cet outil nous offre l'opportunité d'augmenter le débit binaire jusqu'à 80Mb/s. Afin d'augmenter virtuellement le débit binaire à l'entrée du diviseur du trafic, nous classons plusieurs fois chaque paquet capté. Nous mesurons ensuite la

performance du classificateur en calculant pour chaque débit binaire en entrée, le débit binaire à la sortie. Pour modifier le débit binaire en entrée, nous varions un paramètre N qui représente le nombre de fois qu'un paquet est classé. Le débit binaire à l'entrée est alors égale à N*80Mb/s. D'autre part, le débit binaire à la sortie correspond au rapport N*(taille du log) sur le temps mis pour classer tout le trafic. Idéalement, nous devons avoir le même débit en entrée et en sortie, soit une pente égale à 1 en comparant les deux débits. Pratiquement on obtient une courbe similaire jusqu'à l'ordre de 3,1 Gb/s (Figure 3.13). Ceci constitue une garantie que le diviseur du trafic supporte un débit binaire égale à 3,1 Gb/s. Cette capacité peut être meilleure étant donnée que le classificateur s'exécute simultanément avec Tcpreplay qui accède constamment au fichier log pour régénérer le trafic.

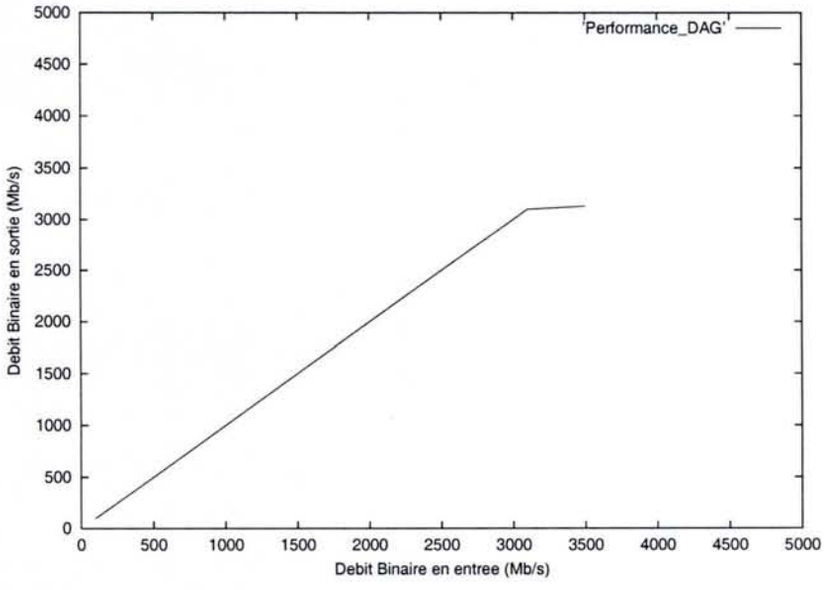


FIG. 3.13 – Performance du classificateur

3.7 Conclusion

Le problème de la classification a été largement étudiée durant ces dernières années et appliquée dans diverses applications réseau dont le routage, le filtrage, la différentiation de services et la facturation. Nous avons montré dans ce chapitre l'intérêt de la classification pour améliorer la détection d'intrusions réseau. Outre le support du haut débit, la division du trafic permet de tenir compte de la politique de détection de l'entreprise et des caractéristiques des systèmes de détection d'intrusions. Nous avons décrit ces propriétés sous forme de règles de division de trafic munies de priorité pour favoriser la supervision de quelques classes du trafic. Par ailleurs et afin d'optimiser la division, nous avons mené une recherche géométrique suivie d'une recherche multi-bits (par octet). La deuxième recherche applique une nouvelle stratégie pour réduire la réplication et le chevauchement des règles. L'idée consiste à fusionner les différentes dimensions définies par les règles de division du trafic. Ensuite on ordonne la recherche multi-bits en favorisant les blocs complets sur les blocs masqués. Les tests effectuées sur la méthode ont permis de supporter des débits allant jusqu'à 3.1 Gb/s. Néanmoins nous constatons une large consommation mémoire que nous envisageons réduire en évitant la recherche géométrique.

La division du trafic constitue la première étape de l'analyse du trafic réseau. Elle offre l'opportunité d'appliquer la méthode de détection optimale à chaque classe du trafic. Nous abordons dans le Chapitre 4 la phase de détection en cherchant les signatures d'attaques dans le contenu des paquets. Chaque signature est représentée par une règle de détection d'attaque. Cependant, le nombre de ces signatures augmente au fur et à mesure qu'on découvre de nouvelles attaques. Ainsi la vérification directe de chaque règle de détection constitue une stratégie lente. Afin d'optimiser cette recherche nous proposons dans le prochain chapitre deux méthodes d'application intelligente de règles de détection d'attaques. Ces techniques sélectionnent des règles candidates avant de les appliquer sur le trafic réseau.

4

Optimisation des règles de détection d'attaques

Sommaire

4.1 Introduction

La division du trafic aide la détection d'intrusions menée sur des réseaux à haut débit. Elle associe la méthode de détection convenable à chaque classe du trafic. Parmi ces méthodes on s'intéresse dans ce chapitre au filtrage des signatures d'attaques. Une attaque peut être décrite sous forme d'informations explicites qui caractérisent le scénario de l'attaque. Ces informations se présentent sous forme de règles de détection. Si une règle est vérifiée alors une alarme est déclenchée, une réponse active est exécutée ou une autre règle de détection est activée. Cependant le nombre de ces règles ne cesse de s'accroître à cause du nombreuses attaques découvertes chaque jour. Ainsi la vérification exhaustive de ces scénarios d'attaques via un parcours séquentiel de la liste des règles constitue une stratégie inefficace. En effet ce procédé augmente le temps de

traitement par paquet ce qui limite la capacité d'un IDS déployé en ligne pour surveiller un réseau à haut débit.

Nous étudions dans ce chapitre un nouveau schéma d'organisation des règles de détection d'attaques stockées sur un IDS. Notre objectif est d'accélérer le parcours de la liste des règles. De plus nous envisageons rassembler les motifs de plusieurs règles d'attaques dans des structures de données communes afin d'assurer un filtrage multiple des motifs. Nous définissons un schéma d'organisation des règles que nous l'appliquons sur le système de détection d'intrusions Snort 1.9. La Section 4.2 de ce chapitre présente cet IDS. Nous étudions en particulier son architecture interne, ses signatures d'attaques puis son mécanisme de détection. Ensuite nous présentons dans la Section 4.3 des travaux récents qui améliorent la détection d'intrusions avec Snort. La Section 4.4 introduit notre méthode d'organisation des entêtes des règles de détection d'attaques. Nous discutons en particulier l'organisation des RTNs (Rule Tree Node), la structure de données utilisée par Snort pour représenter la première partie des règles. La Section 4.5 étudie la seconde partie des règles. Elle correspond à la structure OTN (Option Tree Node) de Snort et elle contient des options à vérifier à partir du contenu des paquets. Les options sont généralement communes entre plusieurs règles. Nous proposons alors une factorisation de ces champs ce qui offre en plus de la détection rapide, un moyen pour ordonner les règles et rassembler les motifs similaires. Enfin nous décrivons dans la Section 4.6 les différentes expériences réalisées et nous analysons les résultats obtenus.

4.2 Le Système de détection d'intrusion Snort 1.9

Snort est un système de détection d'intrusions réseau dont le code est distribué publiquement. Développé initialement par Martin Roesch en 1999 [136], la version originale de Snort décodait simplement les données tcpdump binaires pour les afficher dans un format exploitable par l'humain. Ensuite l'IDS a connu un énorme progrès pour devenir un outil largement déployé dans le milieu académique et industriel. Les versions courantes de Snort (1.9 et 2.x) s'exécutent sous trois modes différents : un renifleur du trafic (options -d -v -e), un enregistreur du trafic (options -l -b) et un détecteur d'intrusions (option -c). La détection d'intrusions se base sur l'approche par connaissances et elle utilise le plus souvent la technique de filtrage du trafic à la recherche des motifs d'attaques. L'outil Snort peut être divisé en 5 composants (Figure 4.1), tous essentiels à la détection d'intrusions.

1. Renifleur : ce module emploie la bibliothèque Libpcap pour capter les paquets transitant sur le réseau.
2. Décodeur de paquet : le décodeur examine tour à tour chaque couche de la pile des protocoles, en commençant par la couche liaison jusqu'à la couche transport. Au cours de cet examen, il transforme le format brut des paquets captés en une structure interne de données (structure *Packet*). Cette structure sera utile pour vérifier le contenu des paquets.
3. Préprocesseurs : Snort dispose de plusieurs modules de prétraitement du trafic qui peuvent être divisés en deux catégories : modules de détection et modules de normalisation. Le premier type des préprocesseurs examine les paquets à la recherche des activités suspectes. On cite par exemple Portscan2 et Flow-Portscan pour détecter les balayage de ports et le ARP-Spoof pour détecter le trafic ARP malveillant. La deuxième catégorie de préprocesseurs normalise le trafic pour que le moteur de détection des attaques puisse y rechercher précisément les signatures. Nous évoquons en particulier les préprocesseurs Frag2 et Stream4 qui permettent de rassembler le trafic fragmenté et segmenté.

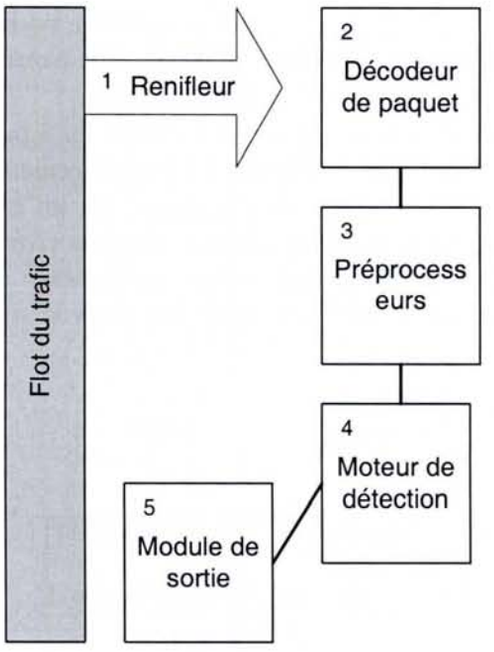


FIG. 4.1 – Architecture de Snort 1.9 [80]

4. Moteur de détection : ce module analyse le trafic en appliquant les règles de détection d'intrusions activées par l'administrateur. Le moteur utilise la structure de données *Packet* pour parcourir le système de règles. Il effectue ensuite l'ensemble des tests définis par chacune des règles. L'ordre des tests ainsi que des règles dépendent de leurs apparitions dans le fichier de configuration de l'IDS.
5. Modules de sortie : Snort intègre plusieurs modules qui assurent des réponses actives et passives aux attaques détectées. L'administrateur peut configurer l'IDS pour émettre des traps SNMP, insérer des événements dans une base de données, utiliser le deamon syslog pour ajouter des entrées dans les fichiers journaux de sécurité, interrompre les connexions TCP suspectes ou tout simplement enregistrer les alertes et les paquets malveillants qui sont à l'origine.

Par ailleurs, plusieurs outils d'administration sont développés autour de Snort. On distingue en particulier les utilitaires de configuration automatique des pare feux (SnortSam), de gestion et de corrélation des alertes (ACID, logsurfer), de configuration graphique de l'IDS (Snort center, ids center) et de gestion de l'ensemble des règles (Oinkmaster). Nous présentons dans la sous section suivante le format des règles de détection d'attaques de Snort.

4.2.1 Règles de détection d'attaques

Une règle de détection d'attaque contient les informations nécessaires pour détecter une intrusion. Ces informations se présentent comme les empreintes laissées par les attaquants. Snort dispose d'une large base de règles qui couvre les différentes étapes du scénario d'attaque présenté au Chapitre 1. Étant donnée la diversité de ces règles, plusieurs systèmes de détection d'intrusions proposent des utilitaires de transformation de ces signatures en leurs propres langages de définition d'attaques. Nous citons à titre d'exemple les IDS Bro [152], Realsecure [54] et le langage STATL [44, 43] utilisé par divers IDS comme NetStat [180]. Nous étudions dans cette sous

section le format général d'une règle de Snort. Cette description nous sert de base pour détailler le fonctionnent interne du moteur de détection de Snort 1.9. Nous montrons ensuite l'intérêt d'une organisation des règles pour accélérer l'analyse du trafic.

On peut diviser une règle de détection d'attaque de Snort en 4 parties : le type, le protocole, l'entête et le corps. On schématise dans la Figure 4.2 l'emplacement de chaque partie sur une simple règle. Cette règle est de type Alert et s'applique sur un trafic TCP. Elle inspecte le contenu des paquets issus de n'importe quelles adresses et ports (Any Any) et reçu sur le réseau 192.168.1.0/24 au port 111 (du RPC). La règle envoie un message d'alerte "mountd access" si le paquet contient le motif "|00 01 86 a5|". On généralise dans la suite le rôle de chaque partie d'une règle.

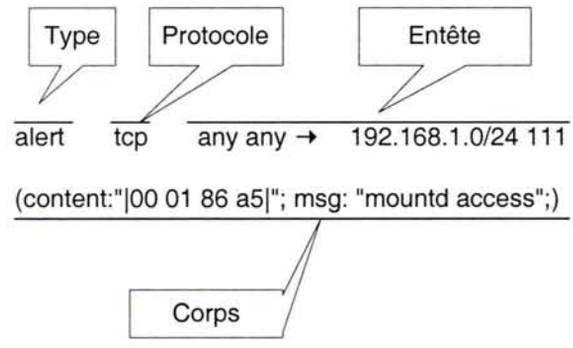


FIG. 4.2 – Règle de détection d'attaque de Snort

Type : Snort définit 5 types de base. Une règle de type *Alert* enregistre le paquet dans un fichier journal et déclenche une alarme pour avertir l'administrateur. Le type *Log* sauvegarde simplement le paquet alors que le type *Pass* ignore les paquets répondant à cette signature. Le type *Active* alerte l'administrateur et active une autre règle de type *Dynamic*. Ainsi ce dernier type permet de désactiver momentanément les règles jusqu'à ce qu'un événement soit détecté.

Protocole : ce paramètre sert à identifier le protocole auquel s'applique la règle. Actuellement, Snort gère quatre protocoles de trafic : TCP, UDP, ICMP et IP, qui sont les principaux protocoles utilisés pour le trafic Internet. L'analyse commence toujours par les règles TCP, UDP ou ICMP et en cas d'échec de détection, elle se poursuit en parcourant les règles IP.

Entête : cette partie de règle définit les paramètres des flux TCP et UDP, des messages ICMP et des paquets IP à analyser. Elle indique l'adresse source, le port source, l'adresse destination et le port destination à surveiller. De plus un opérateur de direction informe Snort du sens d'application de la règle. L'opérateur -> désigne une règle unidirectionnelle alors que l'opérateur <> indique une règle bidirectionnelle. Notons que l'entête d'une règle est stockée dans une structure spéciale appelée **RTN** (Rule Tree Node) et chaînée avec les entêtes des autres règles pour former une liste de RTNs.

Corps : c'est la partie restante d'une règle et comporte diverses options stockées dans une structure spéciale appelée **OTN** (Option Tree Node). Les options servent à la détection (TTL : durée de vie, Flag : drapeaux TCP, TOS : type de service, Content : contenu du paquet, etc), aux réponses actives des attaques détectées (mots clés REACT et RESP) et à l'archivage dans les fichiers de sécurité (LOGTO : nom du fichier log, msg : description de l'attaque, etc).

Snort v1.9 comporte 1747 règles de détection d'attaques alors que la version 2.2 est fournie avec 2508 règles. Les règles sont classées par type de trafic. On trouve alors des règles spécifiques aux serveurs de noms (DNS), aux appels de procédures distants (RPC), aux services Web, aux portes dérobées, etc. L'administrateur configure le système de détection d'intrusions pour satisfaire la politique de sécurité de l'entreprise. Il sélectionne les règles qui décrivent les caractéristiques des mauvais trafics à filtrer. Il peut par ailleurs formuler ses propres règles pour analyser d'autres types de trafic. Nous expliquons dans la sous section 4.2.2 la transformation de ces règles en une chaîne de détection, utilisée ensuite par le moteur de détection de Snort afin pour vérifier le contenu des paquets.

4.2.2 Mécanisme de détection de Snort 1.9

Durant la phase d'initialisation de l'IDS, le moteur de détection de Snort effectue l'analyse syntaxique des règles de détection d'attaques. Ensuite, il construit une liste chaînée à 3 dimensions (Figure 4.3), en tenant compte du type des règles (Alert, Pass, Log, Dynamic, Active), des protocoles utilisés (TCP,ICMP,UDP,IP) et des caractéristiques des connexions (adresse source, adresse destination, port source et port destination). Chaque règle est représentée par une structure OTN (Option Tree Node) qui sauvegarde les options contenues dans la partie corps. L'OTN est attachée à une RTN (Rule Tree Node) partagée par plusieurs OTN et qui exprime la partie entête des règles (Figure 4.3).

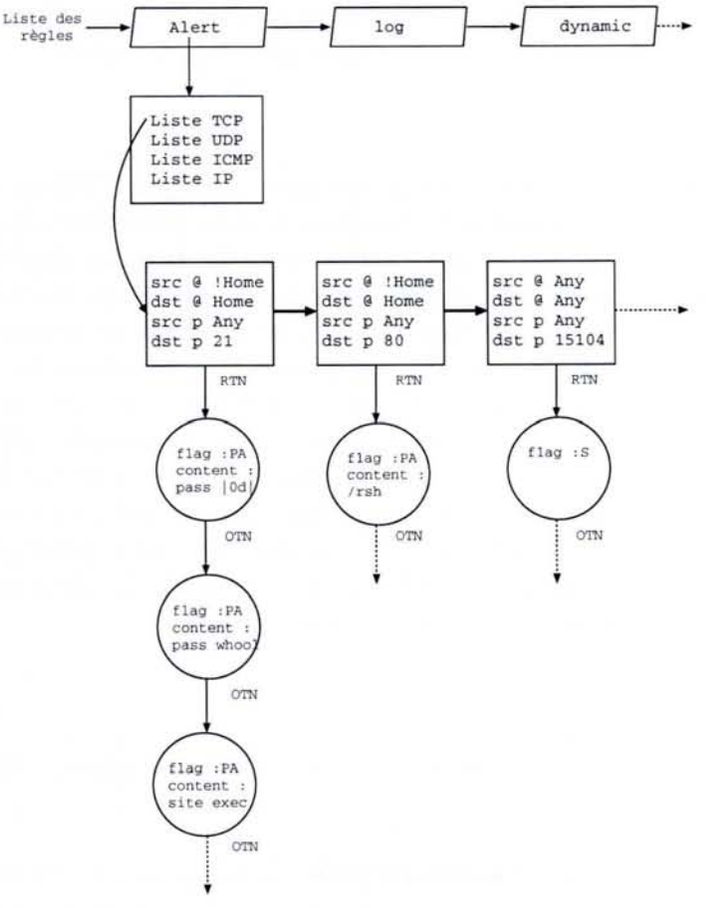


FIG. 4.3 – Chaîne de détection de Snort 1.9

Après avoir construit la chaîne de détection et initialisé les interfaces réseaux, les modules de prétraitement, de détection et de sorties, Snort 1.9 commence son écoute sur les interfaces réseaux. Il utilise pour cela la bibliothèque LIBPCAP et envoie chaque paquet intercepté au décodeur puis à l'ensemble des préprocesseurs. Après la normalisation du contenu, le paquet est envoyé au moteur de détection pour traverser la chaîne de détection. Le parcours des RTNs s'effectue d'une manière séquentielle. Si une RTN est vérifiée alors on vérifie la chaîne des OTNs associée sinon on consulte le prochain RTN. Cette méthode de parcours itératif n'est pas efficace vue la spécificité de quelques RTNs. Par exemple si un RTN avec un port source égale à 80 est vérifiée alors il est inutile de vérifier d'autres RTNs avec des valeurs distinctes de port source. Pour remédier à ce problème, des récents travaux optimisent le mécanisme de détection évoqué ci dessus. Nous présentons ces approches dans la Section 4.3.

4.3 Organisations des règles de détection d'attaques

Snort 1.9 et toutes les versions antérieures traversent séquentiellement la chaîne de détection pour détecter les intrusions. Le temps d'analyse d'un seul paquet est alors important. La méthode supporte un débit moyen de trafic mais devient inefficace lors du passage au haut débit. Cela est dû au court délai séparant la réception des différents paquets à analyser. Lors du surcharge, un système de détection d'intrusions rejette les paquets sans mener l'analyse nécessaire. La situation se complique d'avantage avec l'augmentation du nombre de règles de détection d'attaques ce qui implique la création de longues listes de RTNs et d'OTNs. La lenteur d'une application itérative des règles de détection d'attaques ont incité les développeurs de Snort d'une part et Kruegel et Toth [82] d'autre part à proposer des solutions alternatives de détection.

4.3.1 Snort 2.0

La nouvelle version de Snort définit un gestionnaire de règles [125] qui divise durant l'initialisation de l'IDS l'ensemble de règles de détection en sous ensembles plus fins. L'analyse des paquets s'effectue en sélectionnant d'abord un sous ensemble de règles candidates puis en appliquant les signatures sur le contenu du paquet. On évite ainsi le parcours total et séquentiel de la liste des RTNs. La division en ensemble de règles est fonction du protocole de transport ou réseau. En effet, Snort 2.0 utilise les valeurs des ports source et destination pour regrouper les règles TCP et UDP. Par contre il emploie le type du message pour rassembler les règles ICMP et l'identifiant du protocole de transport pour réunir les règles de détection IP. Les valeurs de ces paramètres servent à construire les sous ensembles de règles. Une valeur peut être unique c'est-à-dire définissant une valeur explicite (un entier pour un port) ou être générique dans le cas contraire (un intervalle ou Any). Snort 2.0 considère les règles qui comportent au moins un paramètre unique des *règles uniques*, et qualifie l'autre catégorie de règles de *règles génériques*. Ensuite chacune des 2 catégories est divisée en 3 sous classes :

- règles sans filtrage du contenu,
- règles avec filtrage général du contenu,
- règles avec filtrage précis du contenu comme par exemple de l'adresse URL pour une requête HTTP.

Nous discutons dans la suite les règles détection sur les protocoles TCP et UDP. Les deux paramètres choisis pour former les ensembles de règles sont le port source et le port destination. Durant la phase d'initialisation de l'IDS, Snort 2.0 stocke dans deux grandes tables les règles

uniques par rapport à chacun de ces deux paramètres (Figure 4.4). Il définit en plus d'autres structures pour sauvegarder les règles génériques. Ensuite, et pendant la phase de détection, le gestionnaire de règles extrait en utilisant les valeurs des ports source et destination du paquet reçu, deux sous ensembles de règles candidates. Trois cas d'analyse sont possibles :

1. Les deux ensembles de règles uniques (par rapport au port source et au port destination) sont non vides. Dans ce cas un conflit de détection est repéré et il est ignoré s'il est explicitement défini par l'administrateur (par exemple une communication entre deux serveurs DNS sur les ports 53). Cependant, un conflit non défini déclenche automatiquement une alarme (par exemple une connexion entre les deux port 20 et 80). A ce stade, l'analyse du paquet est interrompue.
2. Un seul ensemble de règles uniques est non vide. L'analyse du paquet se poursuit normalement en appliquant les règles de cet ensemble. Snort 2.0 traite séparément les trois catégories de règles : les règles avec filtrage général, les règles avec filtrage précis et les règles sans filtrage du contenu.
3. Les deux ensembles de règles uniques sont vides. L'IDS applique dans ce cas les règles génériques. Il traite également les trois groupes de règles avec filtrage général, avec filtrage précis et sans filtrage du contenu.

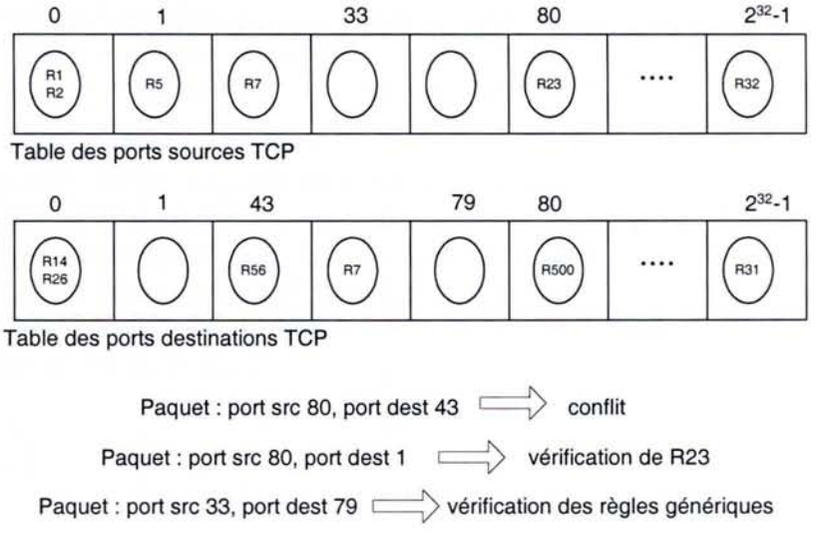


FIG. 4.4 – Organisation des règles : Snort 2.0

Discussion

Le gestionnaire de règles associé à d'autres optimisations qui portent sur l'analyse du flux protocolaire (distinction entre clients et serveurs) et le filtrage multiple des motifs d'attaques améliorent les performances de l'IDS [154]. En outre, la nouvelle organisation des règles regroupe en deux ensembles distincts les règles avec filtrage général du contenu et les règles avec filtrage précis d'un champs du paquet. Cette opération permet d'assurer une recherche simultanée de plusieurs motifs d'attaques via des algorithmes de filtrage convenables que nous étudierons en détail au Chapitre 5. Néanmoins la méthode souffre de quelques lacunes que nous résumons ainsi :

1. Toute valeur non explicite est considérée générique. Par suite les règles qui définissent des intervalles de ports ou qui excluent une valeur (de la forme ! *valeur*) sont classées génériques. Ceci met en cause la division des règles si la majorité d'entre elles sont définies génériques et donc stockées dans le même groupe,

2. La méthode exige l'allocation d'un grand espace mémoire puisqu'elle tient compte de tous les ports sources et destinations. Notons qu'il existe 2^{16} valeurs possibles pour chaque type de ports ce qui représente la taille des 2 tables réservées pour contenir les règles uniques,

3. Si les conflits définis sont considérés normaux, il n'en découle pas le cas pour les conflits indéfinis. Arrêter l'analyse du paquet lors de ces situations en générant directement un événement suspect mène vers un faux négatif si un événement plus sévère est décrit par une des règles en conflit. De plus on engendre un faux positif si une connexion légitime non déclarée par l'administrateur, utilise deux ports uniques source et destination. Par ailleurs, un mauvais ajout des règles augmente le risque des conflits indéfinis ce qui dégrade la qualité de détection des attaques. Afin de remédier à ces problèmes, Snort 2.0 laisse à l'utilisateur la possibilité de choisir sa propre stratégie d'analyse. Un administrateur peut configurer son système de détection d'intrusions pour réaliser la double inspection des deux ensembles de règles uniques en conflit indéfini. D'autres stratégies consistent à s'arrêter dès la vérification d'une règle appartenant à l'un des deux ensembles en conflit ou à choisir selon une certaine probabilité l'un des deux ensembles pour poursuivre l'analyse du paquet. Ces différentes stratégies permettent d'ajuster le niveau de sécurité souhaité. Cependant elles ne résolvent pas le problème de conflit qui provient de la méthode de détection elle-même,

4. La méthode regroupe les motifs apparus dans les règles à filtrage général et précis ce qui offre l'avantage de mener un filtrage simultané de plusieurs motifs d'attaques. La collecte des motifs dépend uniquement des ports source et destination définis dans les règles de détection d'attaques. La méthode ne garantie pas un regroupement des motifs similaires. De plus, le processus de vérification entame aussitôt l'opération de filtrage dés la sélection d'un sous ensemble de règles candidates. Cependant, l'opération de filtrage est connue être lourde et peut être retardée ou même ignorée si on vérifie d'abord les options communes des règles du même groupe.

4.3.2 Snort N.G.

Kruegel et Toth [82] soulignent l'importance d'une application rapide des règles pour réduire le taux de paquets rejetés. Ils confirment alors les résultats publiés en [84]. Ils proposent l'utilisation des arbres de décision pour classer les règles de détection d'attaques. Le processus de classification s'appuie sur les valeurs des options définies par les règles. Par exemple on choisit dans la Figure 4.5 le port destination pour former des groupes de règles. Seulement une option peut ne pas figurer dans la partie corps d'une ou plusieurs règles. Dans ce cas, le champs peut prendre n'importe quelle valeur et la règle correspondante doit être reproduite dans tous les sous groupes lors de la division par rapport à cette option. Afin de résoudre ce problème, les auteurs sélectionnent un ensemble de paramètres significatifs. Ensuite, ils comparent à un seuil le facteur d'apparition des options significatives par rapport au nombre total de paramètres. Ils forment ainsi deux sous groupes de règles candidates. Enfin ils construisent pour chaque groupe un arbre de décision. Ils utilisent le gain d'information (que nous l'utilisons aussi au Chapitre 6) pour sélectionner la meilleure suite d'options qui discriminent les règles de détection d'attaques.

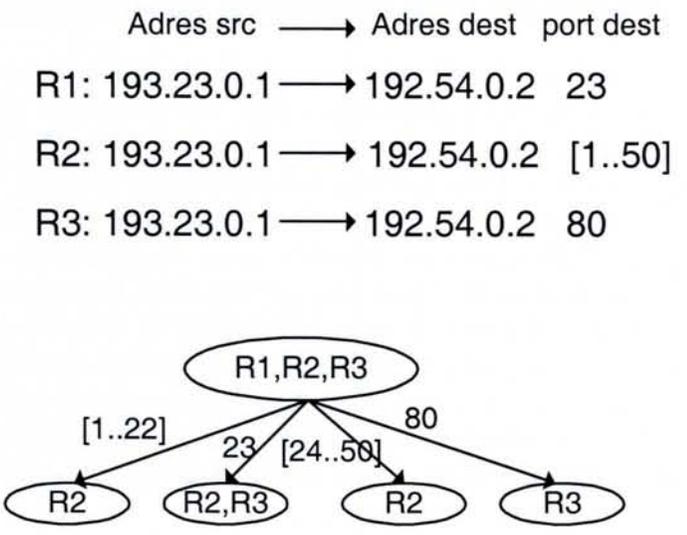


FIG. 4.5 – Organisation des règles : Snort N.G

Discussion

La méthode de classification est intéressante puisqu'elle permet de choisir intelligemment l'ordre des paramètres pour regrouper efficacement les règles. Elle présente des résultats expérimentaux encourageants en présence de plusieurs règles de détection d'attaques [83]. Néanmoins elle présente quelques inconvénients :

- la classification divise successivement l'ensemble des règles en des petits sous groupes créant ainsi un arbre. Au cours de cette division, chaque groupe possède plusieurs fils possibles dont le nombre augmente en fonction des chevauchements entre les règles. Un nombre important de nœuds fils alourdit la phase d'analyse des paquets puisqu'il est indispensable de localiser rapidement et à chaque niveau de l'arbre le sous groupe suivant. Les auteurs proposent des recherches sous linéaires lorsque les paramètres sont de type entier. Par contre ils effectuent une recherche binaire pour analyser le contenu des autres champs comme les adresses IP et les flags TCP. La complexité en temps est égale à la longueur en bit des champs inspectés. Notons qu'une vérification multi-bits auraient accéléré cette recherche.
- En divisant les intervalles qui interfèrent, une même règle peut appartenir à plusieurs sous groupes du même niveau. La réplication des règles augmente la taille des sous groupes ce qui élève la consommation mémoire d'une part, et alourdit la recherche des nœuds fils d'autre part.

Notons que l'option de filtrage "content" ne constitue pas un paramètre pour rassembler les règles. En effet, plusieurs motifs peuvent apparaître simultanément dans le contenu d'un même paquet. Ainsi avec N motifs, le nombre théorique des sous groupes est égale à 2^N ce qui est infaisable pratiquement. Cette restriction ne gène pas la classification des règles puisque l'opération de filtrage doit être retardée au maximum à cause de sa lenteur. De plus, en effectuant d'abord les tests légers définis par les autres options des règles, on ignore en cas de leurs échecs les opérations de filtrage.

Nous proposons dans la Section 4.4 deux nouveaux schémas d'organisation des règles de détection d'attaques. Nous comparons nos techniques aux méthodes proposées dans Snort 2.0 et Snort N.G.

4.4 Représentation des entêtes des règles de détection d'attaques

Nous présentons dans cette section notre méthodologie pour optimiser la représentation des règles de détection d'attaques dans la chaîne de détection. Les techniques que nous décrirons s'appliquent sur une dimension quelconque des règles et ont pour objectif d'éviter le parcours séquentiel d'une chaîne de détection. Nous utilisons la terminologie de Snort pour désigner les structures de données contenant les différentes parties d'une règle. Ainsi un RTN contient la partie entête d'une règle alors qu'un OTN représente la partie corps. Nous divisons ainsi le problème de la construction de la chaîne de détection en 2 sous-problèmes : l'organisation adéquate de la liste des RTNs et la représentation convenable de la liste des OTNs attachée à chaque RTN.

Notre première tâche consiste à modifier la représentation de la liste des RTNs. Chaque RTN est défini via quatre paramètres à savoir les ports source et destination et les adresses source et destination. Nous sommes face à un problème de classification de règles à 4 dimensions que nous résolvons via deux méthodes : l'utilisation du graphe de classification et l'organisation des RTNs.

4.4.1 Graphe de classification des RTNs

La première solution consiste à appliquer l'algorithme de classification introduit au Chapitre 3, pour regrouper les règles de détection d'attaques. Nous conservons la même matrice de classification et nous modifions le graphe de classification en marquant autrement les nœuds finaux (Figure 4.6). Ces nœuds affichent en fait toutes les règles candidates, c'est à dire les règles primaires et secondaires, et non seulement la règle la plus prioritaire comme nous l'avons fait pour traiter des règles de division de trafic. En effet, pour diviser le trafic, il suffisait de connaître la règle la plus prioritaire pour choisir un IDS. Par contre, la détection des attaques au sein d'un IDS exige tenir compte de toutes les règles candidates. Une règle candidate commence par un RTN que nous vérifions les paramètres en parcourant le graphe de classification. L'analyse du paquet se poursuit ensuite en inspectant les chaînes OTN associées aux RTNs sélectionnés d'un nœud final du graphe.

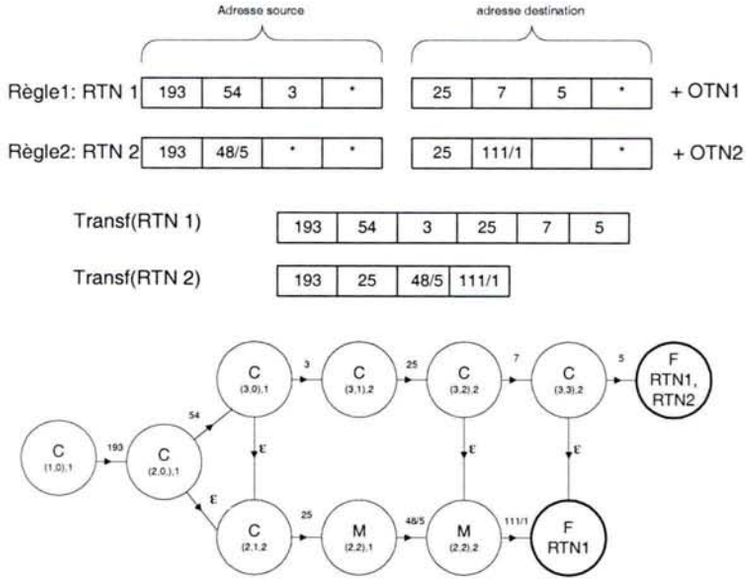


FIG. 4.6 – Graphe de classification des RTNs

Comparaison avec Snort N.G

En comparant notre graphe de classification à l'arbre de décision présenté dans [83], nous remarquons que notre méthode se restreint aux quatre premiers paramètres de la structure RTN. Par ailleurs les deux méthodes ont la même complexité en temps pour retrouver l'intervalle de ports adéquat mais en effectuant des opérations distinctes. Notre travail consiste à retrouver les coordonnées sur la matrice de classification alors qu'en [83], on cherche un nœud fils dans l'arbre de décision. De plus, nous traitons les adresses IP comme des suites d'octets ce qui nécessite 4 opérations de recherche pour des adresses IPv4. Cette stratégie est plus rapide qu'une recherche binaire qui nécessite 32 opérations pour le même type d'adresse. En outre, si les chevauchements de ports ont été considérés dans [83], il n'était pas le cas pour les adresses IP source et destination. De notre coté, nous avons étudié ce problème, distingué entre octets complets et octets masqués, puis construit les pointeurs d'échecs entre les nœuds du graphe.

Comparaison avec Snort 2.0

La stratégie de vérification employée dans Snort 2.0 regroupe les règles de détection d'attaques dans plusieurs sous-ensembles. Cependant, elle ne considère que les ports à valeurs uniques. Ainsi les intervalles de ports sont considérés comme des valeurs génériques. De plus la méthode souffre des conflits indéfinis entre les sous-ensembles de règles uniques et consomme un espace mémoire énorme en prévoyant toutes les valeurs possibles des ports source et destination (2^{16} valeurs possibles pour chaque type de port). Enfin, Snort 2.0 n'utilise pas les adresses IP pour diviser les règles de détection ce qui engendrera des tests inutiles si ces adresses sont communes entre les règles d'un seul groupe.

Nous présentons dans la sous-section 4.5.1 les résultats expérimentaux obtenus en regroupant les règles de détection d'attaques via les graphes de classification. Nous comparons cette méthode à une autre technique d'organisation de règles que nous introduisons dans la sous-section 4.4.2. Le but de la nouvelle technique est de réduire la consommation mémoire constatée en appliquant les graphes de classification.

4.4.2 Organisation des RTNs

La classification des règles présentée dans la sous-section 4.3.1 construit une matrice en tenant compte des ports sources et destinations. La taille de la matrice dépend essentiellement des chevauchements entre les intervalles de ports définis dans la partie entête des règles de détection d'attaques. Ainsi en augmentant le nombre de règles et en variant les types d'attaques à détecter, on obtient une large matrice de classification. Par ailleurs, les transitions entre les nœuds du graphe sont étiquetées par des valeurs d'octets. On définit alors 256 liens distincts pour un chaque nœud. Malgré la rapidité de la méthode, nous étudions dans la suite la possibilité de la réduction de l'espace mémoire consommé.

Nous proposons dans cette sous-section de **garder la représentation liste** des règles de détection d'attaques mais en organisant différemment ses entrées. Nous nous appuyons sur deux principes de recherche qui aident la réorganisation de la liste des RTNs :

1. mémorisation des succès de filtrage : par exemple si un RTN avec un port source égale à 80 est vérifié, alors uniquement les RTNs ayant la même valeur de port sont consultés,
2. mémorisation des échec de filtrage : par exemple si un RTN avec un port destination égale à 80 n'est pas vérifié alors il est inutile de considérer les autres RTNs ayant la même valeur de port.

4.4.2.1 Regroupement des RTNs : classe générique et classe explicite

Une structure RTN comporte 4 champs, les adresses source et destination et les ports source et destination. Chaque champs peut avoir une valeur *explicite* c'est-à-dire une valeur unique, un intervalle ou un ensemble de valeurs. Le champs peut avoir également une valeur *générique* représentée par *. Dans une première étape, nous regroupons les règles ayant les mêmes champs explicites. Par exemple, nous rassemblons les règles qui attribuent des valeurs explicites uniquement au paramètre "port source" ($Champs_1$ de la Figure 4.7). Nous qualifions ce groupe de *groupe générique* puisque à part un seul champs, tous les autres possèdent des valeurs génériques. Par contre, les groupes qui définissent plusieurs entrées explicites sont appelés des *groupes explicites.* Enfin les règles qui ne définissent aucun paramètre sont stockées dans un groupe spécial. Ces différents groupes nous servent de base pour construire la nouvelle liste de règles de détection d'attaques. Nous insérons d'abord les groupes génériques puis nous les suivrons par les groupes explicites avant d'intégrer le groupe spécial. Cette organisation permet d'appliquer facilement les deux principes de recherche mentionnés ci-dessus en indexant les groupes explicites par les entrées des groupes génériques selon le schéma de la règle 4.1. Nous détaillons cette construction dans les sous-sections 4.4.2.2 et 4.4.2.3.

$$(Port\ source = 80, port\ destination = *) \rightarrow (Port\ source = 80,\ port\ destination = 12) \textbf{(4.1)}$$

4.4.2.2 Traitement des groupes génériques

Un groupe générique contient des règles avec un seul paramètre ayant une valeur explicite. Nous représentons les valeurs de ce paramètre sous la forme d'intervalles d'entiers et ceci pour tous les types de paramètres c'est-à-dire les ports et les adresses IP. En effet, les valeurs uniques d'un port ainsi que les ensembles de ports peuvent être représentés via des intervalles d'entiers. Par ailleurs, une adresse IP sur 32 bits est équivalente à un entier alors que le préfixe d'une adresse IP (c'est-à-dire une adresse réseau) peut être vu comme un intervalle d'entiers. Par la suite, nous considérons que les valeurs explicites d'un paramètre sont des intervalles d'entiers. Afin de chercher rapidement les RTNs au sein d'un groupe générique tout en tenant compte des chevauchements des intervalles, nous divisons les entrées en intervalles élémentaires. Ensuite nous trions ces intervalles pour pouvoir mener une recherche dichotomique rapide.

Nous montrons dans la Figure 4.7 un schéma d'organisation de règles ayant seulement trois dimensions. Nous créons trois groupes génériques $Champs_1$, $Champs_2$ et $Champs_3$. Les entrées du $Champs_1$ sont triées et pointent vers des RTNs bien déterminés. Elles comportent également des informations sur les groupes explicites.

4.4.2.3 Traitement des groupes explicites

Un groupe explicite contient les règles qui ont plusieurs paramètre à valeurs explicites. Nous divisons chaque groupe explicite en deux ensembles. Un ensemble des règles dont les valeurs des champs sont apparues dans les groupes génériques et un ensemble qui comporte le reste des entrées (Figure 4.7). Cette répartition permet d'une part l'indexation des entrées du premier ensemble par les règles des groupes génériques. On applique ainsi le premier principe de recherche (succès de filtrage). D'autre part, et en cas d'échec de filtrage dans les groupes génériques, nous poursuivrons la recherche dans la deuxième partie des groupes explicites. On utilise alors le deuxième principe de recherche (échec de filtrage).

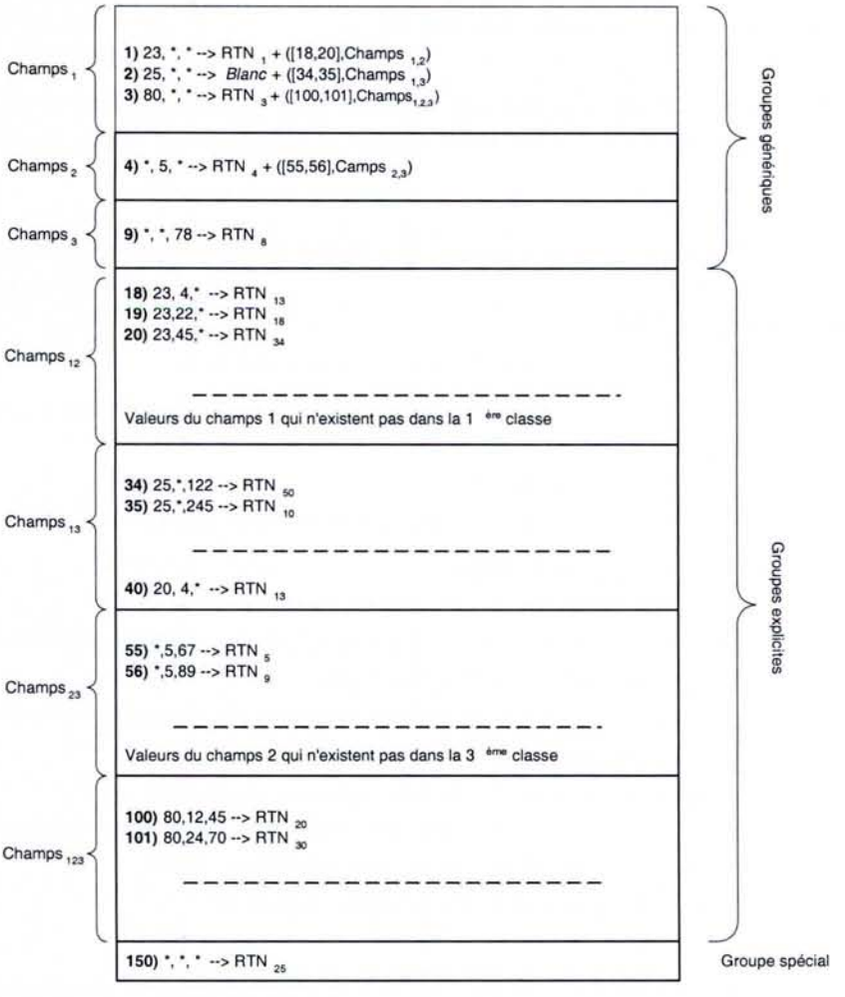


FIG. 4.7 – Recherche des règles de détection

Par ailleurs, nous introduisons une optimisation pratique lors de l'organisation des RTNs. Nous ajoutons des règles génériques dont les valeurs du champs explicite sont fréquemment rencontrées dans les groupes explicites. Nous appelons ces règles, des *règles blanches*, puisqu'elles ne détectent aucune attaque mais servent uniquement de pointeurs directs vers des règles de groupes explicites.

La Figure 4.7 représente une liste triée de règles de détection d'attaques. Nous considérons un cas de recherche avec trois dimensions. Il s'agit d'une simplification par rapport à Snort où un RTN possède 4 champs différents. Toutefois, la méthode est implantée pour un nombre quelconque de champs et donc elle supporte des règles de dimension variable. Nous créons sur cette figure trois groupes génériques, $Champs_1$, $Champs_2$ et $Champs_3$. Par ailleurs, nous avons quatre groupes explicites. $Champs_{12}$, $Champs_{13}$ et $Champs_{123}$ sont indexés par le groupe générique $Champs_1$ et $Champs_{23}$ est indexé par le groupe générique $Champs_2$. Nous insérons en queue de liste le groupe spécial des règles ne contenant que des paramètres génériques. Chaque entrée d'un groupe générique pointe vers des règles appartenant à des groupes explicites. Par exemple, l'entrée numéro 1 pointe vers les entrées [18..20] puisque toutes ces règles ont la même valeur 23 du premier paramètre. En outre, nous ajoutons la règle blanche numéro 2 afin de retrouver rapidement les règles 34 et 35.

4.4.2.4 Algorithmes d'organisation des RTNs

Nous présentons dans cette partie deux algorithmes pour la construction et le parcours de la liste des RTNs. Nous commençons par l'algorithme d'organisation de la chaîne des RTNs (Algorithme 4).

début
  pour chaque *règle r* **faire**
    Classer r par rapport à l'ensemble des champs explicites
  Extraire un ensemble $\mathcal{V}$ des valeurs fréquentes des champs des groupes explicites non apparues dans les groupes génériques.
  pour chaque *valeur v dans* $\mathcal{V}$ **faire**
    Ajouter une règles blanche dans le groupe générique associé
  pour chaque *groupe explicite* $\mathcal{G}$ **faire**
    Diviser $\mathcal{G}$ en 2 sous groupes :
      1- sous groupe 1 dont les valeurs du premier champs explicite existent dans les groupes génériques
      2- sous groupe 2 dont les valeurs du premier champs explicite n'existent pas dans les groupes génériques
  Ordonner les groupes génériques et les sous groupes explicites
  Concaténer tous les groupes dans la liste des règles puis énumérer les règles
  pour chaque *valeur explicite dans un groupe générique* **faire**
    pointer vers les règles possibles des groupes explicites
fin

Algorithme 4: Organisation des règles de détection

Le parcours de la liste ordonnée des règles s'effectue par rapport aux groupes génériques. En effet, nous disposons initialement d'une liste vide *l_candidat*. Nous utilisons ensuite le contenu du paquet pour tester les entrées des groupes génériques. Nous insérons dans la liste *l_candidat* toutes les RTNs des règles qui s'appliquent ainsi que la liste des pointeurs associées. Cependant, si aucune règle n'est vérifiée dans un groupe générique alors on poursuit la recherche dans la deuxième partie des groupes explicites. On résume cette stratégie de parcours dans Algorithme 5.

L'organisation des règles résout le problème du parcours séquentiel de la liste des RTNs. En effet, elle permet via un arrangement des RTNs de réduire le nombre de tests à effectuer sur les entêtes des règles de détection d'attaques. De plus, elle minimise l'utilisation mémoire puisque nous nous basons sur une liste chaînée des RTNs que nous ajoutons ensuite des informations pour orienter la recherche vers les groupes explicites. Notons que la recherche dichotomique assurée au sein de chaque groupe générique s'effectue uniquement par rapport aux valeurs explicites de ce groupe. L'ensemble de ces valeurs est minime vis à vis de toutes les valeurs possibles quand on réunit toutes les règles explicites et génériques. Cette optimisation accélère la recherche dichotomique et différencie notre méthode de l'arbre de recherche N-aire défini par Kruegel et Toth [83]. En effet, leur méthode tient compte des valeurs de toutes les règles candidates, ce qui génère de nombreux nœuds fils alourdissant par suite l'analyse des paquets reçus.

```
début
    pour chaque paquet reçu faire
        l_candidat = ∅
        pour chaque classe générique faire
            Trouver la règle générique r qui s'applique sur le paquet
            si r existe alors
                l_candidat = l_candidat : :r
                pour chaque règle r_i pointé par r faire
                    si le paquet vérifie r_i alors
                        l_candidat = l_candidat : :r_i
            sinon
                Chercher dans la deuxième la deuxième partie des groupes explicites
                l'ensemble des règles possibles R_j
                l_candidat = l_candidat : :R_j
        Verifier le paquet par rapport à l_candidat
fin
```

Algorithme 5: Parcours de la liste triée des règles de détection d'attaque

4.5 Factorisation des corps des règles de détection d'attaques

Nous avons présenté dans la Section 4.3 l'organisation des entêtes des règles de détection d'attaques ce qui correspond dans Snort à la liste des RTNs. Nous discutons dans cette section la vérification de la partie corps des règles. Snort stocke les informations dans des structures de données internes appelées OTN (Option Tree Node). Une structure OTN définit un ensemble de tests appelés options. La vérification de la totalité de ces options permet de détecter une attaque. L'IDS émet dans ce cas une alerte pour avertir l'administrateur. Parmi les options intéressantes, nous soulignons l'opération de filtrage du contenu. Chaque règle contient un ou plusieurs motifs d'attaques qui possèdent le plus souvent des parties communes (Figure 4.8). Étant donnée la multitude des opérations de filtrage et leurs exigences en terme de temps de traitement, divers travaux optimise ce processus. Le filtrage simultané de plusieurs motifs constitue en fait une solution intéressante pour accélérer la détection des intrusions mais suppose bénéficier d'une stratégie efficace pour rassembler les motifs. Notons qu'à ce stade, Snort 1.9 regroupe les motifs des OTNs dans les structures RTN. Néanmoins la méthode ne permet pas d'obtenir un ensemble de motifs similaires.

Par ailleurs, quelques options sont communes à plusieurs structures OTNs. Par exemple, le protocole TCP définit les champs drapeau (FLAG), numéro de séquence (SEQ) et numéro d'acquittement (ACK). Ensuite, le protocole IP présente les options durée de vie (TTL), identifient du paquet (ID) et type de service (TOS). Les options communes engendrent des tests redondants sur la même structure Packet. Ainsi, il est préférable d'éviter ces tests en vérifiant une seule fois une option commune.

Pour ce faire, nous sélectionnons quelques options que nous nous basons dessus pour factoriser les OTNs. Nous définissons deux types d'OTN : les OTNs primaires et les OTNs secondaires (Figure 4.9). Les OTNs primaires rassemblent les parties communes de plusieurs règles alors que les OTNs secondaires comportent les options spécifiques à chaque règle. Dans cette nouvelle

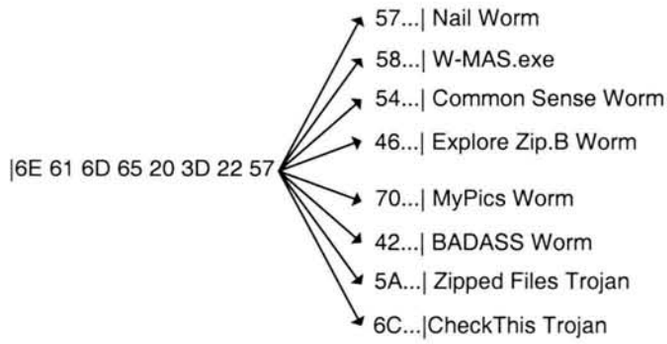


FIG. 4.8 – Recherche des motifs d'attaques

architecture, nous appelons OTN ordinaire, chaque OTN n'ayant pas subie de factorisation. Cet OTN correspond parfaitement à l'ancien type d'OTN de Snort. De plus, nous stockons les motifs des règles, non plus dans les structures RTNs, mais dans des OTNs primaires et ordinaires.

Prenons par exemple les 4 règles suivantes :

R1 : Alert TCP @src prt_src → @dst prt_dst (msg : "α", flags :A+, ttl :1, sid :x_1)
R2 : Alert TCP @src prt_src → @dst prt_dst (msg : "β", flags :A+, ttl :2, sid :x_2)
R3 : Alert TCP @src prt_src → @dst prt_dst (msg : "δ", flags :A+, ttl :3, sid :x_3)
R4 : Alert TCP @src prt_src → @dst prt_dst (msg : "θ", seq :0, sid :x_4)

Les quatre règles ont le même type "Alert" et traitent le même protocole "TCP". Elles possèdent aussi les mêmes adresses source et destination, et les mêmes ports source et destination. Elles appartiennent par conséquent à la même RTN. L'ancienne chaîne de détection est schématisée dans la Figure 4.9. Ensuite en optant pour une factorisation d'états, nous obtenons une nouvelle physionomie de la chaîne des OTNs. Dans ce nouveau schéma, le nombre de tests de l'option "flags" est réduit de 3 à 1. Ce gain est plus important avec un nombre supérieur de règles de détection d'attaques. Par ailleurs la factorisation nous offre l'opportunité de privilégier l'analyse de certaines règles d'attaques. Nous discutons cet avantage dans la sous-section 4.4.1.

4.5.1 Précédence entre les règles

Afin de factoriser les structures OTNs, nous sélectionnons quelques options et nous y attribuons des poids différents. Le poids total d'un OTN est la somme des poids de ses options. Puisque chaque OTN représente une règle de détection d'attaque, nous introduisons via ce procédé des précédences entre les règles.

En effet, soit $D = \{x_1, x_2, ..., x_n\}$ l'ensemble des options possibles. Nous voulons regrouper les règles selon le sous ensemble des options $Y = \{y_1, y_2, ..., y_m\}$ inclus dans D. Nous obtenons 2^m classes différentes.

Exemple 4.5.1 *soit $Y = \{y_1, y_2\}$ l'ensemble des options communes. On dispose alors de 4 classes distinctes :* $[]$, $[y_1]$, $[y_2]$, $[y_1, y_2]$.

Nous construisons un vecteur p de taille n dont chaque élément p_i représente soit un poids non nul d'une option y_i de Y, soit 0 pour une autre option x_i n'appartenant pas à Y. Le poids de y_i dépend de la fréquence d'apparition de l'option dans les attaques détectées sur le réseau. Ensuite nous calculons le poids de chaque règle r_j en sommant les poids élémentaires de chacune de ses

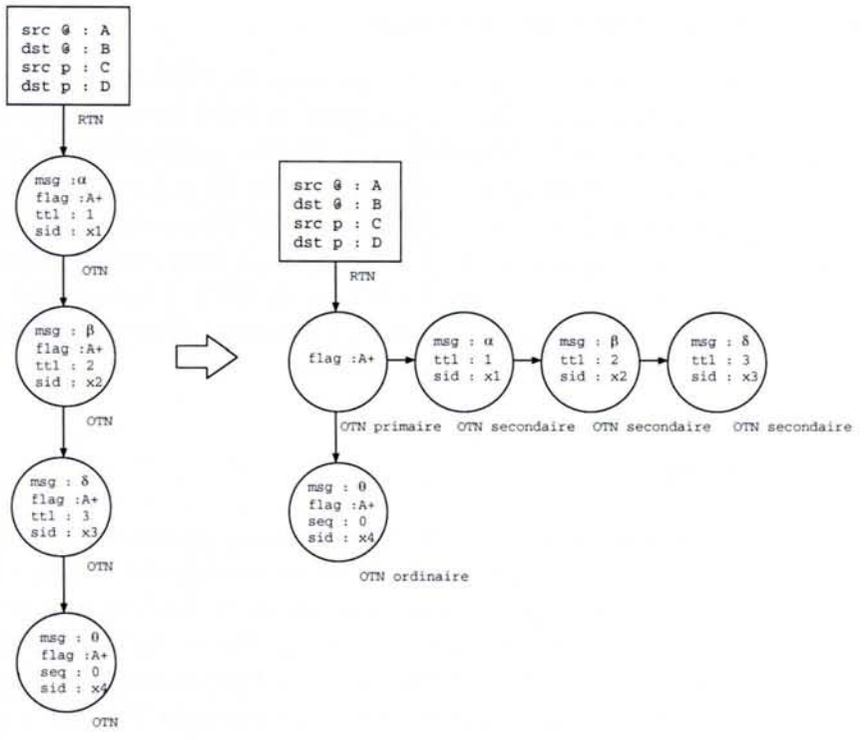


FIG. 4.9 – Factorisation de la chaîne des OTNs

options. Pour ce faire, nous projetons chaque règle r_j sur la base $x = (x_1, x_2, ..., x_n)$ des options possibles des OTNs. On obtient un vecteur $e_j = (e_{1,j}, e_{2,j}, ..., e_{n,j})$ tel que $e_{i,j} = 1$ si l'option x_i existe dans la règle r_j. Le poids d'une règle r_j est le produit scalaire entre les deux vecteurs e_j et p. Nous appelons F la fonction qui calcule le poids des règles de détection d'attaques.

Les règles possédant les mêmes options y_i *de* Y forment une classe a part, notée [ ]. Il est clair que les éléments d'une même classe présentent le même poids c'est à dire $F(r_j) = F([r_j])$. En outre, il est nécessaire d'assurer l'injectivité de F c'est-à-dire qu'à chaque poids correspond une seule classe possible de règles. Ceci nous permet de classer les règles selon le critère poids. Afin d'assurer l'injectivité de F il suffit de choisir convenablement le vecteur poids p. Nous attribuons un poids égale à 2^i à la i^{me} option de Y et un poids nul pour les autres options appartenant à l'ensemble D\Y. Par ailleurs nous ordonnons les éléments de Y selon un ordre croissant d'importance ce qui permet d'affecter un poids plus élevé à l'option la importante.

Exemple 4.5.2 *Soient* $D = \{x_1, y_1, y_2, x_4\}, Y = \{y_1, y_2\}, z \in \{0, 1\}$.
$p(y_1) = 2^0 = 1$, $p(y_2) = 2^1 = 2$, $p(x_1) = p(x_2) = 0$.

$$F([]) = (z,0,0,z).\begin{pmatrix}0\\1\\2\\0\end{pmatrix} = 0 \qquad F([y_1]) = (z,1,0,z).\begin{pmatrix}0\\1\\2\\0\end{pmatrix} = 1$$

$$F([y_2]) = (z,0,1,z).\begin{pmatrix}0\\1\\2\\0\end{pmatrix} = 2 \quad F([y_1,y_2]) = (z,1,1,z).\begin{pmatrix}0\\1\\2\\0\end{pmatrix} = 3;$$

4.5.2 Adaptation de la chaîne de détection

La nouvelle représentation des règles s'adapte dynamiquement à l'environnement d'exécution de l'IDS. En effet, une analyse des alarmes déclenchées met en relief les signatures fréquemment rencontrées. Il suffit d'ajouter des règles blanches pour repérer rapidement les entêtes de ces signatures. Par ailleurs, en comptant les options de ces règles, nous pouvons attribuer des poids plus importants aux paramètres usuels. Nous privilégions ainsi l'inspection des règles dont la partie Corps contient ces options. Cette opération se traduit sur Snort par la remonté automatique des structures OTNs dans la liste des OTNs issue d'un RTN. L'adaptation de la chaîne de détection requiert néanmoins une analyse continue des intrusions détectées. Nous envisageons étudier et automatiser cette opération dans un futur travail.

4.6 Implémentation

Nous présentons dans cette section les résultats expérimentaux obtenus en modifiant la chaîne de détection de Snort. Nous traitons d'abord la classification des règles en utilisant la matrice et les graphes de classification. En suite nous discutons l'organisation des règles de détection d'attaques. Durant ces expériences nous utilisons la version 1.9 de Snort. La base de règles contient 1747 règles possibles distribuées sur 47 classes d'attaques. La majorité des règles s'appliquent sur un trafic HTTP (750 parmi 1606 des règles actives). Il existe également 115 règles pour détecter les portes dérobées et 83 autres pour des signatures de virus et de vers. Après la compilation des règles, Snort 1.9 génère 192 RTNs et 1606 OTNs. Ainsi, en analysant des données non HTTP, la sélection rapide des règles candidates via la matrice de classification ou la liste triée des règles permet d'éviter un nombre important de signatures HTTP. Par ailleurs, en traitant un flux HTTP, les deux méthodes d'application de règles pointent directement vers des règles HTTP. Nous présentons dans la suite les expériences effectuées afin d'évaluer les performances en temps et en mémoire de la version modifiée de Snort.

4.6.1 Classification des règles

La première méthode utilisée pour optimiser la détection d'intrusions avec Snort consiste à classer les règles de détection d'attaques. Pour réaliser cet objectif, nous construisons un graphe de classification similaire à celui de la division du trafic. La nouvelle construction présente néanmoins quelques différences lorsqu'elle est appliquée à des règles de détection d'attaques. En effet, et contrairement aux règles de division du trafic, nous modifions l'algorithme pour ne plus tenir compte des priorité des règles. Nous regroupons ainsi dans les nœuds finaux du graphe toutes les règles candidates qui pointent vers des RTN possibles de Snort. Ayant vérifié ces RTN, le processus d'analyse des paquets se poursuit en examinant les chaînes d'OTNs associées aux RTN sélectionnés. Notons que la modification du graphe pour ignorer les priorités des règles augmente le nombre de nœuds générés et par suite la consommation mémoire du graphe.

Durant la phase d'initialisation de l'IDS, nous parcourons la liste des règles de détection d'attaques pour construire les matrices et les graphes de classification. Nous nous intéressons particulièrement aux règles de type "Alert" qui portent sur des trafics TCP et UDP. Nous obtenons deux matrices de classification pour chaque type de flux. Ensuite, chaque cellule des deux matrices de classification pointe vers un graphe unique. Nous présentons dans le Tableau 4.1 les résultats de cette phase d'initialisation. De plus, étant donné l'augmentation du nombre de règles à cause des nouvelles attaques décelées chaque jour, nous testons cette évolution en variant au cours de l'expérience le nombre de règles de détection d'attaques. Pour ce faire, nous réalisons

quatre configurations possibles de l'IDS. Chaque configuration active un ensemble distinct de règles de détection d'attaques. Nous reportons ensuite la consommation mémoire dans la Figure 4.10. Les quatre configurations se résument ainsi :

- Configuration 1 : elle contient la totalité des règles de détection d'attaques fournies dans la version 1.9 de snort. Elle comporte une variété de signatures pour la détection des mauvais trafic, des attaques sur les services RPC, TELNET, FTP, HTTP et DNS et des déni de services simples et distribués,
- Configuration 2 : nous éliminons deux catégories d'attaques ayant un nombre important de règles, les virus (83 règles) et les portes dérobées (115). Nous réduisons ainsi le nombre de règles de détection d'attaques à 1408.
- Configuration 3 : nous gardons uniquement les règles de détection d'attaques sur le protocole HTTP. Il s'agit de la catégorie de règles la plus riche (750 règles) qui inspecte la majeure partie du trafic transitant sur le réseau de test du LORIA.
- Configuration 4 : nous réduisons le nombre de règles actives en ne gardant que celle qui détectent les attaques web. Nous surveillons le trafic destiné à nos serveurs web pour détecter en particulier des commandes shell insérées dans le contenu des requêtes HTTP.

	Config. 1	Config. 2	Config. 3	Config. 4
Nature des règles	Toutes	Toutes\{Virus, Backdoor}	HTTP	Attaques WEB
Nombre de règles	1606	1408	750	47
Nombre des RTN TCP	142	137	12	1
Nombre des RTN UDP	42	41	1	1
Temps de construction (s)	33,428	30,2	0,5	0,1

TAB. 4.1 – Classification des règles de détection d'attaques

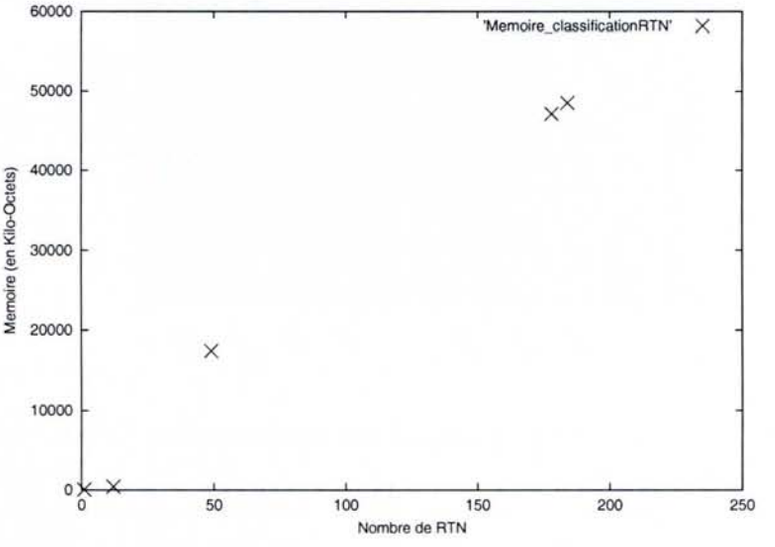


FIG. 4.10 – Consommation mémoire suite à la classification des règles

A l'issue de cette expérience, nous remarquons que le temps nécessaire pour classer les règles de détection d'attaques augmente en fonction du nombre de règles. Ceci ne gène pas la détection des attaques puisqu'il s'agit d'une phase d'initialisation de l'IDS. De plus la consommation

mémoire augmente linéairement en fonction du nombre de règles ce qui diffère des résultats obtenus lors de la division du trafic. Ceci est dû principalement aux modifications apportées au graphe de classification. En effet, nous considérons dans les nœuds intermédiaires et finaux du graphe toutes les règles candidates sans tenir compte de leurs priorités. Nous augmentons ainsi le nombre de liens entre les nœuds ce qui entraîne une utilisation supérieure de l'espace mémoire. Afin de limiter cette consommation, nous expérimentons dans la sous section 4.5.2 la deuxième méthode d'organisation des règles.

Nous considérons dans une deuxième expérience trois fichiers log contenant du trafic Internet collecté sur le réseau du LORIA. Les caractéristiques de ces trafics sont montrées dans le Tableau 4.2 et la Figure 4.11. Ensuite le Tableau 4.3 compare les temps d'exécution de la version Snort 1.9 et de notre version qui classe les règles de détection d'attaques. On remarque une amélioration dans le temps de traitement des paquets. Ce gain est plus important sur log1 qui contient plus de trafic HTTP. La réduction du temps est intéressante dans un contexte de haut débit. En effet, un court délais de traitement par paquet assure une analyse rapide du trafic et évite le rejet des paquets.

	Taille (MB)	TCP	UDP	ICMP	Autres
log 1	222	75,34%	17,34%	6,72%	0,6%
log 2	53	71,92%	12,37%	15,43%	0,28
log 3	32	86,32%	6,02%	7,54%	0,12%

TAB. 4.2 – Caractéristiques des fichiers log

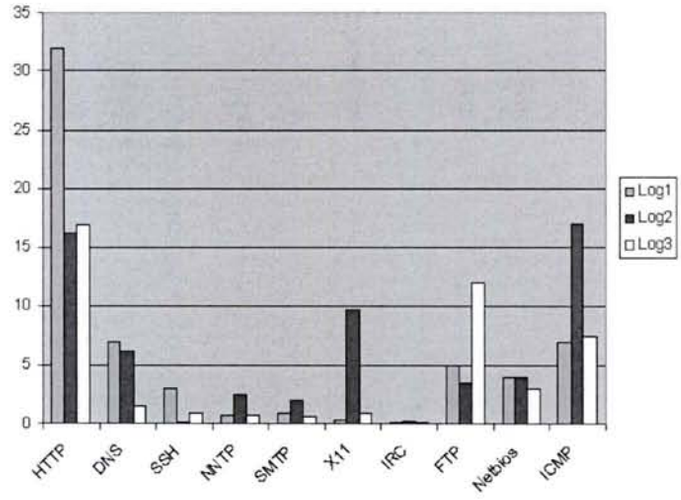


FIG. 4.11 – Composition des fichiers log

	log 1	log 2	log 3
Snort 1.9 (s)	99,29	22,95	9,504
Snort avec classification des règles (s)	13,84	7,65	3,42

TAB. 4.3 – Snort 1.9 Vs. Snort avec classification de règles

4.6.2 Organisation des règles

Nous décrivons dans cette partie l'implémentation et les différentes expériences réalisée sur Snort suite à l'organisation de ses règles de détection d'attaques. Ensuite, nous comparons la version 1.9 de Snort et notre version modifiée de l'IDS.

Durant notre expérimentation, nous traitons les adresses IP comme des entiers sur 32 bits et les préfixes d'adresses comme des intervalles d'entiers. Ainsi chaque règle de détection d'attaques se présente comme une liste d'intervalles. Nous implantons d'abord un module d'organisation de règles de dimension quelconque. Ensuite, nous intégrons le module dans Snort en prenant le cas particulier de 4 paramètres. Nous intéressons en fait aux ports sources et destinations et les adresses sources et destinations. Les deux premiers champs génériques servent à décrire les ports sources et destinations puis nous réservons les champs génériques 3 et 4 aux adresses IP sources et destinations. Notons qu'à ce stade, les groupes génériques ne doivent jamais être vides puisque l'algorithme de parcours des règles se base sur ces ensembles pour retrouver les règles candidates (Algorithme 5). Ainsi, et si à la suite d'une organisation de règles nous manquons un groupe générique, nous insérons une règle aléatoire dans ce groupe. Nous créons ainsi une nouvelle classe qui pointe directement vers les deuxièmes parties des groupes explicites associés.

La première expérience évalue la phase d'initialisation du système de détection d'intrusions. Nous utilisons les quatre configurations de la Section 4.5.1 et nous calculons à chaque fois le nombre de groupes génériques et explicites créées ainsi que le nombre total de pointeurs contenus dans ces classes. Les différents résultats sont portés dans le Tableau 4.4.

	Config. 1	Config. 2	Config. 3	Config. 4
Nature des règles	Toutes	Toutes\{Virus, Backdoor}	HTTP	Attaques WEB
Nombre de règles	1606	1408	750	47
Nombre de groupes génériques des règles TCP	4	4	3	3
Nombre de groupes explicites des règles TCP	7	7	3	1
Nombre de pointeurs vers des RTN TCP	142	138	15	4
Nombre de groupes génériques des règles UDP	3	3	0	0
Nombre de groupes explicites des règles UDP	5	5	0	0
Nombre de pointeurs vers des RTN UDP	44	43	0	0
Temps de construction (s)	0,1	0,1	0,02	0

TAB. 4.4 – Organisation des règles de détection d'attaques

Le Tableau 4.4 montre que l'étape d'initialisation est assez rapide et ne gène pas le déploiement de l'IDS. De plus la quantité de mémoire supplémentaire consommée par rapport à l'ancienne chaîne de détection de Snort 1.9 est assez faible (Figure 4.12). Ce supplément de mémoire est dû à la division des règles qui se chevauchent au sein d'un même groupe générique et aux pointeurs créés depuis les groupes génériques vers les groupes explicites. Par ailleurs, nous réduisons énormément la quantité de mémoire réservée par rapport à la première méthode de classification de règles.

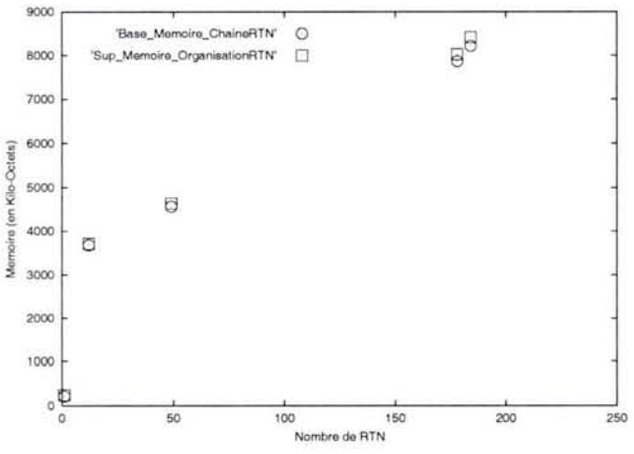


FIG. 4.12 – Consommation mémoire suite à l'organisation de règles

La deuxième expérience évalue la rapidité de l'IDS pour filtrer le trafic réseau. Nous utilisons la première configuration de l'IDS et les 3 fichiers log décrits dans la Section 4.5.1. Nous présentons dans le Tableau 4.5 le temps d'exécution de Snort v1.9 et notre version modifiée d'organisation de règles de détection d'attaques. Nous remarquons une amélioration dans le temps de traitement du trafic. Ce gain est moins important que la première méthode de classification de règles de détection d'attaques. Néanmoins la méthode consomme moins d'espace mémoire. Par ailleurs le gain en temps est plus important sur le log1 qui comporte plus du trafic HTTP. Il s'agit du même constat lorsqu'on a réalisé la classification des règles. Notons que malgré que les règles HTTP sont majoritaires, ils présentent un nombre réduit d'entêtes communes. Par contre les autres catégories de règles possèdent plus d'entêtes séparées. Ainsi, en traitant un trafic web, on sélectionne rapidement les entêtes des règles HTTP ce nous évite de parcourir les entêtes des autres règles. Ceci explique les meilleurs performances sur le fichier journal log1.

	log 1	log 2	log 3
Snort 1.9 (s)	99,29	22,95	9,504
Snort avec classification des règles (s)	29,83	16,93	6,79

TAB. 4.5 – Snort 1.9 Vs. Snort avec organisation des règles

Une comparaison entre Snort 1.9 et Snort 2.0 a été réalisée dans [154] et note que le nouveau mécanisme d'organisation de règles associé à d'autres optimisations qui portent sur l'analyse du flux protocolaire (distinction entre clients et serveurs) et le filtrage multiple de motifs d'attaques contribuent à améliorer les performances de Snort. L'amélioration atteint sur certains données un rapport de 18. Notre travail dans ce Chapitre s'est juste porté sur l'organisation de règles. Par ailleurs, leurs échantillons de tests ainsi que le nombre de règles considérées ne sont pas précisés. Enfin, nous ne connaissons pas la configuration réalisée sur l'IDS. En fait, la vitesse de détection de Snort 2.0 dépend de la manière de considérer les conflits indéfinis. En ignorant ces cas d'analyse, plusieurs règles ne sont pas considérées malgré qu'ils peuvent filtrer des attaques. Nous envisageons dans un futur proche comparer notre version avec Snort 2.0 de point de vue temps d'exécution et consommation mémoire. Nous introduirons d'abord les autres optimisations effectuées par Snort 2.0. Nous utiliserons ensuite les mêmes configurations des IDS et un trafic similaire de trafic.

4.7 Conclusion

Les systèmes de détection d'intrusions emploient les règles de détection d'attaques pour vérifier le contenu du trafic réseau. Un parcours itératif de ces règles alourdit le processus de détection d'intrusions. Afin d'optimiser cette opération, nous avons proposé dans ce chapitre deux méthodes d'application de règles de détection d'attaques. La première méthode procède à une classification des règles en adoptant la technique de division du trafic présentée dans le Chapitre 3. On accélère ainsi l'analyse des paquets ce qui permet de mieux supporter le haut débit. Néanmoins la quantité de mémoire allouée est importante. Afin de résoudre ce problème nous avons proposé une autre technique d'application de règles qui se base sur une organisation adéquate des règles de détection d'attaques. Ce nouveau rangement permet d'inférer à partir des premiers tests portés sur les premiers groupes de règles, la faisabilité des prochaines règles. On se base sur deux principes de recherche, le succès et l'échec de filtrage qui nous permettent de réduire le nombre de tests à effectuer et d'optimiser le temps de détection. Par ailleurs on diminue énormément la consommation mémoire constatée lors de l'application de la première méthode de la classification des règles.

Nous poursuivons dans le prochain chapitre l'analyse du contenu des paquets en appliquant l'ensemble des règles sélectionnées. Une règle définit un ensemble de tests dont le filtrage de motifs constitue une opération très répandue et pesante sur le processus de vérification. Par ailleurs, le nombre de motifs augmente ce qui nous encourage à appliquer des algorithmes de filtrage multiple. Néanmoins les attaquants essaient d'empêcher la détection en répartissant les motifs sur plusieurs paquets. Alors que les autres systèmes de détection d'intrusions procèdent à un réassemblage du trafic, nous proposons dans le Chapitre 5 un filtrage à la volée du contenu des paquets qui tient compte de plusieurs signatures d'attaques et d'une arrivée désordonnée des paquets.

5

Filtrage efficace pour la détection d'intrusions

Sommaire

5.1 Introduction

Le filtrage de motifs constitue une opération très répandue en détection d'intrusions. Cependant, elle est pénalisante dans le cycle d'analyse des paquets. On considère qu'elle occupe 31% du total du temps d'exécution de Snort 1.6.3 [51]. De plus le nombre de motifs augmente en fonction des règles de détection d'attaques. Afin d'accélérer la recherche, des travaux récents proposent l'utilisation du filtrage multiple. Cette stratégie est efficace étant donnée la similarité remarquée entre diverses signatures d'attaques. Nous nous intéressons dans ce chapitre aux techniques de filtrage simple et multiple qui constituent le fondement d'une nouvelle stratégie de

filtrage à la volée pour détecter les signatures d'attaques. Les objectifs de notre nouvelle méthode est d'accélérer le filtrage et de mieux résister aux tentatives d'évasion.

En effet, la précision de recherche constitue un des atouts de la détection d'intrusions avec le filtrage de motifs. Par conséquent, les IDS forcent les attaquants à compliquer leurs plans d'attaques pour camoufler les activités malveillantes. Les intrus utilisent des techniques dites d'évasion. Ils envoient par exemple les paquets en désordre ou les superposent afin de construire des suites de paquets dont la reconstitution conduit à une attaque. Ils emploient également l'insertion qui consiste à envoyer des paquets vus uniquement par le système de détection d'intrusions mais ignorés par l'hôte. L'évasion est l'opération inverse de l'insertion. Elle consiste à envoyer plusieurs paquets dont certains sont ignorés par l'IDS.

Pour diminuer les risques de faux négatifs engendrés par les techniques d'évasion, les systèmes de détection d'intrusions ont recours au réassemblage du trafic. Il s'agit d'une opération intéressante puisque en plus de la reconstitution du trafic, elle offre l'opportunité de résoudre les ambiguïtés. Par exemple, on choisit les données à analyser si un hôte reçoit deux fois le même paquet. Cependant le réassemblage est coûteux en terme de temps et d'espace mémoire nécessaire pour collecter les paquets des mêmes connexions. Il retarde également l'opération de filtrage puisque on peut retrouver la signature d'attaque dés l'arrivée du premier paquet. Nous proposons dans ce chapitre une nouvelle approche de filtrage à la volée du trafic réseau. Cette méthode est capable de traiter chaque paquet dès sa réception tout en assurant la détection avec mémoire d'états.

Nous étudions dans la Section 5.2 quelques techniques de filtrage que nous utilisons pour mener le filtrage à la volée du trafic réseau. Ensuite nous décrivons dans la Section 5.3 l'évolution des techniques de filtrage appliquées à la détection d'intrusions. Afin de remédier aux problèmes d'évasion causés par la fragmentation et la segmentation du trafic, nous présentons dans la Section 5.4 une nouvelle stratégie de filtrage à la volée. La Section 5.5 détaille notre algorithme de filtrage dans le cadre d'une seule signature d'attaque. Ensuite la Section 5.6 étend la méthode pour supporter simultanément plusieurs motifs. La Section 5.7 analyse les résultats expérimentaux obtenus alors que la Section 5.8 conclut le chapitre en soulignant les limites du filtrage brut des signatures d'attaques.

5.2 Fondements

Nous développons dans ce chapitre une nouvelle stratégie de filtrage du trafic réseau. Notre méthode se base sur quelques structures de données et algorithmes de recherche de motifs. Nous consacrons la Section 5.2 à présenter ces différents techniques qui retrouvent la première instance d'un motif dans un texte. Nous notons par T le texte à filtrer et P (ou $P_i, i : 1..k$) le motif (l'ensemble de motifs) recherché(s). Par ailleurs les tailles du texte et du motif(s) sont respectivement n et m (m_i) alors que le nombre total de motifs est k et la somme des tailles des motifs est M.

5.2.1 Algorithme de filtrage naïf

La méthode naïve de recherche des motifs consiste à vérifier de gauche à droite et à chaque position du texte la correspondance des caractères du texte et du motif. Elle effectue ainsi des petits sauts d'un seul caractère pour parcourir le texte. L'algorithme naïf est acceptable si le premier caractère du motif est rare et on aura une complexité temps égale à $O(n-m)$. Cependant la méthode nécessite dans les pire cas $O((n-m+1)*m)$ opérations pour vérifier tout le texte.

Exemple 5.2.1 *Soient T="finefinding" et P="find". La Figure 5.1 montre l'exécution de l'algorithme de filtrage naïf.*

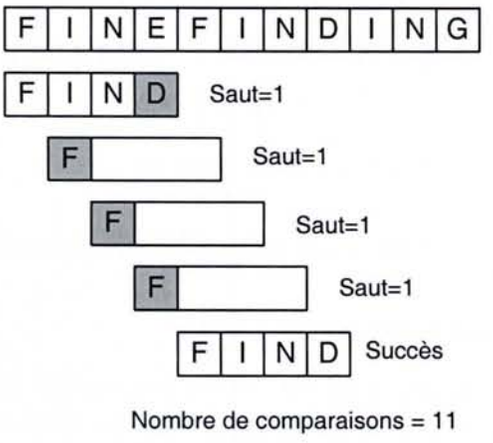


FIG. 5.1 – Méthode de filtrage naïf

5.2.2 Algorithme de Knuth-Morris-Pratt

L'algorithme de Knuth-Morris-Pratt (KMP) [76] est une extension de l'algorithme de Morris-Pratt (MP) [115]. De ce fait on explique d'abord le principe de la méthode MP. En effet, en analysant l'algorithme de recherche naïve on constate que si une discordance de caractères est trouvée après d comparaisons réussies alors la valeur du prochain saut est 1 malgré la connaissance des $d-1$ caractères qui suivent. Par exemple la première étape de recherche dans Figure 5.1 réussit à lire 3 caractères. Ainsi, l'idée principale de l'algorithme MP est de tirer profit de cette information et de calculer dans une phase de prétraitement et pour chaque position d du motif, la taille du plus long préfixe du motif qui coïncide avec un suffixe des d caractères (Formule 5.1) [22].

$$nextMP[d] = \begin{cases} -1 \;\; si \;\; d = 0. \\ max\{k/k < d, \;\; P[0..k-1] \; est \; un \; suffixe \; de \; P[0..d-1]\} \;\; sinon. \end{cases} \tag{5.1}$$

La méthode KMP optimise cette recherche en calculant un saut qui assure en plus que le caractère à la position $d+1$ du motif est différent du prochain caractère du préfixe trouvé (Figure 5.2). Cette garantie évite une nouvelle discordance à la position $d+1$ (Formule 5.2) [22].

$$nextKMP[d] = \begin{cases} -1 \;\; si \;\; d = 0. \\ max\{-1, k/k < d, P[0..k-1] \; est \; un \; suffixe \; de \; P[0..d-1] \; et \; P[k] \neq P[d]\}. \end{cases} \tag{5.2}$$

Les sauts effectués par la méthode KMP sont $d - nextKMP[d]$. L'algorithme présente une phase de prétraitement de complexités en temps et en espace égalent à m chacune. Quant à la phase de recherche, elle requiert au pire des cas une complexité en temps égale à $O(m+n)$.

Exemple 5.2.2 *On considère T="finefinfing" et P="finf". La Figure 5.2 montre l'exécution de l'algorithme de recherche de Knuth-Morris-Pratt. Suite à la première recherche, on reconnaît le préfixe FIN de taille 3. nextKMP[3]=-1 car FI et F ne sont pas des suffixes de FIN et P[0]=P[3].*

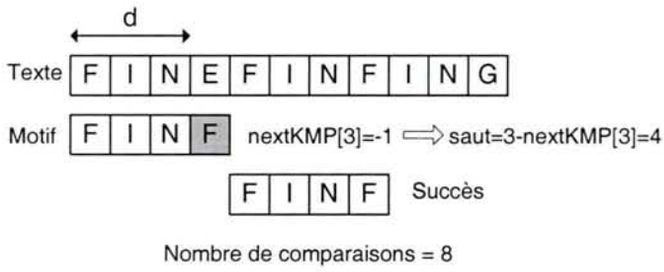
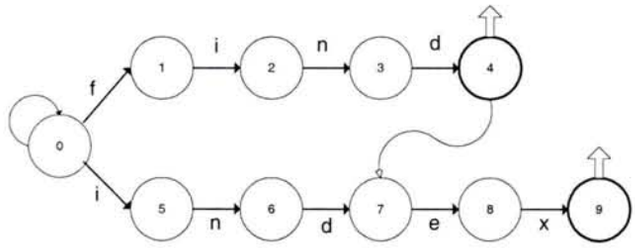


FIG. 5.2 – Méthode de filtrage Knuth-Morris-Pratt

5.2.3 Algorithme d'Aho-Corasick

L'algorithme d'Aho-Corasick (AC) [5] peut être vu comme une généralisation de l'algorithme KMP pour une recherche en parallèle de plusieurs motifs. La méthode construit d'abord l'arbre des préfixes des différents motifs (Figure 5.3). Ensuite elle marque les nœuds qui terminent la reconnaissance d'un ou plusieurs motifs. La dernière étape consiste à construire les pointeurs d'échec entre les nœuds pour représenter pour chaque mot sur l'arbre, son plus long suffixe qui appartient également à l'arbre. Il s'agit du même principe appliqué par la méthode KMP. L'algorithme d'Aho-Corasick affiche une complexité de prétraitement égale en temps à la somme des longueurs des motifs c'est à dire $M = \sum_{i=1}^{k} m_i$. Ensuite l'opération de recherche dure au maximum $O(n + M)$.

Exemple 5.2.3 *Soient T="finefinding", P_1="find" et P_2="index". La Figure 5.3 montre l'exécution de l'algorithme de recherche d'Aho-Corasick. Le premier échec s'effectue au nœud 3 en ayant le caractère 'e' au lieu de 'd'. Le pointeur d'échec du nœud 3 pointe vers le nœud 0.*



S	0	1	2	3	4	5	6	7	8	9
Echec(S)	0	0	0	0	7	0	0	0	0	0

Texte: finefinding
Motifs: find, index
Nombre de transitions avant de trouver "find"= 8

FIG. 5.3 – Méthode de filtrage d'Aho-Corasick

5.2.4 Algorithme de Boyer Moore

Nous présentons d'abord le principe de recherche de l'algorithme Boyer Moore. Ensuite nous discutons les extensions de cette méthode pour retrouver simultanément plusieurs motifs.

5.2.4.1 Principe de la recherche de Boyer Moore

Contrairement aux autres algorithmes déjà présentés, l'algorithme de Boyer Moore [16] effectue des lectures de droite à gauche du texte à filtrer. Il se base sur deux heuristiques de recherche qui permettent de calculer le saut le plus important. La première heuristique, appelée heuristique du mauvais caractère, essaie d'aligner le caractère non trouvé du texte T avec sa première occurrence de droite dans le motif (cas A de la Figure 5.4). Si ce caractère n'apparaît pas alors le saut replace le motif juste après le caractère non trouvé (cas B de la Figure 5.4). La deuxième heuristique, appelée heuristique du bon suffixe, essaie d'aligner la partie du texte déjà vérifiée (et donc égale à un certain suffixe du motif), avec sa prochaine occurrence dans le motif. Cette occurrence doit vérifier en plus que le prochain caractère de gauche du texte est différent du prochain caractère de gauche du préfixe du motif et cela afin d'éviter une nouvelle discordance (cas A de la Figure 5.5). Si on ne trouve pas la même occurrence alors on cherche le plus long préfixe du motif qui est un suffixe de la partie trouvée du motif (cas B de la Figure 5.5).

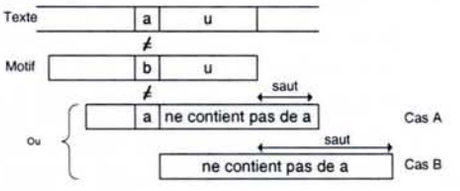


FIG. 5.4 – Heuristique du mauvais caractère de Boyer Moore

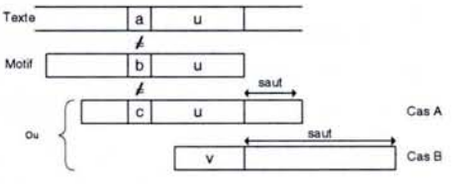


FIG. 5.5 – Heuristique du bon suffixe de Boyer Moore

L'algorithme de Boyer Moore est fréquemment utilisé dans diverses applications de recherche de motifs. Sa phase de prétraitement se déroule en $O(m+\sigma)$ avec σ la taille de l'alphabet utilisé et présente une complexité en espace égale à $O(m+\sigma)$. La complexité en temps d'exécution est dans le pire des cas quadratique et est égale à $O(mn)$. Cependant, cette complexité se réduit à $O(m/n)$ dans les meilleures conditions de filtrage.

Exemple 5.2.4 *Soient T="finefinding" et P="find". La Figure 5.6 montre l'exécution de l'algorithme de recherche de Boyer Moore. L'heuristique du mauvais caractère définit un saut de taille 4.*

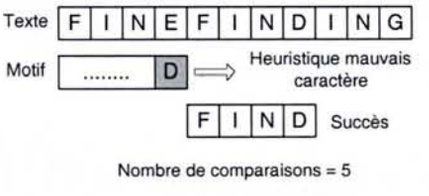


FIG. 5.6 – Méthode de filtrage de Boyer Moore

L'algorithme Boyer Moore Horspool [69] simplifie l'algorithme de Boyer Moore en ne tenant compte que de l'heuristique du mauvais caractère. La méthode s'avère intéressante lorsqu'on dispose d'un large alphabet ce qui correspond au cas du filtrage du trafic réseau pour détecter les motifs d'attaques.

5.2.4.2 Filtrage Multiple se basant sur Boyer Moore

Commentz-Walter [26] adapte l'algorithme de Boyer Moore pour filtrer simultanément plusieurs motifs. Il construit l'automate d'Aho-Corasick des motifs inversés afin de retrouver les préfixes lors d'une lecture de droite à gauche du texte. La vérification se poursuit jusqu'à retrouver un motif ou rencontrer un caractère non défini sur les liens issus de l'état courant de l'automate. La méthode calcule alors le décalage convenable en se basant sur les deux heuristiques de recherche de Boyer Moore.

En outre, Wu et Manber [188] s'inspirent de l'heuristique du mauvais caractère pour proposer une méthode de filtrage de plusieurs motifs. En effet, ils appliquent cette technique sur un bloc de caractères généralement de taille 2 ou 3. La valeur optimale de ce bloc est $log_\sigma 2M$ avec σ la taille de l'alphabet et M la somme totale des tailles des motifs. La recherche s'effectue de droite à gauche et s'il y a une correspondance entre le bloc des caractères lus et le bloc suffixe d'un ou plusieurs motifs, alors une vérification directe se réalise. Cette opération peut s'effectuer également par rapport à un bloc préfixe des motifs ce qui permet de résoudre le problème des suffixes fréquents dans les textes à filtrer (par exemple "ing" et "ion" pour des textes en anglais).

5.2.5 Graphes orientés acycliques de mots : DAWG

Le DAWG [15] est une structure de données qui permet de maintenir tous les sous mots d'un ou plusieurs motifs. Dans cette partie, nous définissons d'abord la structure DAWG appelée aussi automate minimal des suffixes. Ensuite nous présentons deux méthodes de filtrage qui se basent sur le DAWG.

5.2.5.1 Construction du DAWG

On s'intéresse dans cette partie à la construction du DAWG [85]. On note par Σ l'alphabet utilisé et on considère le mot $v = v_1 w v_2 \in \Sigma^*$. On note par $|w|$ la longueur du mot. De plus, on dit que w apparaît dans v à la position $|v_1|$ et il est à la position de fin $|v_1 w|$. On note par $pref(v)$, $suff(v)$ et $sub(v)$ respectivement les ensembles des préfixes, suffixes et sous mots de v.

Définition 5.2.1 (Ensemble de position de fin $endpos_D$**)** *Soit $D = \{v_1, v_2, ..., v_n\}$ et $w \in \Sigma^*$. Une position (resp. une position de fin) de w dans D est un couple (i, j) tel que w apparaît dans v_i à la position (resp. position de fin) j. L'ensemble des positions de fin de w dans D est noté $endpos_D(w)$.*

Définition 5.2.2 (Relation d'équivalence $\equiv_D$**)** *Soit $D = \{v_1, v_2, ..., v_n\} \subset \Sigma^*$. On définit pour $u, v \in sub(D)$, la relation d'équivalence $\equiv_D$ suivante : $u \equiv_D v \Leftrightarrow endpos_D(u) = endpos_D(v)$. On note $[u]_D$ la classe d'équivalence de u selon $\equiv_D$.*

Définition 5.2.3 (DAWG) *Soit $D = \{v_1, v_2, ..., v_n\} \subset \Sigma^*$. Le DAWG A_D de D est le graphe acyclique orienté (X, U) avec $X = \{[u]_D | u \in sub(D)\}$ et $U = \{([u]_D, [ua]_D) | u \in sub(D), ua \in sub(D), a \in \Sigma\}$. L'arc $([u]_D, [ua]_D)$ est étiqueté par a. Le nœud $[\epsilon]_D$ est le nœud source de A_D. Intuitivement, le DAWG est une structure qui permet de lire tous les sous mots de D.*

Définition 5.2.4 (Lien suffixe) *Supposons que* $endpos_D(u) \subseteq endpos_D(v)$ *et que il n'existe pas* $w \in sub(D)$ *tel que* $endpos_D(u) \subseteq endpos_D(w) \subseteq endpos_D(v)$. *On définit alors le lien suffixe de* $[u]_D$ *comme étant l'arc reliant ce nœud au nœud de la classe* $[v]_D$. *Intuitivement, il s'agit du lien vers la classe contenant le plus long préfixe de u.*

Exemple 5.2.5 *Soit D={find,in} ; Sub(D)={ϵ,n,i,d,f,in,nd,fi,ind,fin,find}*
La Figure 5.7 montre le DAWG associé à l'ensemble D. Il est construit par rapport aux classes d'équivalences suivantes :

- $[f] = \{(1,1)\}$
- $[fi] = \{(1,2)\}$
- $[fin] = \{(1,3)\}$
- $[d] \equiv_D [nd] \equiv_D [ind] \equiv_D [find] = \{(1,4)\}$
- $[n] \equiv_D [in] = \{(1,3),(2,2)\}$
- $[i] = \{(1,2),(2,1)\}$
- $[\epsilon] = \{(1,0),(1,1),(1,2),(1,3),(1,4),(2,0),(2,1),(2,2)\}$

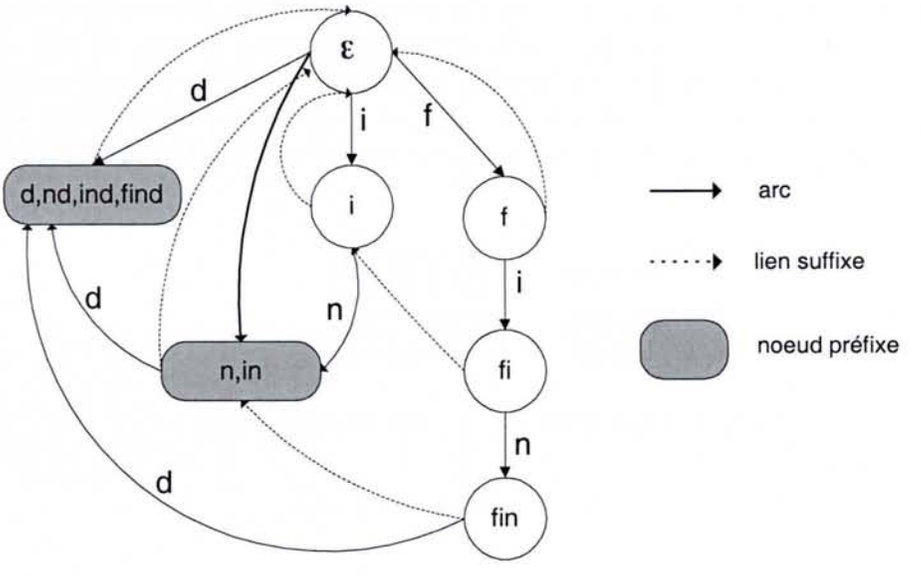


FIG. 5.7 – Structure DAWG des motifs "find" et "in"

Le DAWG est notre principale structure pour réaliser le filtrage partiel des motifs d'attaques. Sa construction est linéaire en temps $O(M)$ et requiert un espace mémoire égale à $O(M\sigma)$ avec σ la taille de l'alphabet et M la taille totale de tous les motifs. Cette opération qui tient compte de plusieurs motifs se déroule pendant l'initialisation de l'IDS. Nous présentons dans la suite trois algorithmes qui se basent sur le DAWG pour retrouver un ou plusieurs motifs.

5.2.5.2 Algorithme Forward DAWG

L'algorithme de filtrage Forward DAWG utilise la structure DAWG pour lire de gauche à droite le texte à la recherche d'un ou plusieurs motifs [31]. L'idée de la méthode consiste au fait qu'un sous mot de taille égale à la longueur du motif correspond peut être à une opération de filtrage réussie. L'algorithme lit le texte à filtrer caractère par caractère ce qui exige n opérations

de lecture. Durant cette recherche, on part du nœud source du DAWG puis au fur et à mesure qu'on lit un caractère, l'état courant du DAWG est mis à jour en utilisant les arcs étiquetés entre les classes d'équivalence des sous mots et en cas d'échec, en prenant les liens suffixes. On calcule à chaque état visité la taille du sous mot retrouvé jusqu'à atteindre la taille du motif recherché ou terminer la lecture du texte.

Exemple 5.2.6 *Soient T="finefinding" et P="find". La Figure 5.7 constitue également le DAWG de l'unique motif "find" puisque "in" est un sous mot de "find". En utilisant ce graphe, l'opération de filtrage réussie en 9 opérations. Chaque caractère lu engendre une seule transition sauf le caractère "e" qui emprunte 2 fois les liens suffixes.*

5.2.5.3 Algorithme Reverse Factor

La méthode de recherche Reverse Factor [94] utilise l'automate minimal de suffixes (ou DAWG) mais l'applique sur des motifs inversés. On parle alors des DAWG inversés des motifs ou BDAWG. La méthode recherche les suffixes des motifs inversés ce qui correspond parfaitement aux inverses des préfixes des motifs originaux (Figure 5.8). Durant cette phase de recherche, on lit les caractères de droite à gauche pour trouver le plus long préfixe. Si la taille de ce préfixe est égale à la taille d'un motif alors il s'agit d'une opération de filtrage réussie. De plus, il est facile de définir la valeur du saut connaissant la taille du plus long préfixe. La méthode de filtrage s'adapte particulièrement aux longs motifs. De plus, et malgré une complexité de pire des cas quadratique en temps et égale à $O(mn)$, la technique présente en moyenne une bonne complexité égale à $O(m + n\frac{log_n m}{m})$.

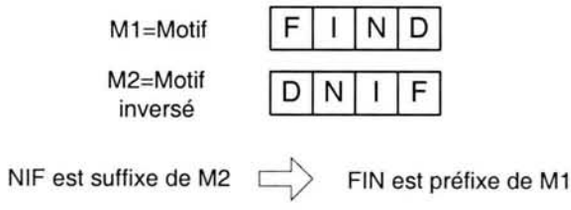


FIG. 5.8 – Recherche des préfixes

Exemple 5.2.7 *Soient T="finefinding" et P="find". La Figure 5.9 montre l'exécution de l'algorithme du Reverse Factor. Suite à la première lecture, on ne trouve aucun préfixe. Le saut réalisé est alors égale à la taille du motif.*

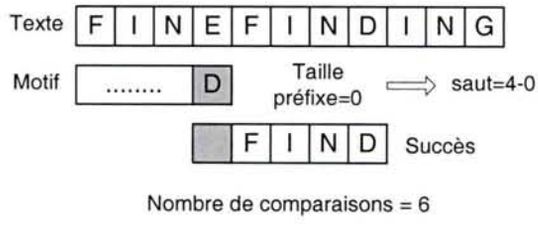


FIG. 5.9 – Méthode de filtrage de Reverse Factor

5.2.5.4 Algorithme Match DAWG

La méthode Match DAWG est proposée dans [30] et elle combine les deux méthodes de recherche d'Aho-Corasick et du Reverse factor. La technique assure le filtrage de plusieurs motifs et peut être appliquée pour un seul motif en remplaçant la structure d'Aho-Corasick par une recherche de Knuth-Morris-Pratt. L'idée de base consiste à localiser les préfixes des motifs en lisant le texte de droite à gauche via le BDAWG défini par le Reverse Factor. Ensuite, il est possible d'utiliser la méthode d'Aho-Corasick pour lire le texte de gauche à droite et retrouver les motifs. L'intérêt de cette double lecture est qu'il est possible maintenant de définir des sauts suite à des échecs de lecture avec la structure d'Aho-Corasick. En effet, on associe à chaque état S de l'automate d'Aho-Corasick, le saut optimal qui garantie qu'aucun motif n'est oublié. Formellement, ce saut est défini par la distance minimale séparant l'état S (qui correspond à un préfixe commun des motifs) et un état final de la structure d'Aho-Corasick (Formule 5.3). Cette distance doit tenir compte également des chemins définis par des liens d'échec d'Aho-Corasick (Formule 5.4).

$$saut[S] = \begin{cases} m - i - 1 \; si \; i \neq m \; (\; avec \; i \; = \; taille \; du \; prefixe \; menant \; a \; S) \\ m \; sinon \end{cases} \tag{5.3}$$

$$saut[S] = min(saut[S], saut[f(S)]) \tag{5.4}$$

Ayant obtenu le saut optimal, on sauvegarde la dernière position visitée du texte (appelée position critique) ainsi que l'état courant de la structure AC (Figure 5.10). Ensuite on lit le texte de droite à gauche à partir de la position du saut pour déterminer les préfixes des motifs. Si on atteint la position critique alors on reprend la recherche de gauche à droite à partir de l'état sauvegardé de la structure d'Aho-Corasick (AC). Sinon, on utilise l'état initial d'AC pour lire le texte à partir de la dernière position visitée par le Reverse Factor.

La technique inspecte au maximum deux fois le texte à filtrer. Cependant, si les motifs sont assez longs et la taille totale des motifs est petite (c'est-à-dire les motifs n'engendrent pas des possibilités de filtrage fréquentes) alors la lecture du texte devient sous linéaire et égale à $O(n(logm))/m)$ [30].

Exemple 5.2.8 *Soient T="finefinding" et P="find". La Figure 5.10 montre l'exécution de l'algorithme Match DAWG.*

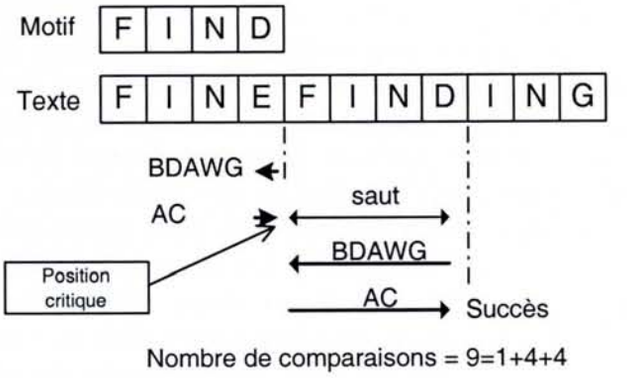


FIG. 5.10 – Méthode de filtrage de Match DAWG

Nous avons présenté tout au long de cette section les algorithmes de recherche de motifs que nous utilisons pour assurer un filtrage à la volée des signatures d'attaques. Avant de développer notre méthode, nous décrivons quelques techniques de filtrage couramment appliquées dans la détection d'intrusions. La majorité de ces méthodes utilisent des algorithmes de filtrage détaillés dans la Section 5.2.

5.3 Evolution des techniques de filtrage pour la détection d'intrusions

Le filtrage de motifs est fréquemment employé dans divers systèmes de détection d'intrusions. Snort utilisait dans ses premières versions la méthode naïve pour retrouver les signatures d'attaques. Afin accélérer sa recherche, l'IDS a intégré ensuite l'algorithme de filtrage Boyer Moore [41]. Cependant la méthode n'assure pas un filtrage simultané de plusieurs motifs. Étant donnée l'augmentation rapide du nombre des motifs et leurs similarités, Fisk et Varghese [51] introduisent la méthode de filtrage Set Wise Boyer Moore. Il s'agit d'une extension de l'algorithme Boyer Moore Horspool dans le cas de plusieurs motifs. Le saut total considéré est le minimum des sauts définis par rapport à chaque motif. Fisk et Varghese comparent les performances de l'algorithme avec la méthode d'Aho-Corasick et l'algorithme de Boyer Moore Horspool. Ils améliorent le temps de recherche d'un facteur de 4,6 par rapport à la stratégie de filtrage séparé de chaque motif d'attaque. Les deux algorithmes de filtrage multiple (Aho-Corasick et Set Wise Boyer Moore) présentent pratiquement les mêmes résultats. Afin d'optimiser la recherche, les auteurs [51] suggèrent une utilisation dynamique de 3 méthodes de filtrage. En effet, la recherche Boyer Moore Horspool suffit pour retrouver un seul motif d'attaque. Par contre, la présence de plusieurs motifs dont le nombre est inférieur à 100 incitent les auteurs à favoriser le Set Wise Boyer Moore. Enfin ils utilisent l'algorithme d'Aho Corasik si le nombre de motifs est supérieur à 100.

Coit, Staniford et McAlerney [25] étudient le filtrage simultané de plusieurs motifs d'attaques. Ils proposent une combinaison d'Aho-Corasick et du Boyer Moore pour assurer un filtrage rapide des signatures d'attaques. La méthode est symétrique à l'algorithme de Comment-Walter. En effet, la lecture des caractères s'effectue de gauche à droite en commençant par la fin du paquet. De plus, l'arbre d'Aho-Corasick est construit à partir des préfixes non inversés des motifs. L'heuristique du mauvais caractère reste la même en prenant le minimum des sauts définis par chaque motif. Par contre l'heuristique du bon suffixe est remplacée par une heuristique du bon préfixe. Il s'agit en fait de localiser l'occurrence d'un préfixe comme un facteur d'un motif et à défaut, de trouver le plus long suffixe de ce préfixe qui est un préfixe d'un des motifs.

Markatos et ses co-auteurs [113, 6] proposent une autre technique de filtrage approximative. Ils divisent les motifs en mots de bits puis ils vérifient leur présence dans le contenu du paquet. Si un des mots ne figure pas dans le paquet alors l'opération de filtrage échoue. Par contre, si tous les blocs de bits existent alors la méthode exige la vérification du bon ordre. Cela passe par l'utilisation d'un algorithme de filtrage standard tel que Boyer Moore.

Notons enfin que la nouvelle version de Snort (2.0) [124] intègre une implantation de l'algorithme de Wu-Manber (voir sous-section 5.2.4.2). Les résultats expérimentaux montrent l'efficacité de l'algorithme par rapport au Set Wise Boyer Moore. De plus Tuck et ses co-auteurs [176] proposent deux versions optimisées d'Aho-Corasick pour réduire la consommation mémoire. Ils utilisent deux techniques employées auparavant dans la recherche des adresses IP. Il s'agit de la compression Bitmap et la compression du chemin. Leur travail s'adapte bien à une implémentation matérielle d'un système de détection d'intrusions.

Toutes ces méthodes de filtrage exigent avoir le contenu complet des paquets avant d'entamer la recherche des signatures d'attaques. Nous supposons dans la suite avoir une partie du trafic pour déclencher l'opération de filtrage. Cette opération se poursuit ensuite au fur et à mesure de la réception des données.

5.4 Mécanisme du filtrage à la volée

Les attaquants utilisent plusieurs techniques pour contourner les IDS (Chapitre 2, sous-section 2.4.1.1). Divers préprocesseurs sont introduits dans Snort afin de contrarier cette nouvelle génération d'attaques. On distingue en particulier les préprocesseurs Frag2 et Stream4 [137]. Frag2 définit par défaut une limite de 4 Mégaoctets de mémoire et 30 secondes de temps pour rassembler les fragments IP. D'autre part, Stream4 est configuré par défaut avec une limite de 8 Mégaoctets de mémoire et une période de 30 secondes pour rassembler les flux TCP.

Il est indéniable que ces préprocesseurs améliorent le niveau de sécurité assurée par Snort. Cependant, le stockage des paquets dans un espace mémoire et la définition d'une période de temps rendent l'IDS vulnérable aux attaques par déni de services et introduisent des retards dans la procédure de détection. Par ailleurs, l'allocation de l'espace mémoire dépend des services analysés alors que la période du temps varie selon la capacité du réseau et la quantité du trafic véhiculé au moment de la détection. En outre, les attaquants peuvent se synchroniser avec la période de temps pour réussir leurs exploits. Il en résulte que fixer les valeurs de ces deux paramètres est une tâche assez cruciale.

Nous introduisons une nouvelle méthode pour supporter la fragmentation et la segmentation du trafic réseau. L'idée consiste à traiter le trafic dès sa réception mais en sauvegardant les résultats de l'analyse dans des structures de données appelées *traces* plutôt que d'enregistrer les paquets. Nous modifions les méthodes de filtrage pour supporter cette stratégie. De plus, une bonne gestion des listes des traces est indispensable pour éliminer rapidement les traces devenues inutiles. Nous accélérons ainsi l'opération de filtrage et par suite le processus de détection. Dans le reste de cette introduction, nous définissons nos hypothèses de travail et nous donnons le fonctionnement global de la solution.

Le mécanisme de détection associe un contexte à chaque nouvelle connexion. Cette étape s'effectue en inspectant les premiers échanges des paquets lors d'une ouverture de connexion. Ensuite, les prochains paquets sont liés au contexte créé. Si le contexte n'existe pas, ou il n'est pas repéré (paquet rejeté, la connexion est active avant le déploiement de l'IDS) alors on détecte un événement suspect. Par contre et dans les situations normales, on associe à chaque paquet une trace qui contiendra *les informations d'emplacement* et *les résultats de filtrage partial.*

Les informations d'emplacement servent à insérer les traces dans la liste des traces associée au contexte crée. Une retransmission d'un paquet engendre la création de deux traces équivalentes. Nous privilégions dans ce travail la première version du paquet. De plus, nous abandonnons la recherche en cas de chevauchement entre les traces. Nous générons dans ces conditions un message d'alerte qui sera enregistré dans un fichier journal de sécurité.

Les opérations de filtrage partial vérifient cinq propriétés (Figure 5.11) que nous sauvegardons dans des listes appelées *listes de connaissances* :

- **P1 :** Le paquet contient une signature d'attaque.
- **P2 :** Le paquet est un sous mot d'une signature d'attaque.
- **P3 :** Le paquet contient un suffixe qui est un préfixe d'une signature d'attaque.
- **P4 :** Le paquet contient un préfixe qui est un suffixe d'une signature d'attaque.
- **P5 :** Le paquet ne comporte aucun sous mots d'une signature d'attaque.

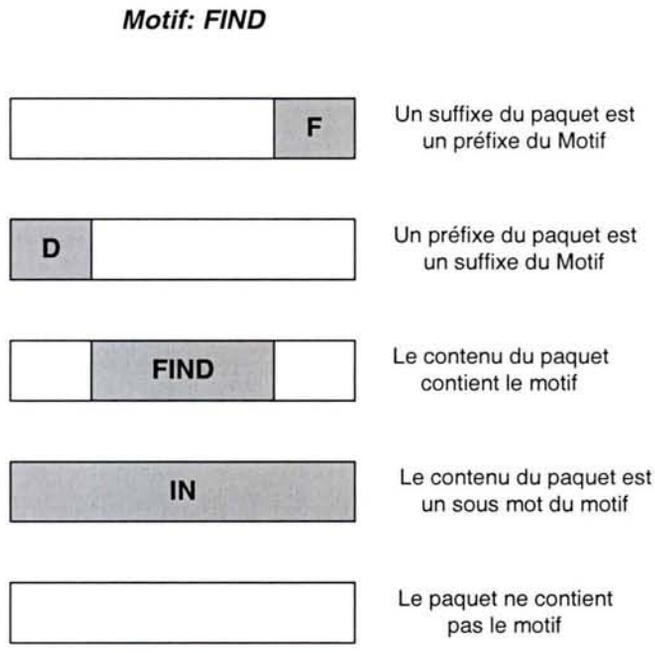


FIG. 5.11 – Apparition des motifs d'attaques dans le contenu d'un paquet

Ayant rempli les listes de connaissances, nous extrayons un *ensemble d'exigences* qui décrivent les prochains événements nécessaires pour retrouver le motif d'attaque. Afin de vérifier ces informations, nous récupérons sur la trace du dernier paquet reçu, les exigences des traces immédiatement voisines. Nous décidons ensuite de la présence du motif. Enfin, nous éliminons les traces devenues inutiles et nous traitons le prochain paquet. Ce nouveau mécanisme de détection évite les inconvénients évoqués du réassemblage du trafic et accélère la détection puisque les premières données peuvent déjà contenir un motif d'attaque. Nous détaillerons dans la suite chacune des étapes du traitement d'un paquet.

5.4.1 Création du contexte de connexion

Les attaquants forgent les paquets IP et les envoient aléatoirement au réseau cible sans ouvrir de connexions. Leurs objectifs sont multiples. Ils tentent via une reconnaissance active de découvrir les caractéristiques du réseau cible. Ils peuvent lancer des attaques de déni de services contre les systèmes de détection d'intrusions en les surchargeant de données incomplètes. Les anciens IDSs analysent automatiquement le contenu du trafic destiné aux hôtes du réseau. Cependant, si ces machines ne sont pas prêtes à recevoir ce genre de trafic, elles rejettent directement les paquets. Le travail effectué par les IDSs (filtrage, analyse protocolaire) s'avère alors inutile. Par suite, les IDS doivent superviser le bon déroulement des connexions avant de vérifier les contenus des paquets.

Nous créons des contextes de connexions pour superviser les connexions actives. Ces contextes constituent une preuve que les machines qui reçoivent le trafic sont prêtes à accepter les paquets. La création du contexte dépend du protocole de transport utilisé. Ainsi, et pour des protocoles avec connexion tels que TCP, l'initialisation du contexte s'effectue par rapport aux demandes d'ouverture de connexions. Par contre les protocoles sans connexions tels que UDP présentent des contextes temporaires.

5.4.1.1 Contextes des connexions TCP

Nous nous intéressons dans cette sous section à la création des contextes des flux TCP. Cette tâche est assez complexe puisqu'elle exige superviser toutes les ouvertures de connexions effectuées sur le réseau. Les données reçues sans contexte de connexions sont considérées invalides. Toutes fois il y a un risque de faux positifs si des paquets sont perdus en cours d'analyse ou que des ouvertures de connexions sont réalisées avant le déploiement de l'IDS.

Un IDS supervise les paquets SYN pour connaître les demandes d'ouverture de connexions TCP. Il prépare ainsi un contexte de connexion et déclenche un timeout pour l'activation réelle de ce contexte. Ceci passe par l'accomplissement de l'opération du "Three Way Handshake". Le contexte est caractérisé par le quadruplet (adresse source, port source, adresse destination, port destination) et sera inséré dans un arbre équilibré pour accélérer sa mise à jour. Chaque nœud de l'arbre représente un seul contexte et comporte le numéro de séquence initial de la connexion et une liste de traces des résultats de filtrage partiel appliqué sur chaque paquet. De plus et afin de garder une taille minimale de l'arbre de connexions, nous libérons le contexte dès la clôture de la connexion. Ceci est réalisé dès la réception du deuxième paquet FIN ou du premier paquet RST appartenant à une connexion active.

Par ailleurs, nous distinguons entre deux types de connexions : les connexions ouvertes en activité et les connexions ouvertes à l'état passif. En effet, nous définissons un premier timeout à partir duquel nous effaçons les informations contenues dans le contexte. Dans ce cas, les informations de filtrage sont considérées obsolètes. Ensuite et si la durée d'inactivité se prolonge, nous transférons le contexte de l'arbre des connexions actives à l'arbre des connexions ouvertes à l'état passif. Ce mécanisme nous assure la rapidité d'accès et de mises à jour des connexions actives.

La notion de contexte assure principalement deux tâches durant l'opération de filtrage. D'abord elle vérifie la validité de la connexion TCP avant de déclencher l'opération de filtrage. De plus, *elle est indispensable pour collecter dans une même liste toutes les traces des paquets appartenant à une même connexion.* Cette opération est nécessaire pour réaliser le filtrage à la volée des signatures d'attaques. Par ailleurs, les contextes de connexions offrent d'autres opportunités de détection :

- supervision de l'exécution normale du protocole de transport : l'établissement d'une connexion TCP suit un schéma d'exécution bien défini. La violation de certaines propriétés révèle parfois des tentatives d'attaques sévères. Nous mentionnons en particulier la prédiction des numéros de séquences TCP, le détournement des sessions TCP et l'inondation SYN.
- détection des balayages de ports : les attaquants créent des connexions TCP anormales pour tester le comportement des machines distantes et par suite connaître leurs propriétés (système d'exploitation, liste des ports ouverts). On s'intéresse particulièrement aux tentatives de balayage de ports avec ouverture totale de connexion, demi ouverture de connexion et les balayages furtifs. Ces techniques qui seront détaillées dans l'Annexe A, sont facilement détectables en inspectant l'arbre de contextes.
- accès rapide à la liste des traces : un segment TCP n'est analysé que s'il appartient déjà à une connexion ouverte. Par conséquent, notre première tâche consiste à chercher le contexte de la connexion. Nous sauvegardons le résultat de cette recherche car elle permet aussi de localiser la liste des traces associée à la connexion TCP. Ainsi on insère rapidement dans cette liste la nouvelle trace du dernier paquet reçu.

5.4.1.2 Contextes des liaisons UDP

L'IDS construit un nouveau contexte pour chaque paquet fragmenté en UDP. Le contexte est caractérisé par le quadruplet (adresse source, port source, adresse destination, port destination) et il est inséré dans un arbre équilibré. Ensuite on l'élimine si tous les fragments du paquet ont été reçus ou après une certaine période d'inactivité. De plus les informations des deux messages d'erreurs ICMP : destination non atteignable et port fermé, permettent de mieux gérer les entrées dans l'arbre de contextes.

5.4.1.3 Contextes des messages ICMP

Les messages ICMP constituent parfois des canaux de communications utilisés par les chevaux de Troie (Loki, Stacheldraht [27]). Le filtrage de ce trafic s'avère alors nécessaire. Nous créons des contextes pour les messages fragmentés en ICMP. Le contexte est caractérisé par le couple (adresse source, adresse destination) et il est stocké dans un arbre équilibré. Nous assurons ainsi le filtrage à la volée de ce type de trafic et nous détectons en plus des attaques de fragmentation classique tel que le Ping de la mort [27].

5.4.2 Création des traces et insertion dans la liste des traces

Suite à la création des contextes de connexions, nous étudions dans cette sous section la deuxième étape de l'analyse à la volée du trafic. Durant cette phase nous associons à chaque paquet TCP ou paquet fragmenté UDP ou ICMP, une trace pour collecter les informations de filtrage. La trace contient également des informations d'emplacement nécessaires pour la reconstruction exacte de la liste des traces. En particulier nous nous intéressons à l'identificateur du paquet IP ($E.id$), la taille des données ($E.l$), le numéro de séquence TCP ($E.sn$), le décalage des données fragmentées ($E.offset$) et les bits de fragmentation ($E.mf$, $E.df$).

L'insertion dans la liste des traces permet non seulement de fusionner les informations de filtrage mais également de détecter quelques attaques supplémentaires telles que le Boink, le Teardrop et le Ping de la mort [27]. En effet, on détecte facilement les gros messages ICMP de taille supérieure à 65 Kilo octets et fragmentés sur plusieurs paquets. De plus on découvre les chevauchements entre les paquets IP ou les segments TCP.

5.4.3 Construction des listes de connaissances

Après avoir vérifié la présence du contexte de connexion et préparé la trace du nouveau paquet reçu, nous entamons la phase de filtrage. Notre but est de retrouver les instances des quatre premières propriétés évoquées au début de la Section 5.1. Ainsi nous remplissons les listes de connaissances suivantes :

- $E.Lq$: liste des positions où le paquet est un sous mot du motif. On utilise la structure DAWG pour remplir le contenu de cette liste et par suite chercher la propriété P2 du paquet. On vérifie d'abord que la taille du paquet est inférieure ou égale à la taille du motif afin de s'assurer que le paquet peut constituer un sous mot du motif. Ensuite on exécute un filtrage de gauche à droite via la structure DAWG jusqu'au premier échec.
- $E.Ls$: liste des tailles des préfixes qui sont des suffixes du motif. Nous vérifions alors la propriété P3 en filtrant du gauche à droite et jusqu'au premier échec le contenu du paquet reçu. Nous utilisons la structure DAWG et nous profitons des états suffixes pour déterminer les suffixes du motif.

- $E.Lp$: liste des tailles des suffixes qui sont des préfixes du motif. Il s'agit de retrouver les instances de la propriété P4 et cela en se basant sur la structure BDAWG. En effet, les inverses des préfixes d'un mot sont des suffixes du mot inversé (Figure 5.8). Ainsi en utilisant la structure BDAWG et en lisant le paquet de droite à gauche jusqu'au premier échec, on détermine les préfixes du motif.
- $E.Lm$: liste des positions du motif dans le paquet. On vérifie la propriété P1 en cherchant les instances du motif dans le contenu du paquet. Une condition nécessaire pour effectuer le filtrage est que la taille du paquet soit supérieure à celle du motif. Ensuite, on peut utiliser la structure DAWG et appliquer l'algorithme de filtrage Forward DAWG. Cependant et afin d'optimiser la recherche, nous employons l'algorithme de Boyer Moore Horspool. En effet, cette méthode offre des sauts de lecture plus importants lors des échecs dans l'opération de filtrage.

5.4.4 Construction des ensembles d'exigences

La quatrième étape de filtrage consiste à construire à partir des listes de connaissances, un ensemble d'exigences qui décrivent ce qui reste à vérifier afin de retrouver le motif. Par exemple si nous avons un préfixe de taille q alors il suffit pour reconnaître tout le motif, d'avoir un suffixe ou une liste de sous mots et un suffixe, successeurs du paquet, ordonnées et de longueur totale égale à $|sig| - q$ avec $|sig|$ la taille de la signature.

On distingue 3 types d'exigences : P, S et PS. Une exigence $E.req.P$ de type P complète les connaissances de la liste des suffixes $E.Ls$ alors qu'une exigence $E.req.S$ complète la liste des préfixes $E.Lp$. Enfin une exigence $E.req.PS$ est une conjonction des deux exigences $E.req.P$ et $E.req.S$. Elle est générée suite à la découverte d'un sous mot du motif pour décrire les préfixes et les suffixes nécessaires pour retrouver le motif.

5.4.5 Transfert des ensembles d'exigences

Nous associons à chaque contexte de connexion une liste de traces pour contenir les résultats du filtrage partiel. Rappelons qu'une trace représente un paquet et donc elle est insérée dans la liste des traces par rapport à la position du paquet dans le flux du trafic analysé (sous section 5.4.2). Nous récupérons dans cette étape les exigences des traces immédiatement voisines à la trace du dernier paquet reçu. Nous combinons ces informations avec les listes de connaissances de la trace. Nous construisons ainsi de nouvelles exigences dans la trace courante qui seront à leur tour transférées vers les prochaines traces voisines et cela jusqu'à la reconnaissance complète du motif ou la fermeture de la connexion. Le transfert des exigences entre les traces voisines se base sur 16 règles de transfert qui tiennent compte des diverses situations de filtrage. Nous présentons dans la suite l'ensemble de ces règles, puis nous expliquons la règle numéro 8 :

1. Si E_k.req.P=α, E_{k-1}.l $\geq \alpha$ et $\alpha \notin E_{k-1}$.Lp $\rightarrow$ élimination de l'exigence.
2. Si E_k.req.P=α, E_{k-1}.l $\geq \alpha$ et $\alpha \in E_{k-1}$.Lp $\rightarrow$ motif trouvé.
3. Si E_k.req.P=α, E_{k-1}.l $< \alpha$ et (α - E_{k-1}.l) $\notin E_{k-1}$.Lq $\rightarrow$ élimination de l'exigence.
4. Si E_k.req.P=α, E_{k-1}.l $< \alpha$ et (α - E_{k-1}.l) $\in E_{k-1}$.Lq $\rightarrow$ élimination de l'exigence + création d'une nouvelle exigence dans la trace E_{k-1} : E_{k-1}.req.P= α - E_{k-1}.l

5. Si E_k.req.S=β, E_{k+1}.l $\geq \beta$ et $\beta \notin E_{k+1}$.Ls → élimination de l'exigence.
6. Si E_k.req.S=β, E_{k+1}.l $\geq \beta$ et $\beta \in E_{k+1}$.Ls → motif trouvé.
7. Si E_k.req.S=β, E_{k+1}.l $< \beta$ et (β - E_{k+1}.l) $\notin E_{k+1}$.Lq → élimination de l'exigence.
8. Si E_k.req.S=β, E_{k+1}.l $< \beta$ et (β - E_{k+1}.l) $\in E_{k+1}$.Lq → élimination de l'exigence + création d'une nouvelle exigence dans la trace E_{k+1} : E_{k+1}.req.S= β - E_{k+1}.l

9. Si E_k.req.PS=(α, β), E_{k-1}.l $\geq \alpha$ et $\alpha \notin E_{k-1}$.Lp → élimination de l'exigence.
10. Si E_k.req.PS=(α, β), E_{k+1}.l $\geq \beta$ et $\beta \notin E_{k+1}$.Ls → élimination de l'exigence.
11. Si E_k.req.PS=(α, β), E_{k-1}.l $\geq \alpha$ et $\alpha \in E_{k-1}$.Lp → élimination de l'exigence + création d'une nouvelle exigence dans la trace E_k : E_k.req.S= β.
12. Si E_k.req.PS=(α, β), E_{k+1}.l $\geq \beta$ et $\beta \in E_{k+1}$.Ls → élimination de l'exigence + création d'une nouvelle exigence dans la trace trace E_k : E_k.req.P= α.
13. Si E_k.req.PS=(α, β), E_{k-1}.l $< \alpha$ et (α - E_{k-1}.l) $\notin E_{k-1}$.Lq → élimination de l'exigence.
14. Si E_k.req.PS=(α, β), E_{k+1}.l $< \beta$ et (β - E_{k+1}.l) $\notin E_{k+1}$.Lq → élimination de l'exigence.
15. Si E_k.req.PS=(α, β), E_{k-1}.l $< \alpha$ et (α - E_{k-1}.l) $\in E_{k-1}$.Lq → élimination de l'exigence + création d'une nouvelle exigence dans la trace E_k : (P_2 : E_k.req.S= β) $\wedge$ P_1 + création d'une nouvelle exigence dans la trace E_{k-1} : (P_1 : E_{k-1}.req.P= α - E_{k-1}.l) $\wedge$ P_2
16. Si E_k.req.PS=(α, β), E_{k+1}.l $< \beta$ et (β - E_{k+1}.l) $\in E_{k+1}$.Lq → élimination de l'exigence + création d'une nouvelle exigence dans la trace trace E_k : (P_1 : E_k.req.P= α) $\wedge$ P_2 + création d'une nouvelle exigence dans la trace E_{k+1} : (P_2 : E_{k+1}.req.S= β - E_{k+1}.l) $\wedge$ P_1

Par exemple la règle numéro 8 traite le cas d'une trace E_{k+1} associée à un paquet de longueur $E_{k+1}.l$ et il existe une trace voisine E_k qui contient une exigence de type S égale à β. Dans ces conditions, si la liste des sous mots de la trace E_{k+1} contient la position β alors on élimine l'exigence S de E_k et on construit une autre exigence dans la trace E_{k+1} de type S égale à $\beta - E_{k+1}.l$.

La Figure 5.12 montre un scénario de transfert d'exigences durant lequel nous avons considéré le motif d'attaque "FIND" et marqué en gras la trace courante du dernier paquet reçu. Dans ce scénario nous avons déjà une exigence de type PS dans la trace d'ordre n indiquant qu'il reste encore un préfixe et un suffixe de taille 1 chacun pour retrouver le motif "FIND". Nous recevons ensuite le paquet d'ordre $n + 1$ contenant la chaîne de caractères "DABF". Les opérations de filtrage partiel permettent de découvrir un suffixe et un préfixe de taille 1 chacun. On utilise ensuite la règle de transfert 12 ce qui permet de modifier l'exigence PS de la trace d'ordre n en une exigence de type P. Par ailleurs et puisque la trace d'ordre $n + 1$ ne possède pas un successeur immédiat, on construit directement une exigence S qui complète le préfixe de taille 1 trouvé dans le paquet n+1.

5.4.6 Gestion de la liste des traces

Nous avons crée trois arbres de connexions pour superviser les flux TCP, UDP et ICMP (section 5.4.1). Ensuite, nous avons sauvegardé dans chaque contexte les paramètres de la connexion et incorporé une liste de traces afin d'assurer le filtrage à la volée. L'insertion dans la liste de traces a été effectuée en respectant l'ordre des paquets dans le flux à analyser. Ceci a assuré un transfert correct des exigences entre les traces voisines qui s'effectue toujours de l'ancienne trace vers la nouvelle. Il s'agit du sens naturel puisque une exigence décrit ce qu'il suffit de détecter

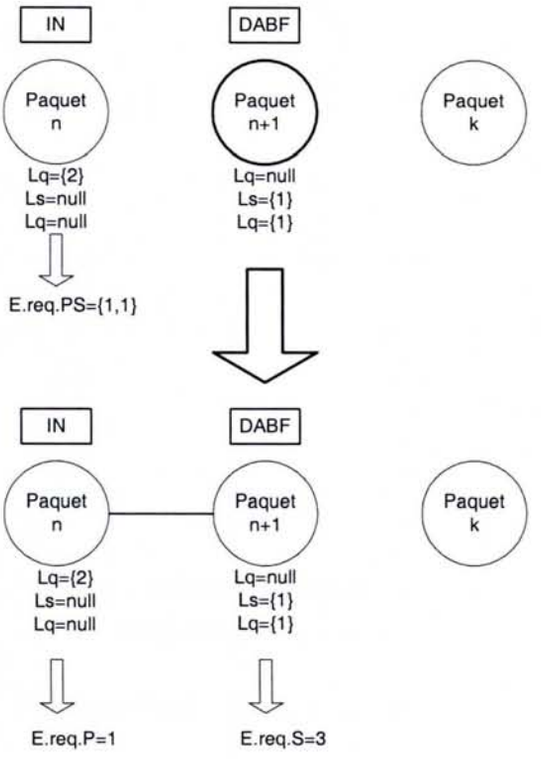


FIG. 5.12 – Mise à jour de la liste des exigences

dans les futures traces pour retrouver le motif. La corrélation des résultats de filtrage se réalise alors sur la liste des traces. Plus la taille de cette liste est minime, plus on accélère la recherche des motifs.

Nous éliminons dans cette étape toutes les traces dont les données sont devenues inutiles pour le reste de l'opération de filtrage. Les traces éliminées possèdent déjà deux traces immédiatement voisines de gauche et de droite. Ainsi elles n'ont plus d'exigences à transférer vers d'autres voisins. Nous appliquons ce principe juste après l'opération de filtrage et avant de traiter le paquet suivant. Nous obtenons de petites listes de traces ce qui accélère d'une part l'insertion des nouvelles traces. D'autre part, nous protégeons les ressources mémoires de l'IDS contre des attaques de déni de services.

Nous présentons dans la Figure 5.13 un scénario de gestion de la liste de traces. Nous supposons que le flux de données contient 6 paquets dont l'arrivée est totalement aléatoire. Les nouvelles traces sont représentées en gras alors que les variables N et P indiquent respectivement la présence d'un successeur et d'un prédécesseur immédiat de la trace courante. Dans ce scénario nous recevons successivement les paquets numérotés 1, 3 et 6. Nous calculons pour chacun, la liste des connaissances puis l'ensemble des exigences. L'arrivée du paquet numéroté 2 déclenche le premier transfert d'exigences des traces 1 et 3 vers la trace 2. Ensuite les traces 1 et 2 sont éliminées puisqu'elles ne disposent plus de nouveaux voisins alors que la trace 3 devient la tête de la liste. En recevant le paquet numéroté 4, nous transférons les exigences de la trace 3 vers la trace 4 puis nous éliminons la trace 3. Enfin le dernier paquet 5 engendre un transfert d'exigences des traces 4 et 6 vers la trace 5 puis l'élimination totale de la liste des traces. Durant ce scénario de traitement des paquets, nous constatons que la mise à jour continue de la liste des traces lui a assuré une taille réduite malgré l'arrivée totalement aléatoire des paquets.

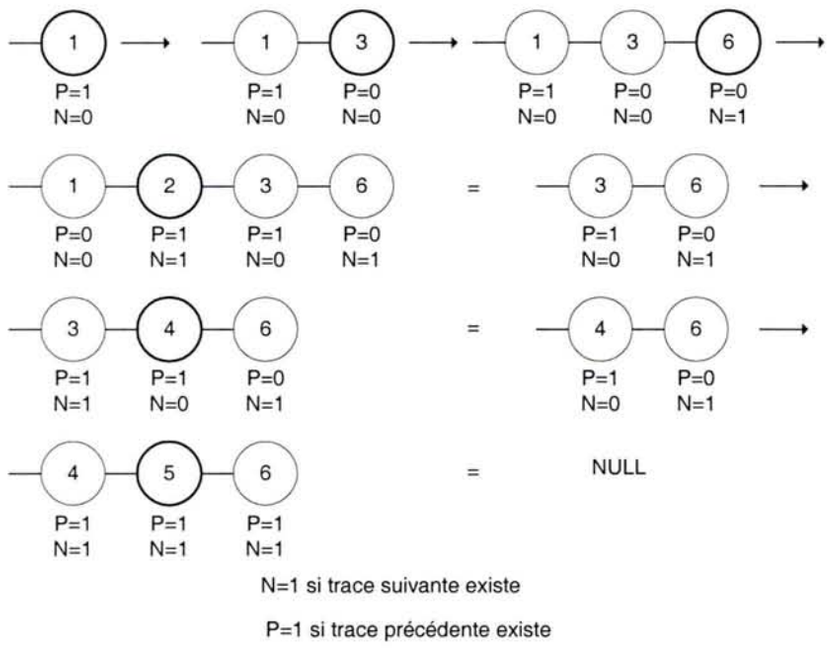


FIG. 5.13 – Gestion des listes de traces

La méthode de détection s'intéresse uniquement à l'analyse de la nouvelle trace créée après le transfert des exigences des traces voisines. La stratégie assure une recherche rapide des motifs d'attaque (une seule trace est inspectée par paquet) mais ne permet pas d'exprimer la conjonction de plusieurs exigences stockées dans des traces différentes. Ceci pose un problème puisqu'une exigence de type PS, traduisant la détection d'un sous mot du motif, peut se transformer en une conjonction de deux exigences P et S sauvegardées dans deux traces distinctes. Dans ces conditions, la vérification d'une des deux propriétés ne suffit pas pour affirmer la détection du motif.

Afin de résoudre ce problème nous avons introduit la notion d'"association". Il s'agit d'une variable partagée entre plusieurs exigences et qui sera libérée partiellement suite à la vérification de chaque exigence et définitivement si une seule règle détient encore l'association. De plus si l'une des exigences n'est pas vérifiée alors l'association est libérée ce qui entraîne l'élimination de toutes les exigences qui lui sont attachées.

5.5 Détection d'un seul motif d'attaque

Nous présentons dans Algorithme 6 le filtrage à la volée d'un seul motif d'attaque. La fonction Filtrer implémente la méthode de filtrage Boyer Moore Horspool tandis que LireDAWG permet de construire la liste des sous mots. Les deux fonctions Préfixe et Suffixe extraient respectivement les listes des préfixes et des suffixes du motif à partir des deux extrémités du paquet. Ensuite, la fonction Extraire_précédent (resp. Extraire_successeur) cherche s'il existe une trace qui précède (resp. devance) la trace du paquet courant. Enfin la fonction Traiter_précédent (resp. Traiter_successeur) implémente les règles de transfert d'exigences 1,2,3,4,9,11, 13 et 15 (resp. 5, 6, 7, 8, 10, 12, 14 et 16).

```
Phase de prétraitement
Construction du DAWG du motif
Construction du DAWG inversé du motif
n =| sig |
Phase de traitement
liste : liste des traces
pour chaque Paquet reçu faire
    Soit E la trace associée au dernier paquet reçu
    E.l =| paquet |
    si E.l ≥ n alors
        si Filtrer(paquet) alors "Motif trouvé"
    sinon
        E.l_q=LireDAWG(paquet)
    E.Lp=Préfixe(paquet)
    E.Ls=Suffixe(paquet)
    E.SN=paquet.sn //Numéro de séquence
    E.Offset=paquet.offset //décalage
    Insérer(liste,E)
    E_0 = Extraire_précédent(liste,E)
    E_1 =Extraire_successeur(liste,E)
    si E_0 == E_1 == NUL alors
        pour chaque élément e dans E.Lq faire
            si e==1 alors
                E.req.S=n-E.l
            sinon
                si e+E.l==n alors
                    E.req.P=n-E.l
                sinon
                    E.req.PS=(e,n-E.l-e)
        pour chaque élément e dans E.Lp faire
            E.req.S=n-e
        pour chaque élément e dans E.Ls faire
            E.req.P=n-e
    si E_0 ≠ NUL et E_1 == NUL alors
        Traiter_précédent(E_0,E)
    si E_1 ≠ NUL et E_0 == NUL alors
        Traiter_successeur(E_1,E)
    si E_0 ≠ NUL et E_1 ≠ NUL alors
        Traiter_précédent(E_0,E)
        Traiter_successeur(E_1,E)
```

Algorithme 6: Cas d'une seule signature d'attaque

5.6 Détection simultanée de plusieurs motifs d'attaques

Nous détaillons dans cette section la méthode de filtrage à la volée de plusieurs motifs. D'abord, nous modifions nos structures de données pour pouvoir contenir les résultats de filtrage de plusieurs signatures d'attaques. Nous définissions pour k motifs, un tableau de taille k

comportant les différentes listes de connaissances et les ensembles d'exigences par rapport aux k motifs recherchés. Les opérations de filtrage partiel (extraction des préfixes, suffixes et sous mots) s'effectuent toujours avec les même structures de données, c'est-à-dire le DAWG et le BDAWG des motifs. Ces structures tiennent compte de plusieurs signatures ce qui accélère la recherche en effectuant une seule lecture du contenu de chaque paquet et évite la construction séparée de plusieurs DAWG et BDAWG des motifs.

Par ailleurs nous modifions l'algorithme de filtrage complet des signatures d'attaques. L'utilisation de la méthode Boyer Moore Horspool n'est plus efficace dans le contexte de plusieurs motifs. Nous remplaçons cet algorithme par trois autres méthodes de filtrage multiple déjà expliquées dans la Section 5.2 : l'algorithme d'Aho-Corasick, le Forward DAWG et le MATCH DAWG. Nous comparons dans notre expérimentation les trois méthodes de filtrage. Nous constatons la supériorité de la méthode d'Aho-Corasick. Une analyse détaillée de ces résultats est présentée dans la Section 5.7.

L'Algorithme 7 montre le filtrage à la volée de plusieurs signatures d'attaques. Cet algorithme présente quelques différences par rapport au cas d'un seul motif d'attaque. En effet, la recherche des sous mots s'effectue si la taille du paquet est inférieure à celle du plus grand motif d'attaque. Ensuite, l'opération de filtrage multiple est réalisée si la taille du paquet est supérieure à celle du plus petit motif recherché. Nous présentons dans la suite l'algorithme complet du filtrage de plusieurs signatures d'attaques. Nous employons toujours sur les mêmes règles de transfert d'exigences présentées dans la sous-section 5.4.5.

5.7 Implémentation

Nous présentons dans cette section les différentes expériences réalisées pour évaluer la performance du filtrage multiple à la volée de plusieurs signatures d'attaques. Nous développons un module à part qui implémente les divers algorithmes de filtrage. Nous intégrons dans ce projet les méthodes de construction du DAWG, du DAWG inversé et de trie d'Aho-Corasick. De plus nous implantons les techniques de filtrage de Boyer Moore Horspool, d'Aho Corasick, de Forwad DAWG et de MATCH DAWG. Nous testons nos programmes pour retrouver plusieurs motifs dans un texte formé de plusieurs phrases désordonnées. Cette implémentation nous permet de valider la méthode avant de l'introduire dans le système de détection d'intrusions Snort.

La deuxième partie de l'implantation consiste à intégrer le filtrage à la volée dans Snort. Nous considérons la nouvelle chaîne de détection d'attaques que nous proposons au Chapitre 4 Section 4.4. Ainsi nous regroupons dans des OTNs primaires les ensembles de motifs nécessaires pour construire les DAWG, DAWG inversés et tries d'Aho-Corasick. Ces structures sont insérées dans la structure ds_list que comporte chaque nœud OTN. Notons que la structure ds_list sert à stocker les arguments des fonctions de traitement associées aux options des OTNs. En particulier, elle nous permet d'introduire les arguments à la fonction de filtrage à la volée de plusieurs motifs d'attaques.

Durant notre expérimentation, nous testons le filtrage sur des trafics simulés et des trafics réels (Voir Tableau 5.1). Le but du trafic simulé du log1 est de valider le bon fonctionnement des méthodes de filtrage dans Snort. Nous essayons de détecter des phrases que nous insérons dans des segments TCP et des paquets IP envoyées en désordre. Le log2 représente également un trafic simulé et comporte un grand nombre de fragments et de longues sessions TCP. Nous testons via ce procédé l'influence de ce type de trafic sur nos algorithmes. En particulier, nous évaluons la gestion des listes de traces. Enfin log3 et log4 comportent des trafics réels dont les caractéristiques sont représentées au Tableau 5.1.

```
Phase de prétraitement
k : Nombre des signatures
n : Table contenant les tailles des k signatures
n_min =| sig_min | ; n_max =| sig_max |
Construction du DAWG des k signatures
Construction du DAWG inversé des k signatures
Phase de traitement
liste : liste des traces
pour chaque paquet reçu faire
    Soit E la trace associée au dernier paquet reçu
    E.l =| paquet | ;
    si E.l ≥ n_max alors
        Tab_Match=Filtrer(paquet)
        pour i=1 à k faire
            si tab_match[i] ≠ ∅ alors signature i trouvée
    sinon
        si E.l ≥ n_min alors
            Tab_Match=Filtrer(paquet)
            pour i=1 à k faire
                si tab_match[i] ≠ ∅ alors signature i trouvée
            E.l_q=LireDAWG(paquet)
        sinon
            E.l_q=LireDAWG(paquet)
    E.Lp=Préfixe(paquet)
    E.Ls=Suffixe(paquet)
    E.SN=paquet.sn //Numéro de séquence
    E.Offset=paquet.offset //Décalage du paquet
    Insérer(liste,E)
    E_0 = Extraire_précédent(liste,E)
    E_1 =Extraire_successeur(liste,E)
    si E_0 == E_1 == NUL alors
        pour i=1 à k faire
            pour chaque élément e dans E.Lq[i] faire
                si e==1 alors
                    E.req[i].S=n[i]-E.l
                sinon
                    si e+E.l==n[i] alors
                        E.req[i].P=n[i]-E.l
                    sinon
                        E.req[i].PS=(e,n[i]-E.l-e)
            pour chaque élément e dans E.Lp[i] faire E.req[i].S=n[i]-e
            pour chaque élément e dans E.Ls[i] faire E.req[i].P=n[i]-e
    si E_0 ≠ NUL et E_1 == NUL alors
        Traiter_précédent(E_0,E)
    si E_1 ≠ NUL et E_0 == NUL alors
        Traiter_successeur(E_1,E)
    si E_0 ≠ NUL et E_1 ≠ NUL alors
        Traiter_précédent(E_0,E)
        Traiter_successeur(E_1,E)
```

Algorithme 7: Cas de k signatures d'attaques

	Type	Size (Mb)	TCP (%)	UDP (%)	ICMP (%)	Autres (%)	# paquets Fragmentés	# flux TCP
log 1	Sim.	2.10^{-3}	80	10	10	0	2	1
log 2	Sim.	1.2	50	25	25	0	2000	1000
log 3	Réel	102	90.1	6.7	3.1	0.1	0	1026
log 4	Réel	10^3	75.3	17.8	6.7	0.2	46	18460

TAB. 5.1 – Caractéristiques du trafic

Nous présentons dans le Tableau 5.2 les différents résultats obtenus suite à l'analyse des quatre fichiers log. Snort 1.9 exécute l'algorithme de filtrage d'Aho-Corasick alors que notre version met en œuvre trois algorithmes de filtrage différents : le FDAWG, l'Aho-Corasick et le Match DAWG. Nous remarquons que le filtrage à la volée est possible ce qui évite le réassemblage des flux TCP et des fragments IP. De plus les temps d'exécution sont meilleurs avec l'algorithme d'Aho-Corasick et ceci avec les versions 1.9 et modifiée de Snort. Néanmoins avec un trafic trop fragmenté ou contenant plusieurs segments TCP (donc de longues sessions TCP), le filtrage à la volée est plus performant quelque soit l'algorithme de recherche employé. En effet, en inspectant le contenu du log2 de taille 1.2 Méga-Octet, le temps d'exécution de notre méthode est 0.1 seconde alors que celui de Snort 1.9 s'élève à 2.4 seconde. Pour confirmer ce résultat nous varions le nombre de segments et de fragments et nous calculons le temps nécessaire pour analyser du trafic. La Figure 5.14 montre l'effet négatif du réassemblage à alourdir le système de détection d'intrusions Snort 1.9.

	Snort 1.9		Snort Modifié			
	Alert	Temps(s)	Alert	FDAWG(s)	AC(s)	MDAWG(s)
log1	2	0.1	3	0.1	**0.1**	0.1
log2	0	2.4	1	**0.11**	**0.11**	**0.12**
log3	169	9.17	170	9.57	**7.62**	9.51
log 4	2974	89.76	3001	100.29	**83.9**	99.79

TAB. 5.2 – Résultats des expérimentations

Nos expériences montrent la supériorité de l'algorithme d'Aho-Corasick sur les autres méthodes de filtrage multiple et en particulier celle du Match DAWG. Cela est dû d'une part à la taille réduite des motifs, et d'autre part au large alphabet utilisé. En effet, malgré que la méthode de filtrage Match DAWG présente une complexité supérieure en lisant deux fois le contenu du paquet, elle donne en pratique des résultats meilleurs lorsqu'elle est appliquée sur un alphabet petit. Son utilisation est donc intéressante en bioinformatique pour retrouver des séquences ADN. La rapidité de la recherche s'explique par des sauts plus importants calculés suite à la recherche des suffixes (via le DAWG inversé) et la recherche des préfixes (via l'arbre d'Aho-Corasick). Néanmoins, l'alphabet utilisé en analysant le trafic réseau comporte des caractères alphanumériques et des caractères spéciaux. De plus la recherche sensitive en respectant la casse des caractères augmente la taille de l'alphabet. Par conséquent, des sauts calculés sur la base des préfixes via la méthode d'Aho-Corasick suffisent pour accélérer la recherche. On évite ainsi l'utilisation du DAWG inversé et par suite une deuxième lecture de droite à gauche du contenu de chaque paquet reçu.

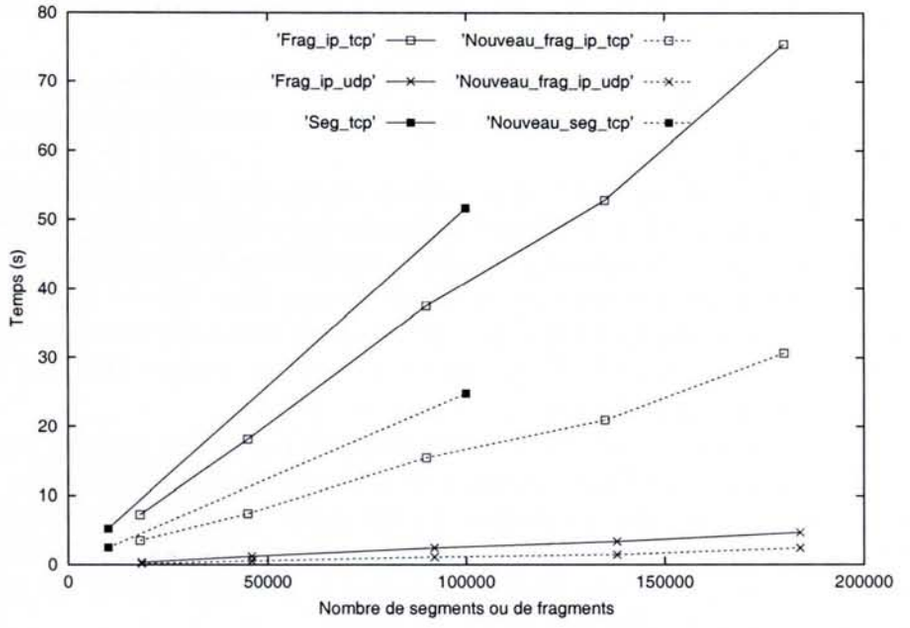


FIG. 5.14 – Effet de la fragmentation/segmentation sur le filtrage du trafic

Par ailleurs, la méthode de filtrage d'Aho-Corasick est plus rapide que celle du Forward DAWG. Celle supériorité est due aux sauts importants définis par Aho-Corasick en se basant sur la recherche des préfixes. Quant à la méthode du Forward DAWG, elle utilise le DAWG des motifs qui tient compte de tous les sous mots de motifs et non seulement des préfixes ce qui se traduit par des sauts plus petits. Néanmoins, le DAWG reste notre principale structure pour identifier les préfixes des paquets qui sont des suffixes des motifs (via le DAWG) et les suffixes des paquets qui sont des préfixes des motifs (via le BDAWG). Par conséquent, la combinaison DAWG, Aho-Corasick constitue notre meilleure stratégie pour assurer un filtrage à la volée des signatures d'attaques.

5.8 Conclusion

Nous avons proposé dans ce chapitre une méthode de filtrage à la volée du trafic réseau qui évite le rassemblement du trafic. La technique cherche simultanément plusieurs motifs d'attaques en tenant compte d'une arrivée désordonnée des paquets IP. De plus elle utilise les contextes de connexions pour superviser convenablement l'activité réseau. L'expérimentation portée sur des trafics simulés et réels, montre l'efficacité de la méthode surtout lorsqu'elle se base sur l'algorithme de filtrage multiple d'Aho-Corasick.

Par conséquent, la détection d'intrusions avec le filtrage exact des signatures d'attaques permet de vérifier rapidement la présence des motifs dans le contenu des paquets. Cette méthode de détection traite le contenu du paquet comme une simple chaîne de caractères ce qui épargne l'IDS d'une phase d'identification puis de décodage protocolaire. Ceci permet de concevoir des IDS légers qui sont plus efficaces si le nombre total des signatures d'attaques est petit. De plus la méthode suppose une présence spécifique du motif dans le scénario d'attaque afin d'éviter la génération des faux positifs. Malheureusement, ces hypothèses ne sont pas toujours vérifiées ce qui nous mène aux lacunes suivantes :

- le nombre de signatures peut être important même après une sélection des règles candidates qui satisfont les premiers tests portés sur le trafic,
- l'abstraction du niveau applicatif sous forme d'une chaîne de caractères conduit à des faux positifs si le filtrage des motifs s'effectue dans des zones de texte différentes de celles prévues par les signatures d'attaques,
- le filtrage de motifs s'effectue sur tout le contenu du paquet ce qui représente parfois un large espace de recherche. Les IDS peuvent limiter cette zone mais ils risquent les faux négatifs si les attaquants réussissent à décaler adéquatement les signatures d'attaques,
- la détection d'intrusions avec filtrage de motifs ne considère pas le contexte applicatif des données filtrées ce qui déclenche des opérations de recherche inutiles. Nous prenons l'exemple d'un IDS qui essaie de filtrer des commandes du service TELNET bien que la première phase d'authentification ne soit pas encore achevée,
- avec un simple filtrage de motifs, l'IDS ne supervise pas l'exécution normale des protocoles applicatifs ainsi que les contraintes imposées dans les spécifications RFC de ces protocoles. Par exemple, il est impossible de localiser les bits et les champs réservés des protocoles applicatifs, de superviser les échanges clients/serveurs ou de vérifier les noms de domaines contenus dans les enregistrements DNS (par exemple le RFC2181 impose que les tailles des noms de domaines soient inférieures à 255 caractères).
- un motif d'attaque présente plusieurs formes possibles. Par exemple les adresses URI peuvent être en ascii, en hexadécimale ou en unicode. Par ailleurs, le motif "site exec" considéré comme un motif d'attaque sur le service FTP, possède la même sémantique si la commande "site" de FTP est séparée par un séparateur valide (une ou plusieurs tabulations ou espaces). Ainsi utiliser une forme unique du motif s'avère insuffisante et prévoir toutes les formes possibles est une tâche impossible.

Par conséquent la détection d'intrusions peut se passer d'effectuer l'analyse protocolaire pour comprendre la nature du trafic, contrôler l'exécution des protocoles et optimiser les opérations de filtrage. Nous étudions ce type de détection dans le Chapitre 6 puis nous l'appliquons sur un ensemble de protocoles applicatifs fréquemment utilisés dans les communications réseaux.

6

Analyse protocolaire par arbres de décisions

Sommaire

6.1 Introduction

Plusieurs systèmes de détection d'intrusions utilisent la recherche de motifs pour filtrer le trafic réseaux ou localiser les séquences des activités anormales dans les fichiers journaux de sécurité. Appliquées sur un trafic réseau, les opérations de filtrage permettent de retrouver la position exacte des motifs d'attaques. Certains IDS comme Snort restreignent leurs analyses protocolaires au niveau transport puis utilisent le filtrage pour chercher les signatures dans le contenu du protocole du transport. La méthode est rapide puisqu'elle ne requiert pas la compréhension du niveau applicatif. Elle est intéressante pour filtrer rapidement quelques classes de trafic. Mais d'autres services réseaux exigent une analyse plus détaillée afin d'éviter toute forme d'évasion et détecter les anomalies dans les exécutions des protocoles. Il s'agit donc de réaliser l'analyse protocolaire afin de superviser les applications réseaux.

Nous nous intéressons dans ce chapitre à l'analyse protocolaire. Cette méthode de détection est précédée par une phase de décodage pour extraire les champs du (des) protocole(s) à analyser. Ayant compris la composition du trafic, nous nous basons sur la spécification du protocole pour vérifier sa bonne exécution d'une part et d'autre part détecter les scénarios des attaques connues. Ces opérations s'appuient généralement sur un ensemble de tests à effectuer sur les champs du protocole. L'ordre des tests dépend des caractéristiques du trafic et de l'ensemble des scénarios d'attaques recherchées. De plus il intervient pour conclure rapidement la présence des attaques. Nous utilisons dans ce chapitre la théorie d'information pour définir la bonne séquence des tests et par suite mieux analyser le trafic.

La Section 6.2 de ce chapitre explique les avantages de l'analyse protocolaire et son apport pour remédier aux lacunes du filtrage brut des motifs d'attaques soulignées à la fin du Chapitre 5. Ensuite nous présentons dans la Section 6.3 notre langage pour spécifier les attaques sur les protocoles applicatifs. La Section 6.4 construit un arbre de décision qui implémente la procédure de vérification du trafic. Ensuite nous présentons dans la Section 6.5 les deux étapes de prétraitement et de traitement du trafic nécessaires pour détecter les intrusions. Enfin, la Section 6.6 applique la méthode sur un ensemble de protocoles applicatifs fréquemment utilisés dans les communications réseaux.

6.2 Intérêt de l'analyse protocolaire

L'analyse protocolaire supervise la bonne exécution des protocoles. Elle permet de comprendre le contenu du trafic pour mener une recherche plus ciblée des signatures d'attaques. Les conséquences immédiates d'un tel procédé sont de vaincre les tentatives d'évasion, réduire l'espace de recherche des signatures et diminuer considérablement les fausses alertes. Nous détaillons dans la suite ces divers avantages à travers des exemples concrets.

- **Protection contre l'évasion :** les attaquants essaient de contourner les IDS en évitant les signatures d'attaques. Par exemple au lieu d'entrer directement la commande "site exec" de FTP ou "vrfy root" de SMTP, ils peuvent séparer les commandes et les arguments par des séparateurs valides (plusieurs espaces et tabulations en ascci, hexadécimal et unicode). L'analyse protocolaire permet d'extraire chaque champs du protocole, de normaliser son contenu (élimination de l'encodage hexadécimale et unicode) puis de confronter les commandes aux arguments. L'opération de filtrage des deux signatures présentées ci-dessus se transforme alors en un problème d'identification des commandes utilisées durant l'exécution des services réseaux correspondants.
- **Réduction de l'espace de recherche :** l'analyse protocolaire extrait les champs utiles du protocole pour mener ensuite la détection d'intrusions. Elle permet ainsi de réduire l'espace de recherche des signatures d'attaques. Par exemple au lieu de chercher des scripts CGI et PHP dans le contenu des segments TCP, on peut se restreindre à l'URI des requêtes HTTP. Notons que l'opération de filtrage est toujours nécessaire mais elle s'applique dans des zones réduites du paquet. De plus, si l'opération de filtrage devient plus rapide, une étape de décodage est nécessaire pour extraire les champs utiles du protocole.
- **Efficacité de la détection :** la diminution de l'espace de recherche contribue à réduire le nombre de faux positifs. En effet, une alerte n'est déclenchée que si le motif d'attaque a été trouvé dans un champs précis du protocole. De plus le processus de détection cherche des signatures d'attaques plus exactes et ceci en remplaçant quelques opérations de filtrage par des inspections directes du contenu des champs du protocole. Par exemple, la signature Snort Sid1201 détecte via le motif "HTTP/1.1 403" les tentatives d'accès non autorisées

à des pages web sécurisées. Sachant que les serveurs HTTP émettent des réponses sur 3 digits pour toute requête HTTP, l'intrusion peut être détectée en identifiant la réponse 403 du serveur. Ceci évite le déclenchement des fausses alertes si le motif "HTTP/1.1 403" existe dans une autre partie de la réponse HTTP. Il empêche également les faux négatifs si le serveur web émet la réponse "HTTP/1.0 403".

- **Normalisation du trafic :** la normalisation du trafic au niveau IDS s'avère intéressante pour éviter les tentatives d'évasion. Cependant elle exige la connaissance des caractéristiques du protocole utilisé. Par exemple, certains protocoles tel que TELNET envoient leurs données en mode caractère autorisant ainsi l'utilisation des caractères de mise en forme comme l'effacement des caractères et l'effacement des lignes (roll→→ot = root). La négociation de ces options s'effectue durant les premiers échanges entre le client et le serveur et se poursuit ensuite durant la transmission des données. Ainsi, une analyse protocolaire au niveau IDS permet de superviser ces échanges et par suite de connaître la méthode adéquate pour normaliser le trafic avant son traitement.
- **Archivage plus détaillé :** la plupart des protocoles réseaux intègrent une phase d'authentification avant d'échanger les données entre clients et serveurs. Au cours de cette étape, l'analyse protocolaire permet de sauvegarder des informations intéressantes comme les identifiants des utilisateurs. Ensuite, si un scénario d'attaque est détecté alors les entrées des fichiers de sécurité seront enrichies par ces informations utiles. La détection d'intrusions avec filtrage de motif trouvera la signature d'attaque mais sera incapable de fournir des informations supplémentaires.
- **Contrôle des exécutions des protocoles :** l'analyse protocolaire permet de détecter les failles dans les implémentations des protocoles réseaux et de repérer facilement les paquets forgés. En effet, l'analyse des données révèle parfois des violations des recommandations RFC comme l'utilisation des bits réservés ou des incohérences dans les champs des protocoles. Par exemple, afin de réduire la quantité de données transmises, le protocole DNS référence via des pointeurs, les noms de domaines déjà évoqués dans la trame. Ces pointeurs doivent référencer des champs corrects pour ne pas perturber le fonctionnement du client DNS. Les contraintes imposées dans les RFC ne concernent pas uniquement une trame de données mais également les échanges clients/serveurs. L'exécution d'un protocole applicatif suit généralement une séquence de phases bien déterminée. Par exemple, un client FTP doit recevoir la notification de son succès d'authentification avant d'envoyer des commandes de transfert de fichiers. Par ailleurs, un client DNS ne reçoit pas de réponses de résolution de noms de domaine tant qu'il n'a pas émis de requêtes. Notons que ces types d'attaques ne sont pas détectés par le filtrage de motifs étant donné l'absence de signatures d'attaques. Les données sont en fait cohérentes mais leurs modes de transmission ne le sont pas.

Par conséquent l'analyse protocolaire constitue une tâche un peu lourde à cause du décodage des protocoles applicatifs. Cependant, elle offre des opportunités de détection supplémentaires ce qui rend son utilisation intéressante. De plus, étant donnée que la vérification des champs des protocoles est plus rapide, le temps mis durant la phase de décodage est aussitôt compensé quand les opérations de filtrage sont exécutées plusieurs fois par de nombreuses règles de détection d'attaques.

Les systèmes de détection d'intrusions ont intégré progressivement l'analyse protocolaire dans leurs supervisions du trafic réseau. Graham [61] a souligné qu'en 1996 NetRanger décode 10 protocoles alors que RealSecure en analyse 25. Dragon et Snort ne supportent en 1999 que 5 protocoles alors que Network ICE supervise jusqu'à 70 protocoles. Ensuite et à partir de 2002 divers fournisseurs de systèmes de détection d'intrusions ont augmenté la pile des protocoles

traités. Par exemple, Intervert, Symentec et Netscreen analysent 25 protocoles. Par ailleurs, le nouveau système de détection d'intrusions RealSecure v7 d'ISS supervise 100 protocoles. NFR v3 supporte aussi un grand nombre de protocoles alors que Snort 2.1 ajoute de nouveaux plugins pour décoder quelques protocoles fréquemment utilisés. Notons que la nouvelle version de Snort présente quelques lacunes car malgré le décodage partiel de quelques protocoles tels que HTTP, TELNET et FTP, elle n'assure pas une analyse avec sauvegarde d'états des exécutions des protocoles. Actuellement, le principal objectif du décodage dans Snort est d'extraire puis de normaliser quelques champs des protocoles afin d'éviter les évasions lors des opérations de filtrage. Bro [128] analyse plutôt les activités durant les sessions de protocoles. Il assure alors une analyse protocolaire détaillée. De plus, il permet d'enregistrer des sessions HTTP, FTP et DNS dans des fichiers historiques. Si une attaque est détectée alors l'inspection de ces fichiers fournit des détails sur la session correspondante.

Nous présentons dans la suite notre procédé pour mener l'analyse protocolaire. Nous nous intéressons d'abord à la spécification des attaques sur les protocoles applicatifs. Cet description nous sert de base pour construire des arbres de décisions. Ensuite nous détaillons la phase de détection qui se déroule en deux étapes. Une première étape de prétraitement qui assure entre autres le décodage et la normalisation du trafic. Puis une deuxième étape de parcours des arbres de décisions pour détecter des scénarios d'attaques.

6.3 Langage de description des attaques

Afin de mener une analyse protocolaire, nous devons spécifier convenablement les attaques sur les protocoles à étudier. Nous définissons dans cette section un langage générique pour décrire les scénarios d'attaques. Le schéma de ces attaques est en fait exprimé via deux types de règles : des règles de détection et des règles d'actions. Les règles de détection effectuent une analyse en se basant sur le contenu d'un seul paquet alors que les règles d'actions étendent cette vérification sur plusieurs trames. Notons que les règles décrivent uniquement les comportements anormaux et que les contraintes imposées par les protocoles comme les bits réservés seront directement vérifiées durant une phase de prétraitement du trafic. Bien que ces contraintes puissent être exprimées via des règles de détection, nous optons pour cette solution afin de simplifier les règles et par suite garantir un arbre de décision minimal.

La spécification des scénarios d'attaques sur un protocole applicatif se décompose en trois parties (voir BNF). D'abord nous définissons les constantes et variables globales nécessaires pour écrire les règles d'attaques. La deuxième partie de la spécification décrit les règles de détection. Une règle de détection peut être vue comme un ensemble de tests à effectuer sur les champs du protocole et les variables définies dans la première partie de la spécification. Les tests sont des comparaisons logiques ou des opérations de filtrage sur un champs précis du protocole qu'on appelle paramètre. La vérification d'une règle permet de mettre à jour les variables globales ou d'émettre directement un message d'alerte si une attaque est détectée. Notons que ces règles seront traduites sous forme d'un arbre de décision afin d'ordonner efficacement l'ensemble des tests à réaliser.

La troisième partie de la spécification définit les règles d'actions. Ces règles décrivent le comportement de l'IDS après la vérification des règles de détection c'est-à-dire à la fin du parcours de l'arbre de décision. Les règles de détection sont précédées par des pré-conditions, qui si elles sont vérifiées permettent de réaliser 4 types d'actions. Les pré-conditions portent sur les variables globales de la spécification. Ensuite les actions peuvent être "stop", "continue", "update" et "log". L'action "stop" arrête l'analyse du paquet courant alors que l'action "continue" analyse le

prochain protocole applicatif empilé dans le contenu du paquet. L'action "update" met à jour les variables globales de la spécification et sauvegarde des informations intéressantes sur la session en cours du protocole. Enfin, l'action "log" déclenche une alarme suite à la détection d'une attaque qui s'est déroulée en plusieurs étapes.

Nous présentons dans la suite le langage de spécification des scénarios d'attaques puis nous montrons un extrait de la spécification des attaques sur le protocole FTP.

BNF du langage

#Partie1 : Définition des constantes et des variables
'VAR' corps1
corps1 := nomVar valeurVar ;
valeurVar := valeur | valeur::valeurVar

#Partie2 : Règles de détection
'RULE' corps2 ;
corps2 := condition | condition $\wedge$ coprs2
condition := paramètre opérateur terme
paramètre : un champs du protocole
opérateur := contient | égale | dans | inf | sup | $\neg$ opérateur | ...
terme := valeur | valeur::terme | nomVar | nomVar::terme

#Partie3 : Règles d'actions
'BHV' corps3
corps3 := condition '$\Rightarrow$' action argument ;
condition : expression booléenne
action := update | log | exit | continue

r_1 : RULE 1 direc = to_server, cmd = site, arg = "exec", msg : Site Exec Attempt
r_2 : RULE 2 direc = to_server, cmd = pass, arg contain "hack", msg : Suspect Password
r_3 : RULE 3 direc = to_server, cmd = help, argSize > 900, msg : Buffer Overflow Attempt

TAB. 6.1 – Extrait de la spécification des attaques sur le protocole FTP

6.4 Construction de l'arbre de décision

L'analyse protocolaire effectue une suite de tests sur les champs des protocoles applicatifs. Notre but est d'optimiser ces opérations de tests. Dans le Chapitre 4, l'organisation des options dans des structures OTNs concerne principalement des protocoles de la couche Internet (IP, ICMP) et Transport (TCP, UDP). Ces protocoles sont communs à plusieurs applications Internet et donc nous vérifions le contenu des mêmes champs via des structures uniques (OTN). Néanmoins, en détaillant l'analyse au niveau applicatif, chaque protocole présente ses propres champs. Nous optons alors pour une représentation séparée de l'ensemble des tests. Cette opération est réalisée via des arbres de décisions.

La construction de l'arbre de décision est assurée par un système d'inférence. On définit trois règles d'inférences qui prennent en entrée un nœud de l'arbre et génèrent ensuite d'autres nœuds fils. Une seule règle s'applique à chaque fois et cela jusqu'à aboutir à des feuilles de l'arbre. De plus le choix du test à réaliser sur chaque nœud dépend des caractéristiques du trafic. Nous présentons d'abord notre système d'inférence puis nous expliquons les critères de choix du test à effectuer sur un nœud de l'arbre.

6.4.1 Système d'inférence

Avant d'expliquer le système d'inférence, nous présentons la structure "nœud" de l'arbre de décision et nous définissons quelques fonctions utiles utilisées dans les prémisses des règles d'inférences.

Un nœud de l'arbre intègre principalement un test à effectuer sur un champs particulier du protocole. Nous représentons la structure nœud comme un enregistrement $\langle c, \mathcal{R}, \mathcal{F}, \mathcal{L} \rangle$ avec c une condition, $\mathcal{R}$ l'ensemble des règles de détection candidates, $\mathcal{F}$ l'ensemble des paramètres testés jusqu'au nœud courant et $\mathcal{L}$ l'ensemble des règles déjà vérifiées. Rappelons qu'un paramètre est un champs du protocole, qu'une condition est un triplet ⟨ paramètre, opérateur, terme ⟩ et qu'un terme représente une valeur ou une liste de valeurs (Voir BNF).

Exemple : *le nœud racine de l'arbre de décision possède une condition vide, un ensemble $\mathcal{R}$ contenant toutes les règles de détection et deux ensembles vides $\mathcal{F}$ et $\mathcal{L}$.*

L'arbre de décision comporte 4 types de nœuds : les nœuds ordinaires, les nœuds spéciaux, les feuilles positives et les feuilles négatives.

nœud ordinaire : c'est un simple nœud dans l'arbre de décision, dont le test ne permet pas de vérifier une règle candidate ou d'éliminer l'ensemble $\mathcal{R}$ de toutes les règles candidates,

nœud spécial : c'est un nœud dont le test permet de vérifier uniquement une partie des règles candidates de l'ensemble $\mathcal{R}$,

feuille positive : il s'agit d'un nœud final dans l'arbre de décision avec un ensemble de règles vérifiées $\mathcal{L}$ non vide,

feuille négative : c'est un nœud final de l'arbre de décision dont l'ensemble de règles vérifiées $\mathcal{L}$ est vide.

Dans la suite, on note par P l'ensemble des paramètres et par ξ_i un opérateur possible sur ces paramètres. De plus, e désigne le paramètre choisi pour réaliser le prochain test, v une valeur possible de e et c une condition qui traduit un test à effectuer sur un nœud. La première définition présente la fonction *Param* qui extrait les paramètres d'une règle ou d'un ensemble de règles.

Définition 6.4.1 *Soient r une règle de la forme $r \equiv e_1\ \xi_1\ v_1 \wedge ... \wedge e_k\ \xi_k\ v_k$ et $\mathcal{R}$ un ensemble de règles. La fonction Param est définie par $Param(r) = \{e_1, ..., e_k\}$. On étend cette fonction sur l'ensemble des règles $\mathcal{R}$ par $Param(\mathcal{R}) = \bigcup_{r \in \mathcal{R}} Param(r)$*

Exemple 6.4.1 *Soit $R = \{r_1, r_2, r_3\}$ un ensemble de règles définies dans la spécification du protocole* FTP *et présentées dans le Tableau 6.1. Alors Param(r_1)=Param(r_2)={direc, cmd, arg} ; Param(r_3)={direc, cmd, argSize} ; Param(R)={direc, cmd, arg, argSize}.*

On définit maintenant la fonction *Proj*, qui étant donnés un ensemble de règles de détection et un paramètre, extrait toutes les valeurs possibles de ce paramètre.

Définition 6.4.2 *Soient $\mathcal{R}$ un ensemble de règles et e un paramètre. On définit la fonction de projection Proj par* $Proj(R, e) = \{v \mid \exists\ \xi_i,\ r\ tq\ ((e\ \xi_i\ v) \wedge r) \in \mathcal{R}\}$.

Exemple 6.4.2 $Proj(\{r_1, r_2, r_3\}, direc) = to\_server$.

La définition 6.4.3 décrit la fonction *Cand* qui, à partir d'un ensemble de règles $\mathcal{R}$ et d'une condition c, retourne toutes les règles vérifiant cette condition.

Définition 6.4.3 *Soient $\mathcal{R}$ un ensemble de règles et c une condition. On définit la fonction Cand par* $Cand(\mathcal{R}, c) = \{r \in \mathcal{R} \mid r \equiv c \wedge r'\}$.

Exemple 6.4.3 $Cand(\{r_1, r_2, r_3\}, direc = to\_server) = \{r_1, r_2, r_3\}$.

On note par *suivant* la fonction qui sélectionne le prochain paramètre à tester sur les futurs nœuds. Afin d'assurer la rapidité de la détection, le choix du paramètre ne doit pas être arbitraire mais dépend des caractéristiques du trafic. Pour ce faire, la fonction *suivant* applique un des critères de sélection des paramètres tels que le gain d'information (qui réduit l'entropie –désordre–), la proportion du gain d'information (qui normalise le gain d'information) et l'index de Gini (qui réduit le taux d'erreur). Nous présentons ces critères dans la sous-section 6.4.2. A présent, nous nous intéressons à la dernière fonction utilisée dans le système d'inférence et elle s'agit de la fonction *fils*. Cette fonction se sert de la fonction *suivant* pour déterminer les nœuds fils d'un nœud N.

Définition 6.4.4 *Soit $N = (c, \mathcal{R}, \mathcal{F}, \mathcal{L})$ un nœud de l'arbre tel que $\forall r \in \mathcal{R}\ :\ Param(r) \not\subseteq \mathcal{F}$. La fonction fils appliquée à N retourne les nœuds fils de ce nœud :*

$$\left\{ \begin{array}{l} (e\ \xi_1\ v_1, \mathcal{R}_1, \mathcal{F} \cup \{e\}, \mathcal{L}); ...; (e\ \xi_m\ v_m, \mathcal{R}_m, \mathcal{F} \cup \{e\}, \mathcal{L}); \\ \\ (\neg(e\ \xi_1\ v_1 \vee ... \vee e\ \xi_m\ v_m), \mathcal{R}_{m+1}, \mathcal{F} \cup \{e\}, \mathcal{L}) \end{array} \right\}\ avec :$$

- $e = suivant(c, \mathcal{R}, \mathcal{F})$
- $\{v_1, ...v_m\} = Proj(\mathcal{R}, e)$
- $\forall\ i \in [1..m]\ \mathcal{R}_i = Cand(\mathcal{R}, e\ \xi_i\ v_i)$
- $\mathcal{R}_{m+1} = \{r \in \mathcal{R} \mid e \notin Param(r)\}$

On note par $fils_i$ ($1 \leq i \leq m$) le nœud $(e\ \xi_i\ v_i, \mathcal{R}_i, \mathcal{F} \cup \{e\}, \mathcal{L})$ et $fils_{m+1}$ le dernier nœud $(\neg(e\ \xi_1\ v_1 \vee ... \vee e\ \xi_m\ v_m), \mathcal{R}_{m+1}, \mathcal{F} \cup \{e\}, \mathcal{L})$.

Exemple 6.4.4 $fils(racine) = fils((\emptyset, \{r_1, r_2, r_3\}, \emptyset, \emptyset)) =$
$(direc = to\_server, \{r_1, r_2, r_3\}, \{direc\}, \emptyset); (\ not(direc = to\_server), \emptyset, \{direc\}, \emptyset)$.

Nous présentons dans Figure 6.1 notre système d'inférence. Nous disposons de 3 règles d'inférences qui, en s'appliquant successivement sur des nœuds, permettent la construction de l'arbre. En effet :

la règle *décomposer :* elle permet de construire l'arbre en ajoutant de nouveaux nœuds fils au nœud sur lequel la règle s'applique. On vérifie d'abord que les règles candidates du nœud possèdent au moins un paramètre non encore testé et donc l'ensemble $\mathcal{R}$ des règles candidates n'est pas encore vérifié.

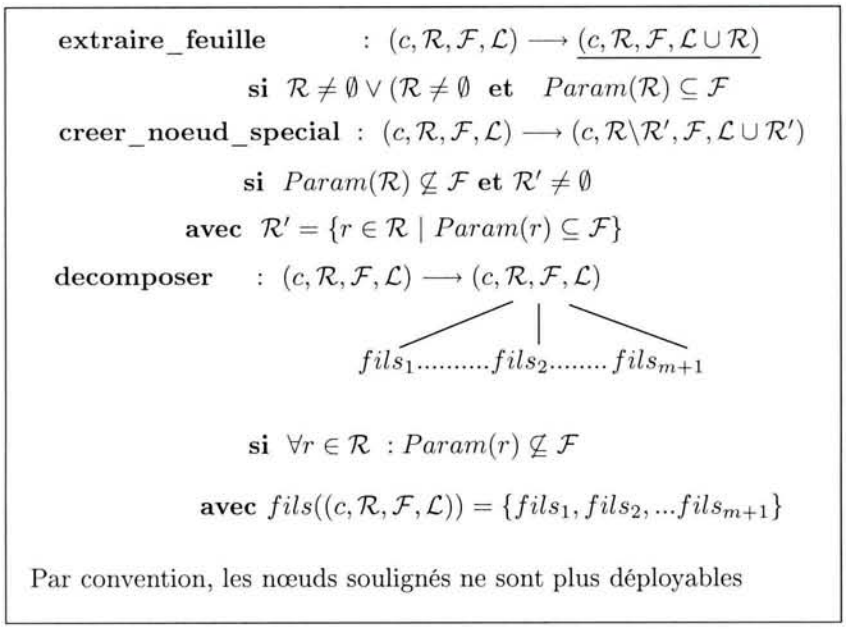


FIG. 6.1 – Système d'inférence

la règle *créer_noeud_spécial :* cette règle transforme en un nœud équivalent, un nœud dont seulement une partie de ses règles candidates ont été vérifiées. Le nouveau nœud extrait les règles satisfaites de l'ensemble des règles candidates $\mathcal{R}$ et les ajoute à l'ensemble des règles vérifiées $\mathcal{L}$.

la règle *extraire_feuille :* cette règle crée une feuille à partir d'un nœud ordinaire. Elle s'applique si tous les paramètres des règles candidates du nœud ont été sélectionnés ou s'il n'existe plus de règles candidates dans l'ensemble $\mathcal{R}$. La feuille est dite positive si l'ensemble $\mathcal{L}$ est non vide et elle est qualifiée négative dans le cas contraire.

Par conséquent toutes les règles transforment un nœud de l'arbre en un autre nœud jusqu'à aboutir à des feuilles. On utilise principalement les deux règles *décomposer* et *créer_noeud_spécial* pour déployer l'arbre jusqu'à aboutir à des situations de vérification ou d'élimination totale des règles candidates. A ce stade, on emploie la règle *extraire_feuille.*

6.4.2 Sélection des paramètres

La sélection du prochain paramètre pour déployer l'arbre de décision constitue un facteur intéressant pour détecter rapidement les attaques sur les protocoles applicatifs. Supposons qu'on dispose d'une règle de détection qui décrit une attaque fréquemment lancée sur le réseau. Si la règle possède un paramètre qui la distingue par rapport aux autres règles alors il serait intéressant de commencer la vérification par ce paramètre. Inversement, si les attaques sur un protocole particulier sont trop rares et qu'il existe un paramètre commun à toutes les règles de détection alors il vaut mieux commencer par ce test afin d'abandonner vite la recherche.

Afin d'optimiser la sélection des paramètres, nous appliquons des méthodes de la théorie d'information à savoir le gain d'information, la proportion du gain d'information et l'index de Gini. Notons que Kruegel et Toth [83] utilisent le même procédé pour classer les règles de détection d'attaques (voir Chapitre 4, sous-section 4.2.3.1). Ils attribuent les mêmes priorités aux règles

(cas équiprobable) puis rangent les règles en des petits groupes. Notre problème est différent puisque nous essayons de vérifier les règles et non pas de les regrouper. En fait, nous utilisons les valeurs possibles des paramètres des règles pour former deux groupes : la classe des "attaques" et la classe de "non attaque". Pour ce faire, on tient compte de l'apparition des paramètres dans un trafic échantillon étiqueté par les deux classes. On en déduit ensuite l'ordre convenable pour aboutir rapidement à l'une des deux classes des "attaques" ou de "non attaque".

6.4.2.1 Gain d'information

Le premier critère utilisé pour sélectionner les paramètres est le gain d'information. Afin d'expliquer cette méthode, on introduit d'abord la notion d'entropie. L'entropie est en fait une mesure qui indique l'impureté ou le désordre d'un ensemble d'exemples par rapport à une classification (Formule 6.1).

$$Entropie(N) = -(p_1 log_2 p_1 + p_2 log_2 p_2) \tag{6.1}$$

avec p_1=probabilité(attaque) et p_2=probabilité(non attaque).

La gain d'information par rapport à un paramètre A est la réduction d'entropie obtenue en choisissant ce paramètre pour partitionner les données. Elle est égale donc à l'entropie initiale des données moins la moyenne proportionnelle de l'entropie après la partition (Formule 6.2).

$$Gain(N, A) = Entropie(N) - \sum_{v \in val(A)} \frac{\mid N_v \mid}{\mid N \mid} Entropie(N_v) \tag{6.2}$$

Avec N_v est un sous ensemble de N dont la valeur du paramètre A est égale à v.

Notre but consiste à maximiser le gain d'information en choisissant le meilleur paramètre pour déployer l'arbre de décision.

6.4.2.2 Proportion du gain d'information

Le gain d'information présente l'inconvénient de préférer les paramètres ayant beaucoup de valeurs distinctes et qui partagent les données en de nombreux petits sous ensembles purs. Afin de remédier à cette lacune, Quinlan introduit la proportion du gain d'information [134]. Il s'agit d'une alternative qui divise le gain d'information par une mesure proportionnelle à la taille de la partition générée par les valeurs du paramètre choisi.

$$SplitInfo(N, A) = \sum_{v \in val(A)} \frac{\mid N_v \mid}{\mid N \mid} log_2 \frac{\mid N_v \mid}{\mid N \mid} \tag{6.3}$$

$$GainRatio(N, A) = \frac{Gain(N, A)}{SplitInfo(N, A)} \tag{6.4}$$

En fait, plus un paramètre à de valeurs distinctes, plus son SplitInfo est grand (Formule 6.3). En considérant le SplitInfo comme un coût de division, on peut normaliser le gain d'information (Formule 6.4). Le choix du prochain paramètre se base alors sur le gain maximal d'information qui ne divise pas trop les données ce qui permet de générer l'arbre de décision le plus rapide.

6.4.2.3 Index de Gini

Le dernier critère considéré pour sélectionner le prochain paramètre de test est l'index de Gini. Il s'agit d'une mesure qui calcule le taux d'erreur si on classe aléatoirement une donnée dans une des classes possibles après une partition par rapport à un paramètre A (Formule 6.5). Notre but est de choisir l'attribut qui minimise ce taux d'erreur (Formule 6.6). Il s'agit en fait de favoriser les divisions qui rassemblent les données d'une même classe dans un nœud unique.

$$Gini(N) = \sum_{i \neq j} p(i \mid N)p(j \mid N) = 1 - (p_1^2 + p_2^2) \tag{6.5}$$

avec p_1=probabilité(attaque) et p_2=probabilité(non attaque).

$$Gini_{split}(N, A) = \sum_{v \in val(A)} \frac{\mid N_v \mid}{\mid N \mid} Gini(N_v) \tag{6.6}$$

6.5 Détection d'intrusions avec sauvegarde d'états

Les arbres de décisions assurent une recherche rapide des intrusions. Nous définissons un système d'inférence qui construit automatiquement un arbre de décision pour chaque protocole applicatif supporté par l'IDS. Nous réalisons cette tâche avant le déploiement de l'IDS ce qui permet de générer un ensemble de bibliothèques à intégrer dans le système de détection d'intrusions. Nous nous intéressons dans cette section à la phase de détection. Il s'agit d'utiliser les arbres de décision déjà construits. Le processus de détection se divise en deux phases. Une première phase de prétraitement qui assure entre autres le décodage et la normalisation des champs des protocoles à analyser. Ensuite la phase de détection qui consiste à parcourir un ou plusieurs arbres de décisions tout en exécutant les opérateurs associés aux paramètres.

6.5.1 Phase de prétraitement des données

Dès la réception d'un nouveau paquet, le système de détection d'intrusions entame une phase de prétraitement. Il vérifie d'abord la légitimité du trafic en lui associant un contexte de connexion. Ensuite il normalise le contenu de quelques champs du protocole pour éviter les tentatives d'évasion lors de la phase de traitement. Nous détaillons dans la suite ces deux étapes de prétraitement.

6.5.1.1 Contextes de connexions

La première étape de prétraitement débute dès la capture du paquet et elle consiste à identifier le contexte de la connexion. Cette opération, intéressante pour une stratégie de détection avec filtrage brut de motifs, présente le même intérêt dans le cadre d'une analyse protocolaire. En effet, elle évite le traitement de données envoyées aléatoirement au réseau cible avec des intentions de reconnaissances ou de déni de services.

Les contextes de connexions sauvegardent les informations utiles sur les protocoles de transport et réseaux utilisés. Ils supervisent les opérations d'ouverture de connexions TCP et maintiennent les connexions actives établies sur le réseau. Néanmoins, la définition du contexte telle qu'elle est présentée au Chapitre 5 est insuffisante pour réaliser l'analyse protocolaire. En effet afin d'assurer le filtrage à la volée, nous rassemblons les traces des paquets circulant dans une même direction. Nous indexons les contextes par le quadruplet ⟨*adresse source*, *port source*,

adresse destination, *port destination*⟩ ce qui permet de créer deux contextes différents pour la même connexion, un pour le sens client → serveur et un autre pour le sens serveur → client. Avec une stratégie de filtrage brut des motifs d'attaques, les notions client et serveur sont peu utiles. Cependant, l'analyse protocolaire supervise les échanges client/serveur ce qui implique la création d'un contexte commun aux deux sens du trafic. La solution retenue consiste à indexer le contexte par le quadruplet ($\alpha = min(ip_1, ip_2)$, *port utilise par* α, $\beta = max(ip_1, ip_2)$, *port utilise par* β) (Figure 6.3). Ensuite les contextes sont rangés dans des arbres balancés puisque les paquets qui se suivent appartiennent souvent à la même connexion.

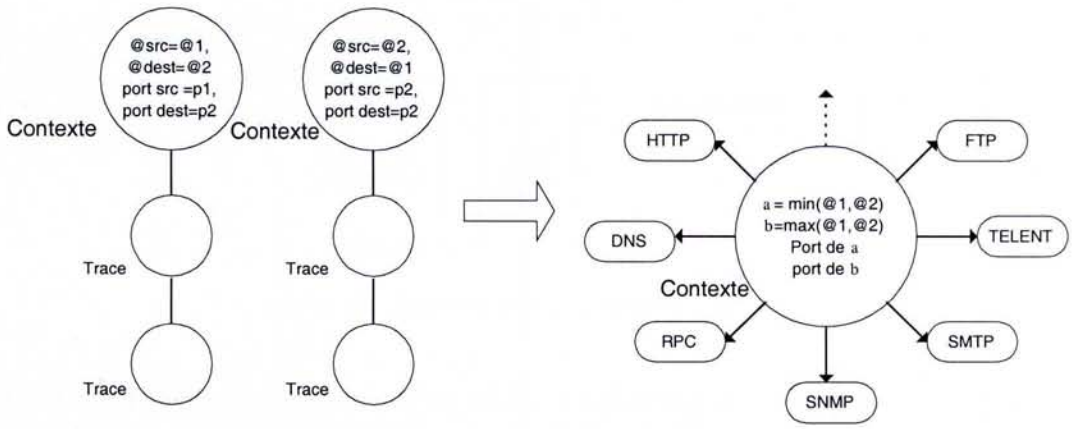


FIG. 6.2 – Indexation des contextes de connexions pour une analyse protocolaire

6.5.1.2 Conteneurs de protocoles et traces de paquets

La deuxième étape de prétraitement consiste à identifier les protocoles applicatifs et à effectuer les tests de conformité avant de parcourir l'arbre de décision. Suite à l'identification du protocole, nous extrayons les informations utiles à partir des champs du protocole. Nous distinguons à ce stade entre deux types d'informations, les informations permanentes et les informations temporaires, que nous sauvegardons dans deux structures différentes à savoir le conteneur de protocole et la trace de paquet.

Les informations permanentes sont les entrées utiles pendant toute la session protocolaire. Il s'agit par exemple de l'identifiant de l'utilisateur lors d'une session TELNET ou de la phase d'exécution du protocole SMTP. Ces données servent à superviser les différentes phases d'exécution du protocole applicatif ou à élaborer des fichiers journaux de sécurité plus détaillés. Par conséquent, on conserve ces informations pendant toute la durée de la connexion dans une structure de données permanente appelée conteneur. Le conteneur représente le contexte du protocole applicatif et il est unique par session. Il est relié directement à l'arbre de connexions, ce qui permet d'y accéder rapidement après avoir vérifié la validité de la connexion au niveau transport. De plus, les protocoles applicatifs peuvent être encapsulés les uns dans les autres (du SSL sur HTTP par exemple). Nous formons alors plusieurs types de conteneurs qui se succèdent selon l'ordre d'encapsulation des protocoles. Pour ce faire, chaque conteneur dispose d'un pointeur vers le conteneur du prochain protocole encapsulé (Figure 6.3).

Les informations temporaires sont les données utilisées pour vérifier la conformité des champs du paquet (taille d'un champs, bits réservés, etc) ou pour parcourir l'arbre de décision associé au protocole analysé. Ces entrées servent à analyser le paquet courant et sont éliminées dès la fin de

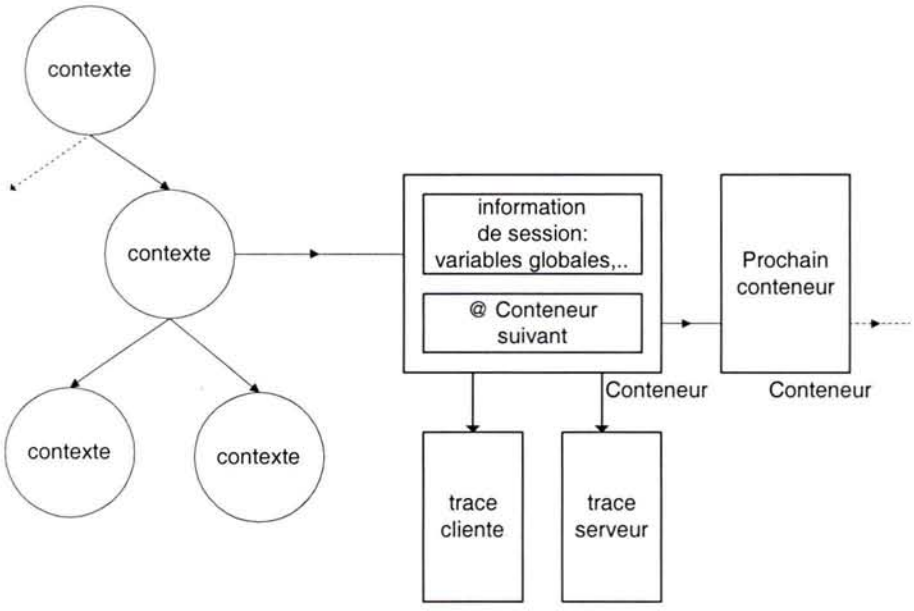


FIG. 6.3 – Phase de prétraitement du trafic

la phase de traitement. Nous sauvegardons ces informations temporaires dans une structure de données appelée trace. La trace est liée logiquement au conteneur du protocole et peut avoir deux types : une trace cliente et une trace serveur. D'une part, la trace cliente contient les données temporaires d'un paquet émis par l'utilisateur du service et elle est mise à jour dès la réception d'un nouveau paquet client. D'autre part, la trace serveur contient les informations d'un paquet émis par le serveur et subit le même mécanisme de mise à jour. La présence des deux traces cliente et serveur nous offre l'opportunité de superviser les échanges client/serveur, très utile dans le cadre d'une analyse protocolaire.

Après avoir rempli les structures de données nécessaires à l'analyse protocolaire (conteneur et traces), nous achevons la phase de prétraitement par des opérations de normalisation et de tests de conformité des champs des protocoles contenus dans le paquet. Ainsi nous résumons les différentes étapes de prétraitement comme suit :

1. identifier les protocoles contenus dans le paquet
2. sauvegarder les informations permanentes dans le conteneur et les informations temporaires dans une trace cliente ou serveur
3. normaliser les données pour éviter les attaques d'évasion
4. effectuer les tests de conformité des champs du protocole
5. superviser la bonne exécution d'un protocole applicatif. Cette opération s'effectue d'abord par rapport aux informations de session en sauvegardant dans le conteneur du protocole l'état courant du serveur. La Figure 6.4 montre le scénario implanté pour le protocole FTP. Ensuite nous réalisons une analyse au niveau paquet en comparant les deux dernières traces, cliente et serveur, liées au contexte du protocole applicatif (par exemple envoi d'une requête DNS, reception de la réponse du serveur).

Notons que les deux phases 4 et 5 peuvent être réalisées sur l'arbre de décision. Néanmoins nous préférons les inclure dans l'étape de prétraitement pour conserver la simplicité de l'arbre.

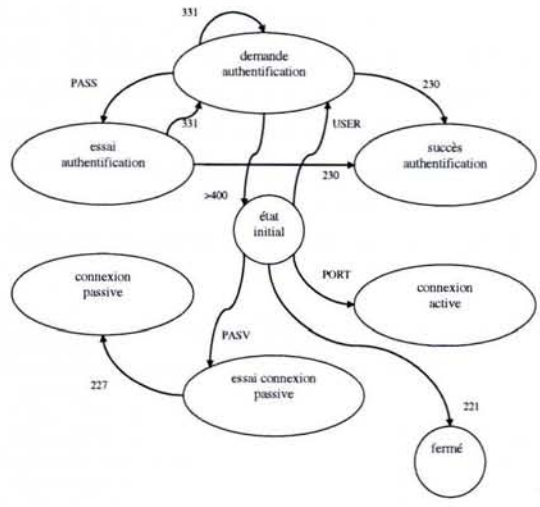


FIG. 6.4 – Exécution du protocole FTP

6.5.2 Phase de traitement des données

La dernière phase de l'analyse protocolaire consiste à parcourir l'arbre de décision en utilisant les informations du conteneur et des traces cliente et serveur. Si on atteint une feuille positive de l'arbre alors une attaque est détectée et on déclenche une alarme. Parmi les opérateurs utilisés pour traverser l'arbre on note l'opérateur "contient" qui suscite l'utilisation du filtrage. En effet l'opération est nécessaire pour inspecter le contenu de quelques champs du protocole ou pour filtrer les données d'un protocole non encore supporté par l'IDS. Néanmoins nous retardons l'utilisation de cet opérateur puisqu'elle est coûteuse en temps et peut engendrer un grand nombre de nœuds fils.

Nous présentons dans la Section 6.6 notre expérimentation qui porte sur le protocole RPC. Nous montrons via cette réalisation l'intérêt de l'analyse protocolaire pour mieux superviser le trafic réseau.

6.6 Implémentation

Nous nous sommes intéressés dans notre implémentation à une liste de protocoles fréquemment utilisées dans les communications réseaux. Nous divisons les protocoles étudiés en deux catégories. La première famille concerne des protocoles qui ne présentent pas un format unique de trames. Ils traduisent généralement des commandes passées par un utilisateur ou une application réseau. En particulier, nous étudions les protocoles FTP, TELNET, SMTP et HTTP. Nous interprétons les données de ces services via leurs fonctionnalités. On distingue par exemple une commande FTP, l'argument de cette commande, sa taille et la réponse du serveur FTP. La deuxième famille des protocoles étudiés possède un format standard de trame. Nous utilisons les champs de cette trame pour définir les paramètres employés pour déployer l'arbre de décision. En particulier, nous considérons les protocoles RPC, DNS et SNMP. Notre expérimentation s'est restreinte au protocole RPC. Nous montrons dans la suite le format des données RPC ensuite nous décrivons notre expérimentation et les résultats obtenus.

6.6.1 Protocole RPC

RPC (Remote Procedure Call) est une technique utilisée dans le modèle "client-serveur". Elle gère les interactions entre des clients et des serveurs (Figure 6.5). Le client appelle des procédures qui sont exécutées sur un ordinateur distant grâce à un serveur d'application. Divers services se basent sur RPC comme NFS, NIS et Mount. Notons que le programme serveur RPC utilise des ports éphémères c'est à dire des ports non connus à l'avance. Ceci nécessite un "diffuseur de ports dynamiques" (Port Mapper) pour garder une trace des numéros de port éphémères utilisés par les différents programmes RPC. Sur Sun RPC, ce diffuseur de ports tourne sur le port UDP 111.

Requête	Taille (octet)	Réponse
Entête IP	20	Entête IP
Entête UDP	8	Entête UDP
ID de transaction	4	ID de transaction
Appel (0)	4	Réponse (1)
version RPC	4	Status (0=accepté)
N° programme	4 jusquà 400	Vérificateur
N° version	4	Status d'acception
N° procédure	4 N	Résultat de la procédure
Crédits	jusquà 400	
Vérificateur	jusqu'à 400	
Paramètre de la procédure	N	

FIG. 6.5 – Format d'échange RPC

6.6.2 Expérimentation

Nous nous intéressons dans cette partie à l'analyse de quelques services RPC. Nous définissons 12 paramètres pour spécifier les attaques sur le protocole RPC qui sont : flux, direction, type (requête=0, réponse=1), programme, procédure, argument, taille_argument, servicePmap, état, état_acceptation, msg et action. Ensuite nous utilisons les signatures d'attaques définies par Snort [21] et RealSecure v7 [169] pour modéliser quelques scénarios d'attaque. Nous montrons dans le Tableau 6.3 un extrait de la spécification RPC qui nous permet de construire l'arbre de décision.

La construction de l'arbre de décision s'effectue avant le déploiement de l'IDS et se traduit par une bibliothèque à intégrer dans le système de détection d'intrusions Snort 1.9. Ensuite l'IDS charge pendant son initialisation la bibliothèque en question et enregistre un pré-processeur pour pouvoir identifier le protocole et remplir la trace RPC. Cette étape d'extraction des données à partir du contenu des paquets nécessite la connaissance du format des messages RPC [158], la représentation des données XDR [159] et les synoptiques des procédures appelées [160]. Pour ce faire, nous employons la bibliothèque ONC RPC qui offre plusieurs routines pour extraire les champs du protocole RPC. Ensuite nous supervisons en détail l'exécution du service portmapper (diffuseur de ports dynamique).

```
VAR PortSweep 0
...
%%
#set 1
RULE (direc :From_client ; Flux :TCP ; prog :100249 ; argSize>999 ; msg :"RPC snmpXdmi overflow")
RULE (direc :From_client ; Flux :TCP ; prog :100083 ; proc :15 ; msg :"RPC ttdbserv overflow")
...
#set 2
RULE (direc :From_client ; Flux :TCP ; prog :100000 ; proc :3 ; servicePmap :100087 ; action :PortSweep++)
RULE (direc :From_client ; Flux :UDP ; prog :100000 ; proc :3 ; ServicePmap :100087 ; action :PortSweep++)
RULE (direc :From_client ; Flux :TCP ; prog :100000 ; proc :3 ; ServicePmap :100099 ; action :PortSweep++)
RULE (direc :From_client ; Flux :UDP ; prog :100000 ; proc :3 ; ServicePmap :100099 ; action :PortSweep++)
...
#set 3
RULE (direc :From_client ; Flux :TCP ; prog :100000 ; proc :4 ; msg :"RPC Pmap Dump") )
RULE (direc :From_client ; Flux :UDP ; prog :100000 ; proc :5 ; ServicePmap :100005 ;
        msg :"RPC Callit with mountd")
...
%%
BHV PortSweep > MAX_RPC_GET_SWEEP ⇒ Log "RPC PortSweep attempt"
...
```

TAB. 6.2 – Extrait de la spécification du protocole RPC

Le service portmapper permet à des clients RPC d'interroger une machine distante sur les numéros de ports attribués aux différents services RPC qu'elle offre. Il se base généralement sur le protocole UDP et utilise le port 111. Par ailleurs il est fréquemment sollicité par les attaquants qui essaient d'identifier les services fournis par la machine. Ainsi en connaissant les versions de ces services, ils peuvent exploiter les vulnérabilités connues et attaquer le système via par exemple des dépassements de tampon.

Le service portmapper définit 6 procédures. Les 3 premières procédures Set, Unset et Get prennent en argument la structure mapping, qui contient le numéro du programme, sa version, le protocole et le port utilisé. Ensuite la fonction dump ne prend aucun argument et retourne une liste de mapping. Elle permet en fait de lister les informations liées à des services RPC de la machine. Il existe également la fonction Null et la fonction Callit qui permet d'appeler un service en ne connaissant pas son numéro de port et cela par l'intermédiaire d'un autre service.

La spécification RPC du Tableau 6.3 montre quelques attaques sur le protocole RPC. La première règle détecte une tentative d'un dépassement de tampon sur le service snmpXdmi. Nous découvrons cette attaque en identifiant la valeur 100249 du paramètre programme et un argument de taille supérieure à 999 octets. La règle sid:569 de Snort détecte aussi l'attaque mais en effectuant deux opérations de filtrage des motifs |00000F96| et |00018799|. Elle définit également deux valeurs de décalage (offset) afin de réduire l'espace de recherche des motifs d'attaques. Néanmoins les attaquants manipulent la trame RPC pour modifier les positions des motifs et par suite empêcher la détection.

La deuxième liste de règles du Tableau 6.3 détecte un balayage des services RPC. On identifie dans chaque règle le service portmapper (programme = 100000) et le service RPC recherché (pmapService). Snort 1.9 détecte séparément chaque attaque en maniant des opérations de filtrage de motifs. Cependant, la détection du balayage n'est pas encore supportée par l'IDS. De même, Snort applique le filtrage pour détecter les appels suspects des procédures dump et callit du portmapper. Ces attaques sont décrits dans l'ensemble 3 du Tableau 6.3. Elles sont décelées lors d'une analyse protocolaire en identifiant directement les procédures dump (4) et callit (5) du service pormapper.

Notre expérimentation a pour objectif de détecter des attaques sur le service RPC et particulièrement sur le programme portmapper. D'abord, on a défini un filtre sur RPC et collecté une partie du trafic transitant sur notre réseau de test. Nous avons remarqué que la majorité des paquets comportent des appels Callit au service ypserv. Malheureusement ces échantillons ne permettent pas de détecter des attaques RPC étant données qu'elles portent sur un seul type de trafic qui ne contient pas d'attaques. Par conséquent, nous créons nos propres paquets pour établir des sessions RPC anormales. Nous nous basons sur les deux protocoles de transport TCP et UDP et nous appelons une variété de services tels que mountd, autofsd et rstatd. Ainsi, nous construisons 414 paquets qui représentent 46 formes d'attaques.

Nous exécutons la version 1.9 de Snort afin de détecter les différentes attaques. Seulement 3 classes sont décelées qui concernent l'usage de la procédure Dump sur des trafics TCP (sid:598) et UDP (sid:429) et une requête Get pour s'informer du service rstatd (sid:583). En analysant les règles de détection d'attaques impliquées, nous remarquons qu'elles s'appuient toutes sur des opérations de filtrage de motifs appliquées sur tout le contenu du paquet. Les autres règles de détection qui ne sont pas déclenchées, emploient un filtrage de motifs dans un espace réduit de recherche en définissant des décalages et des profondeurs de recherche. Cette diminution qui sert à optimiser la détection d'attaques risque de générer de faux négatifs si elle s'applique aléatoirement sans effectuer une analyse protocolaire sur le contenu du paquet.

Nous réitérons l'expérience en incorporant notre bibliothèque de détection d'attaques sur RPC. Toutes les formes d'attaques sont détectées puisqu'il s'agissait d'identifier simplement le programme portmapper, ses procédures et les services RPC sollicités. Notons que la détection passe par une phase d'identification du protocole afin de remplir la trace RPC associée au paquet analysé. C'est une charge supplémentaire d'analyse qu'est rapidement compensée en évitant durant la phase de détection des opérations de filtrage de motifs.

6.7 Conclusion

Le filtrage de motifs constitue une opération intéressante dans le processus de détection d'intrusions. Néanmoins son application directe sur le contenu des paquets peut engendrer un grand nombre de faux positifs ainsi que de faux négatifs. Nous avons présenté dans ce chapitre les avantages de l'analyse protocolaire qui permettent de combler les lacunes du filtrage brut des motifs d'attaques. Parmi ces avantages, on note la détection d'autres types d'attaques liés à l'exécution anormale des protocoles applicatifs. Cette opération s'effectue après une étape d'identification du protocole et de décodage de la trame reçue. Ensuite, une suite de tests sont réalisés sur les champs du protocole ce qui permet de conclure la présence des attaques. L'ordre de ces tests est vital pour accélérer la détection. Il dépend essentiellement des caractéristiques du trafic et de l'activité intrusive couramment détectée sur le réseau. Par conséquent, nous avons appliqué trois méthodes de la théorie d'information (gain d'information, proportion du gain d'information et index de Gini) pour organiser convenablement l'ensemble de ces tests. Ainsi nous avons construit plusieurs arbres de décision, un pour chaque protocole applicatif analysé. Nous avons appliqué cette méthode d'analyse sur sept protocoles applicatifs fréquemment utilisés dans les communications réseau. Ensuite, notre expérimentation s'est portée sur le protocole RPC et a permis de détecter des attaques non décelées par Snort 1.9.

Suite à la division du trafic, nous avons présenté deux méthodes de détection d'intrusions à savoir le filtrage de motifs et l'analyse protocolaire. Ces techniques détectent des scénarios d'attaques qui débutent généralement par une phase de reconnaissance active. Les attaquants emploient cette stratégie pour découvrir les vulnérabilités sur le réseau cible et attaquer par la

suite les services non protégés du site. Par conséquent, la détection des attaques de reconnaissance est intéressante pour comprendre les intentions des intrus. Nous présentons dans l'Appendice une méthode décentralisée pour détecter cette activité intrusive.

7

Conclusions et Perspectives

7.1 Conclusions

La détection d'intrusions se heurte à plusieurs handicaps dont le haut débit, les attaques d'évasion et la génération abondante de faux positifs. Au cours de cette thèse, nous avons abordé ces différents problèmes et proposé quelques solutions afin d'optimiser la détection d'intrusions.

D'abord la détection d'intrusions peut être adaptée à la politique de détection de l'entreprise et aux caractéristiques des systèmes de détection d'intrusions déployés sur le réseau. Par exemple le filtrage d'un trafic crypté est impossible au niveau réseau si on ne connaît pas les clés de sécurité. De plus, lors des situations de surcharge, l'analyse du trafic doit traiter en premier lieu les classes suspectes du trafic. Enfin, la méthode de détection employée par un IDS s'adapte mieux à une classe de trafic que à une autre. Nous avons décrit ces propriétés sous forme de règles de division de trafic et ajouté des ordres de priorité pour privilégier l'application de certaines règles. Ensuite nous avons proposé un nouvel algorithme de classification des paquets qui s'exécute en deux étapes : une recherche géométrique dans une matrice de classification suivie d'un parcours d'un graphe acyclique directe. La construction du graphe tient compte des priorités des règles et contourne les chevauchements en divisant les champs inspectés des règles en différents blocs puis en favorisant les blocs complets sur les blocs masqués.

Suite à la division du trafic, nous avons abordé la détection au niveau des systèmes de détection d'intrusions (IDS). Les IDS définissent des signatures d'attaques sous forme de règles de détection. Un parcours itératif de ces règles constitue une stratégie inefficace. Afin d'optimiser cette opération nous avons proposé et comparé entre deux méthodes d'application de règles. La première stratégie adapte la technique de classification proposée pour diviser le trafic. En classifiant les règles de détection d'attaques, nous avons réussi à accélérer la détection d'intrusions mais via une consommation supérieure de la mémoire. Afin de réduire cet usage, nous avons présenté une deuxième stratégie qui regroupe puis organise les règles de détection d'attaques. L'idée de la nouvelle représentation des règles est d'en déduire à partir des premiers tests la vérification d'autres ensembles de règles. Nous accélérons ainsi la sélection des règles candidates en réduisant le nombre de tests à effectuer.

Chaque règle candidate de détection d'attaques comporte un ensemble de tests dont la vérification permet de déceler des intrusions. En particulier, le filtrage d'une signature est opération fréquemment sollicitée. Afin d'échapper à cette méthode, les attaquants emploient des techniques dites d'évasion. Par exemple, la segmentation et la fragmentation du trafic sont deux moyens possibles pour réussir les tentatives d'évasion. Pour contrarier ces méthodes, les systèmes de détection d'intrusions se basent sur un processus de réassemblage et d'organisation des paquets.

Néanmoins, cette procédure alourdit la détection et requiert un espace mémoire supplémentaire pour stocker le trafic. Nous avons proposé une méthode de filtrage à la volée qui tient compte de plusieurs signatures d'attaques et une arrivée désordonnée des paquets. La solution emploie les graphes acycliques directes des mots pour filtrer les préfixes, les suffixes et les sous mots de motifs. Ensuite, elle utilise l'algorithme de filtrage d'Aho-Corasick pour retrouver les motifs dans le contenu des paquets.

Néanmoins, un filtrage brut des signatures d'attaques engendre un grand nombre de faux positifs si les motifs trouvés ne se rapportent pas aux conditions d'attaques. Par ailleurs, certaines attaques ne sont détectées qu'en tenant compte du contexte d'exécution des protocoles impliqués. L'analyse protocolaire résout ces limites en examinant les champs des protocoles et en supervisant les échanges clients/serveurs. Ces opérations se traduisent par un ensemble de tests dont l'ordre d'exécution importe beaucoup pour accélérer la détection d'intrusions. Nous avons employé trois méthodes de la théorie d'information, le gain d'information, la proportion du gain d'information et index de Gini, pour construire des arbres de décision adaptatifs. Chaque arbre représente un protocole applicatif et sera utilisé durant la phase d'analyse pour détecter les attaques.

En conclusion, la surveillance des réseaux à haut débit constitue de nos jours un problème difficile. Avant de procéder à la détection, une sélection des classes de trafic à analyser par chaque IDS est une solution intéressante pour tenir compte des capacités des IDSs et des propriétés du trafic. La phase de détection elle même, doit s'adapter au trafic reçu et se protéger contre les techniques d'évasion. Les résultats de cette surveillance distribuée aideront ensuite à définir d'autres politiques de division et d'analyse du trafic. Ceci nous mène aux perspectives de ce travail.

7.2 Perspectives

Supervision des activités des IDS

La division du trafic permet d'équilibrer la charge d'analyse sur plusieurs systèmes de détection d'intrusions. En effet, le contrôle continu des activités des IDS nous permet de souligner rapidement les situations de surcharge et par suite nous balançons le trafic vers d'autres IDS équivalents moins occupés. Nous envisageons développer ce point dans un futur proche afin d'assurer une analyse permanente du trafic réseau et cela même en cas d'arrêt brusque de certains IDS.

Reconnaissances passive et active pour l'auto (re)configuration des IDSs et du diviseur de trafic, le support du processus de détection et la vérification des attaques détectées

Nos diverses configurations du diviseur de trafic et des systèmes de détection d'intrusions s'effectuent manuellement en introduisant explicitement les adresses et les ports des différents services fournis sur le réseau. Nous essayons d'automatiser cette tâche en spécifiant uniquement dans les fichiers de configuration les services à explorer. Ensuite, une reconnaissance passive (et active) continue du réseau permet d'identifier ces services et d'assister l'administrateur dans sa configuration des dispositifs de sécurité. Cette reconnaissance construit une base de connaissances qui sera non seulement utile pour des tâches de déploiement, mais également pour supporter le processus de détection d'intrusions. En effet, elle résout les ambiguïtés de détection rencontrées lors de l'analyse du trafic. Ces ambiguïtés révèlent le plus souvent des tentatives d'évasion (TTL

insuffisante, adresse physique falsifiée, etc) ou des attaques de reconnaissances (utilisation des adresses non attribuées lors d'une recherche d'un service particulier sur le réseau).

Enfin, une bonne partie des alarmes déclenchées désignent des faits qui ne constituent plus (pas) des menaces sur le réseau supervisé. Par exemple, un serveur Web vulnérable Microsoft IIS a pu être corrigé par l'administrateur. Par ailleurs, la signature définie pour cette vulnérabilité ne concerne pas les autres types de serveurs HTTP tel que Apache. La connaissance de ces serveurs aidera alors à filtrer un nombre important d'alarmes.

Analyse protocolaire appliquée à d'autres applications réseaux

Nous envisageons profiter de l'analyse protocolaire pour superviser d'autres applications sur le réseau telles que les protocoles de sécurité et le routage. En effet, divers travaux vérifient les protocoles de sécurité à la recherche des scénarios qui mettent en cause certaines des propriétés de confidentialité ou d'authenticité. Nous pouvons compléter ces travaux en trouvant explicitement ces scénarios d'attaques. L'analyse protocolaire permet de suivre l'exécution de ces protocoles et de détecter la rediffusion des messages (en signant les anciens messages) et l'usurpation d'adresses (comparaison des TTL). L'étude des autres conditions nécessaires pour détecter ces scénarios d'attaques constitue une de nos perspectives.

Par ailleurs la supervision des protocoles de routage est intéressante pour protéger l'interconnexion des réseaux. Cette tâche devient nécessaire dans le contexte Ah-hoc puisque le segment réseau est virtuellement accessible à tous les utilisateurs dans un rayon qui dépasse le périmètre de l'entreprise. Nous envisageons déployer plusieurs senseurs afin de superviser les exécutions des protocoles de routage et de comparer les messages échangés entre les routeurs afin de détecter les incohérences qui révèlent la compromission d'un routeur ou l'altération ou l'insertion de nouveaux messages par des intrus malveillants.

Appendice
Détection décentralisée des balayages de ports

1 Introduction

La reconnaissance constitue une étape importante dans le cycle de vie d'une attaque. Elle peut être passive en récoltant des informations via une écoute passive du trafic réseau. Par ailleurs, un attaquant peut mener une reconnaissance active en testant via des paquets anormaux le comportement des hôtes du réseau cible. La reconnaissance est encore utile pour un administrateur qui veut localiser les vulnérabilités sur son réseau. Il corrige ainsi les failles de sécurité pour prévenir des futures attaques. Nous nous intéressons dans cet appendice à un type bien connu de reconnaissance active. Il s'agit du balayage de ports. La plupart des IDS placent une sonde à l'entrée du réseau afin d'analyser tout le trafic. Sur notre architecture de détection, cette tâche peut être accomplie au niveau du diviseur de trafic.

Cependant, l'analyse du trafic en un point unique du réseau présente quelques inconvénients. D'abord le haut débit présente un handicap pour pouvoir traiter tous les paquets. Une étude réalisée par Fidelis Security Systems [142] en 2003 montre que la bibliothèque PCAP rejette 9,5% du trafic si le débit binaire est égal à 962MBps et le temps de traitement par paquet est 10μs. Ensuite, les formes de balayage sur un vaste réseau local sont si nombreuses qu'il est difficile de les détecter. Enfin, un seul point de détection est souvent un point de faille qui peut être exploité par des attaquants afin de stopper l'analyse du trafic.

Nous présentons dans cet annexe un système décentralisée de détection du balayage des ports. Notre système installe des sondes sur différents segments du réseau et met en œuvre plusieurs agents pour analyser le trafic. Chaque agent dispose de sa propre mission mais coopère avec l'ensemble pour réussir la détection. Le reste de l'annexe est organisé ainsi : la Section 2 détaille les différentes techniques de balayage de ports. Ensuite, nous présentons dans la Section 3 les anciens travaux pour détecter cette activité intrusive. La Section 4 présente notre système de détection du balayage de ports alors que la Section 5 décrit l'implémentation et les résultats expérimentaux obtenus.

2 Techniques de balayage de ports

De nos jours, les attaquants disposent de plusieurs outils pour effectuer le balayage de ports. Ces outils à double tranchons sont disponibles aussi bien sur les sites de piratage que sur les sites de sécurité informatique. Nous citons à titre d'exemple Nessus, SAINT, PortFlah, Fport, et Nmap [117]. Les outils implémentent plusieurs types de balayage et intègrent diverses stratégies

pour réussir les attaques. Nous présentons dans la suite les différents types, stratégies et formes courantes de balayage.

2.1 Types de balayage

L'outil Nmap, initialement développé par Fyodor [57], effectue plusieurs types de balayage. Nous classons ses méthodes et d'une façon générale les types de balayage de ports en 4 catégories [42].

- Balayage avec ouverture complète de connexion : ce type de balayage ouvre une connexion TCP avec la victime. Le comportement de ce dernier permet de déterminer l'état du port scruté (Tableau 1). Cependant, l'inconvénient majeur de la technique réside dans l'achèvement de la connexion TCP et donc elle sera inscrite dans les journaux de sécurité.

Port Ouvert	Port fermé
Client → SYN Serveur → SYN/ACK Client → ACK	Client → SYN Serveur → RST/ACK Client → RST

TAB. 1 – Balayage avec ouverture de connexion

- Balayage avec demi ouverture de connexion : cette méthode de balayage n'achève pas l'opération Three Way Handshake (3WH) d'ouverture de connexion TCP. Elle échappe donc aux anciens utilitaires d'audit. Néanmoins le balayage exige parfois un privilège "super utilisateur" pour par exemple envoyer des paquets RST. Nous présentons dans le Tableau 2 le balayage SYN. Nous remarquons que la troisième phase du 3WH n'est pas réalisée.

Port Ouvert	Port fermé
Client → SYN Serveur → SYN/ACK Client → RST	Client → SYN Serveur → RST/ACK

TAB. 2 – Balayage avec demi ouverture de connexion

- Balayage furtif : ce type de balayage vise à contourner les pare feux et les routeurs filtrants. L'intrus emploie intelligemment les drapeaux TCP pour rendre le paquet comme s'il faisait partie d'une connexion déjà ouverte. Cependant, puisque il n'y a pas eu d'établissement de connexion, la victime interrompt la communication et répond en fonction de l'état de son port. Le balayage TCP FIN utilise cette faiblesse protocolaire pour déterminer l'état du port (Tableau 3). La technique est aussi qualifiée "balayage inverse" puisque elle cherche à determiner les ports fermés pour en déduire les ports ouverts. Elle est également appliquée lors des balayage SYN/ACK, XMAS (tous les drapeaux TCP sont activés) et NULL (aucun drapeau TCP n'est activé).

Port Ouvert	Port fermé
Client → FIN Serveur → Pas de réponse	Client → FIN Serveur → RST

TAB. 3 – Balayage furtif

- Balayage divers : les attaquants utilisent d'autres types de balayage en se basant sur des protocoles autres que TCP. En effet, ils envoient des paquets UDP de tailles nulles ce qui génère des messages d'erreurs ICMP si le port est fermé. Une autre astuce consiste à utiliser un serveur FTP proxy pour débuter une connexion passive des données. L'attaquant fixe l'adresse de la victime et un port temporaire pour un transfert passif des données.

2.2 Stratégies de balayage

Une stratégie simple de balayage consiste à scruter une suite continue de ports. Afin de détecter cette activité, les systèmes de détection d'intrusions et les pare-feux cherchent des séquences contiguës de numéros de ports. Les attaquants évitent la détection et scrutent aléatoirement les ports. Par ailleurs, ils remplacent un balayage continu dans le temps par un autre qui est plus réparti. Pour contrarier ces tentatives, les IDS définissent des fenêtres temporelles suffisamment grandes et comptent le nombre de connexions effectuées sur certains ports.

Durant le balayage, les intrus usurpent plusieurs adresses pour dissimuler leurs vraies identités au sein d'un gigantesque trafic. Ils empêchent ainsi une détection facile de l'origine de l'attaque. Enfin, une autre technique plus sophistiquée consiste à exercer un balayage coordonné. Plusieurs entités participent à cette attaque. Ensuite, l'attaquant collecte les divers résultats pour obtenir les informations de la reconnaissance.

2.3 Formes de balayage

Le balayage de ports présente 3 formes possibles que nous décrivons ci dessous :

- Balayage vertical : l'intrus vise une seule machine puis scrute une série de ports sensibles afin de détecter leurs états et en déduire les services fournis sur cet hôte.
- Balayage horizontal : l'intrus vise un seul port mais sur une plage d'adresses appartenant au réseau cible. Il essaie de localiser une machine offrant un service particulier.
- Balayage en bloc : c'est la combinaison des deux balayages précédents c'est à dire chercher un ensemble de ports sur une plage d'adresses.

Notons enfin que ces formes d'attaques sont dures à détecter lorsqu'elles sont lancées par un groupe d'attaquants. Dans ce cas, les intrusions sont qualifiées d'attaques distribuées ou coordonnées.

3 Etat de l'art : Détection des balayages de ports

Foukia [55] propose une méthode de détection des balayages TCP qui se base sur la fréquence d'apparition des connexions suspectes. Si les intervalles de temps qui séparent les événements suspects sont inférieurs à un seuil, alors une alerte est déclenchée. Le mécanisme de détection s'appuie sur deux populations d'agents mobile : les IDA (Intrusion Detection Agent) et les IRA (Intrusion Response Agent). La première catégorie d'agents détecte le balayage de ports. Elle calcule les distances de hamming entre les connexions courantes ce qui permet d'évaluer leur degré d'anomalie. Si ce degré dépasse un certain seuil alors une phéromone est émise. Il s'agit d'un agent menu d'un diamètre de propagation et d'un gradient de rétrécissement pour émettre une substance qui guidera la deuxième catégorie d'agents (IRA) vers le lieu d'attaque. Une fois installé sur l'hoté compromis, un IRA entreprend les actions nécessaires pour réparer le nœud et prévenir les futures attaques similaires. La solution proposée dans [55] est distribuée mais elle est appliquée à un seul type balayage.

Le système de détection d'intrusions Snort [136, 137] intègre plusieurs pré-processeurs (plugins) qui effectuent des divers traitements avant d'entamer la phase de détection. Parmi ces fonctionnalités supplémentaires, on souligne le préprocesseur de détection de balayage de ports "portscan2" développé par Mullen [137]. Cet utilitaire surveille un ensemble d'adresses pour un nombre maximum de balayage de ports pendant une période de temps fixe. Le pré-processeur organise les connexions suspectes dans un arbre indexé par les adresses sources des intrus Chaque nœud est lié à un deuxième arbre des adresses destinatrices (adresses appartenant au réseau local) qui sauvegarde le nombre et les numéros de ports scrutés. Un nœud du deuxième arbre contient une liste des connexions suspectes, les ports source et destination employés et les types des balayages réalisés. L'ensemble de ces arbres interconnectés sont soumis à une mise à jour continue afin de détecter les activités intrusives, les rapporter dans les fichiers journaux de sécurité et nettoyer les entrées devenues obsolètes. Notons qu'un nouveau pré-processeur appelé Flow-Portscan a été inclus dans Snort. Son but est de détecter les balayages horizontaux et verticaux tout en réduisant le nombre de faux positifs et la consommation mémoire engendrée par l'ancien pré-processeur. La méthode supervise les flux de deux catégories de machines : les "talkers" qui sont des nœuds actifs très sollicités et les "scanners" qui constituent le reste des hôtes. Chaque catégorie définit son propre seuil d'événements suspects dans une fenêtre de temps qui peut être fixe ou variable. Les deux pré-processeurs de Snort sont capables de détecter les balayages furtifs mais trouvent des difficultés pour résoudre le problème des balayages aléatoires, coordonnés et lents.

Staniford et ses co-auteurs [164] proposent une méthode de détection des balayages aléatoires et lents. Ils définissent un ensemble de fonctions d'évaluation heuristique afin attribuer un score d'anomalie à chaque événement suspect. Cette valeur intervient d'une part dans le calcul du score d'anomalie globale et d'autre part elle fixe la durée de vie de l'événement suspect. En effet, les événements, de durée déterminée, sont représentés dans un graphe d'événements et reliés aux voisins proches via un processus d'"annealing". Si le score d'anomalie total d'un groupe d'événements dépasse un seuil déterminé alors une alerte est déclenchée. Spice, l'implémentation de cette solution, présente des bons résultats, seulement elle repose sur une corrélation centralisée des événements suspects.

D'autres approches ont été proposées pour détecter le balayage de ports. Par exemple GrIDS [165] construit des graphes pour superviser les changements rapides des activités sur le réseau et entre autres le balayage de ports. De plus, Emerald [130] supervise les déviations dans les profits des adresses surveillées ce qui permet de détecter le balayage de ports.

4 Architecture pour la détection des balayages de ports

Nous présentons dans cette section notre système de détection de balayage de ports. La solution s'appuie sur la combinaison de 5 agents (Figure 1). Nous résumons dans un premier temps les fonctionnalités de chaque composant du système puis nous expliquons leur déploiement sur le réseau.

1. Senseur : Il capte et analyse le trafic sur un brin réseau. Le senseur dispose d'un ensemble de signatures d'attaques qu'il utilise pour filtrer les paquets reçus. Il supervise également le bon déroulement des connexions TCP ce qui lui permet de détecter plusieurs scénarios de balayage. Par ailleurs, il génère un événement suspect suite à la détection d'un coup de sonde. Cet événement est placé dans un tampon circulaire pour être consommé par un agent statique de détection de balayage vertical.

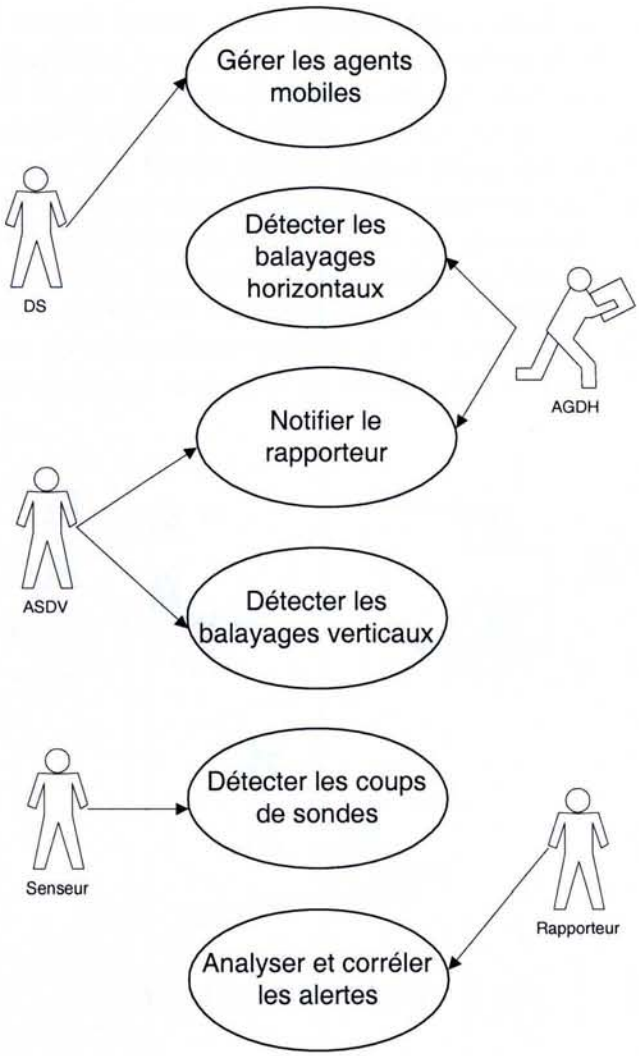


FIG. 1 – Diagramme de "cas d'utilisation" du système

2. Agent Statique de Détection Verticale (ASDV) : il reçoit via un tampon circulaire tous les événements de coups de sonde émis par le senseur. L'agent assure un ensemble de tâches comme la mise à jour de la table de coups de sonde et le remplissage d'un tableau blanc avec les derniers événements détectés. En outre, l'ASDV détecte les balayages verticaux réalisés sur les machines installées sur le brin réseau du senseur. Enfin, il élimine périodiquement les coups de sonde devenus obsolètes.

3. Agent Général de Détection Horizontale (AGDH) : c'est un agent qui détecte les balayages horizontaux et en blocs effectués sur les hôtes du réseau local. Il utilise principalement le tableau blanc rempli par l'agent statique ASDV.

4. Directeur de Sécurité (DS) : il contrôle la bonne exécution des différents agents déployés sur le réseau (création, déplacement, terminaison)

5. Rapporteur : il collecte et résume les alarmes déclenchées par les agents statiques et mobiles et les directeurs de sécurité déployés sur le réseau.

Les différents agents se déploient différemment sur le réseau (Figure 2) mais interagissent ensemble afin de détecter le balayage de ports. En effet, nous installons sur chaque brin du réseau local un senseur pour détecter les événements de coups de sonde. De plus, nous ajoutons sur la machine du senseur un agent statique ASDV pour détecter les balayages verticaux et un directeur de sécurité pour manipuler l'agent mobile AGDH. Les directeurs de sécurité sont rassemblés en des petits groupes pour former une zone. Ainsi le réseau se voit divisé en des petites zones indépendantes. Nous choisissons sur chaque zone un directeur de sécurité responsable qui crée l'agent AGDH. Cet agent détecte les balayages horizontaux apparues dans sa zone et notifie le directeur de sécurité responsable de l'état de détection. Ce dernier partage l'information de l'activité intrusive avec d'autres directeurs de sécurité responsables de zones ce qui permet de détecter des balayages sur tout le réseau. Nous détaillons dans la suite la tâche précise de chaque composant du système.

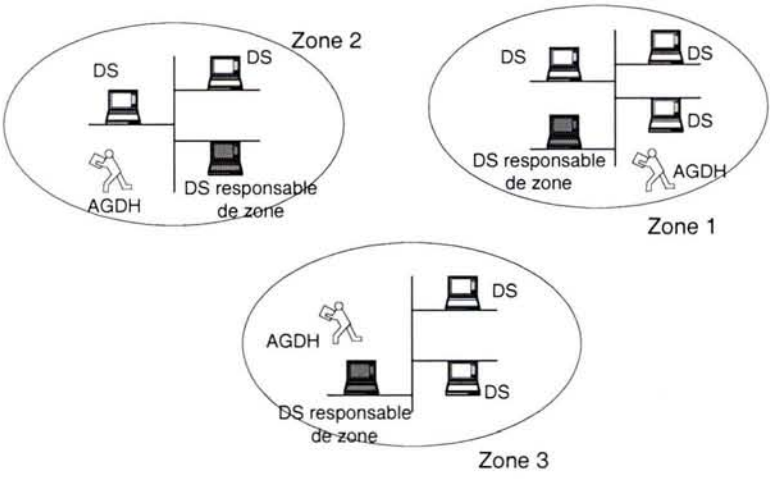


FIG. 2 – Déploiement des agents

4.1 Senseur

Le senseur capte les paquets transitant sur un segment réseau puis cherche les signatures d'attaques dans le contenu des protocoles réseaux et transports. On distingue deux types de signatures. La première catégorie s'applique directement sur le contenu du paquet c'est à dire sans superviser les phases d'établissement de la connexion. Ces signatures décrivent les combinaisons anormales des drapeaux TCP (exemple XMAS, NULL, SF, etc), les datagrammes UDP inhabituels et les fragments suspects des paquets IP. La deuxième catégorie de signatures détecte les exécutions anormales du protocole TCP. On supervise alors l'établissement de la connexion via les contextes de connexions. Un contexte est identifié par le tuple ($\alpha = min(ip_1, ip_2)$, *port utilise par* α, $\beta = max(ip_1, ip_2)$, *port utilise par* β) et il est rangé dans un arbre balancé (Figure 3).

Le nettoyage de l'arbre constitue une opération importante pour le bon déroulement de l'analyse. Cette fonction élimine périodiquement les connexions restées longtemps inactives ce qui garantie un arbre minimale de connexions. Nous diminuions ainsi la quantité de mémoire utilisée et nous réduisons le temps nécessaire pour parcourir l'arbre lors des mises à jours des contextes. De plus nous détectons les connexions incomplètes (demie ouverture de connexions), considérées comme des tentatives de balayage.

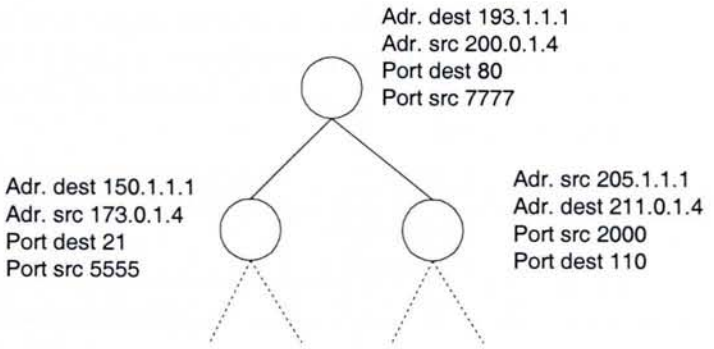


FIG. 3 – Arbre balancé des états de connexions TCP

Si un paquet anormal ou une connexion suspecte a été détectée alors le senseur génère un événement de coup de sonde. Cet événement est caractérisé par 8 paramètres qui sont l'adresse de l'attaquant, l'adresse de la victime, le port source, le port destination, le TTL, le temps de capture, le protocole de transport et le type du coup de sonde. Ensuite, l'événement est déposé dans un tampon circulaire pour être consommé par un agent statique responsable de l'analyse locale des événements suspects. Ce mécanisme assure la séparation entre les deux opérations de détection et d'analyse des coups de sonde. De plus il notifie l'ASDV dés l'apparition de nouveaux événements. Enfin il gère les rafales d'événements en les stockant dans une file circulaire.

Le tampon est indexé par 2 pointeurs "in" et "out" qui indiquent respectivement les prochaines positions de dépôt et de retrait des événements. Par ailleurs, il est contrôlé par des sémaphores assurant que le producteur ne dépose pas les données si le tampon est plein et que le consommateur ne retire pas les données si la file est vide.

4.2 Agent Statique de Détection Verticale (ASDV)

L'agent statique consomme les événements issus du senseur afin d'analyser les coups de sonde et détecter les balayages verticaux. Il utilise 3 tables pour stocker et traiter les informations reçues.

- Table Vscan : elle regroupe dans une fenêtre de temps T les coups de sonde effectués par un attaquant sur une unique machine appartenant au brin supervisé (Tableau 4). Si ce nombre est supérieur à un seuil MAXV alors l'agent statique annonce au rapporteur la détection d'un balayage vertical.

IP Dest	IP Src	Nombre de sondes	Liste de ports	Nombre de ports
152.8.1.8	198.11.1.2	5	12,13,14	3
152.8.1.8	125.25.34.44	1	80	1
152.8.1.8	125.25.34.45	1	80	1

TAB. 4 – Table Vscan

- Tableau blanc : si le seuil MAXV de détection des balayages verticaux n'est pas atteint alors l'événement reçu peut appartenir à un autre type de balayage (Bloc, Horizontal). L'agent statique inscrit l'événement sur un tableau blanc constamment consulté par des agents mobiles (Tableau 5). Il notifie également le directeur de sécurité installé sur la même machine de la présence de ces nouvelles données afin qu'il appelle l'agent mobile

AGDH. Avant d'appeler cet agent, le directeur de sécurité vérifie si un ancien appel a été déjà effectué et non encore satisfait. Dans ces conditions, il ignore l'appel pour ne pas surcharger le réseau. Dans le cas contraire, il notifie le directeur de sécurité responsable de zone pour lui dépêcher l'agent AGDH.

Port	Nombre d'adresses	Liste d'adresses	Nombre d'adresses distinctes
80	3	125.25.34.44 125.25.34.45	2
14	1	198.11.1.2	1

TAB. 5 – Tableau Blanc

- Table générale de coups de sondes : cette table rassemble tous les événements inscrits dans la table Vscan et permet de simuler le déplacement de la fenêtre glissante (Tableau 6). Les événements inscrits dans la table générale ne sont pas corrélés par l'adresse de l'intrus comme il est le cas pour la table Vscan. De plus, chaque événement est muni de sa date de création et considéré obsolète après une période de temps T qui correspond à la taille de la fenêtre.

IP Dest	IP Src	Port Dest	Port Src	TTL	Temps	Proto	Type
152.8.1.8	198.1.1.2	12	25	64	15 :46 :08	TCP	NULL
152.8.1.8	198.1.1.2	12	25	64	15 :46 :08	TCP	NULL
152.8.1.8	198.1.1.2	13	25	64	15 :46 :09	TCP	NULL
152.8.1.8	198.1.1.2	13	25	64	15 :46 :09	TCP	NULL
152.8.1.8	198.1.1.2	14	29	64	15 :46 :12	TCP	NULL
152.8.1.8	125.25.34.44	80	3434	64	15 :46 :13	TCP	XMAS
152.8.1.8	125.25.34.45	80	3435	64	15 :46 :13	TCP	XMAS

TAB. 6 – Table générale des coups de sondes

4.3 Agent Général de Détection Horizontale (AGDH)

Nous divisons le réseau en plusieurs zones. Chaque zone contient des senseurs et un unique agent général pour détecter les balayages horizontaux et en bloc. La division du réseau en zones indépendantes permet de limiter les déplacements des agents mobiles et par conséquent d'accélérer la détection des attaques. De plus, en couvrant seulement une partie du réseau, les agents mobiles transportent moins d'informations ce qui évite la surcharge réseau.

La mission principale de l'agent AGDH est de détecter les balayages horizontaux et en bloc dans une zone réseau. Pour ce faire, il corrèle les événements récupérés du tableau blanc des machines visitées avec ses propres informations ce que lui permet de mettre à jour son état. L'appel de l'agent mobile AGDH s'effectue lorsque le directeur de sécurité local détecte de nouvelles entrées dans le tableau blanc. Ce mécanisme évite au AGDH le parcours naïf de l'ensemble des hôtes et par suite préserve les ressources des hôtes et du réseau. Par ailleurs, on évite les appels récurrents de l'agent général si une requête est émise et non encore satisfaite. Cependant, si le temps d'attente dépasse une période seuil alors une alerte est envoyée au directeur de sécurité responsable de zone pour analyser le comportement de l'agent AGDH. De son coté et à chaque message reçu, l'agent général ajoute les adresses des machines appelantes dans la liste de son itinéraire pour consulter ensuite les tableaux blancs (Figure 4).

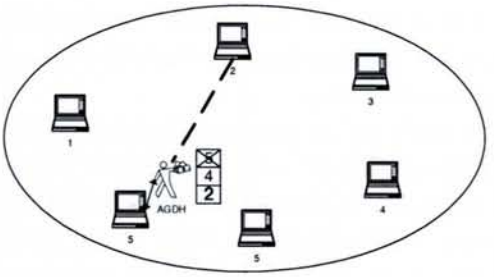


FIG. 4 – Parcours du AGDH au sein de son groupe

Afin de détecter les balayages horizontaux et en blocs, l'agent général met à jour son état en fonction des données des tableaux blancs des hôtes visités. Les connaissances du AGDH sont stockées dans un arbre équilibré indexé par les ports attaqués (Figure 5). Chaque nœud de l'arbre comporte une liste des adresses IP à l'origine du balayage. De plus nous associons à chaque adresse le nombre d'événements suspects. Si ce nombre est supérieur à un seuil MAXH alors un balayage horizontal est détecté. Par ailleurs, si plusieurs nœuds possèdent un nombre de coups de sondes supérieure à un seuil MAXB alors un balayage en bloc est décelé.

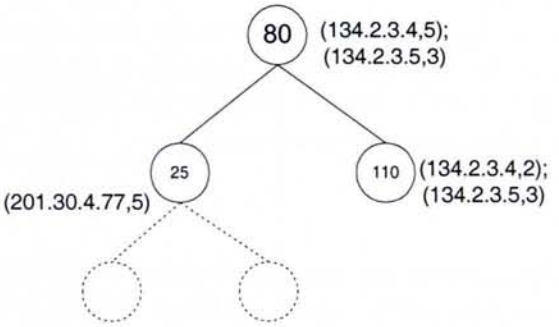


FIG. 5 – Structure de données de AGDH

Le nettoyage des informations obsolètes transportées par les agents généraux est une opération importante pour assurer la légèreté des agents. Cette opération est réalisée périodiquement après un nombre de sauts et/ou une longue période d'inactivité des agents. Chaque agent est menu d'une estampille absolue qu'est fonction du nombre de sauts effectués et du temps resté sur chaque hôte. De plus, chaque nœud de l'arbre présente une estampille relative qui est égale à la valeur de l'estampille absolue lors de la dernière mise à jour. Ensuite, à chaque période de l'estampille absolue, l'agent général déclenche sa propre opération de nettoyage. Il élimine toutes les entrées dont les estampilles relatives diffèrent trop de l'estampille absolue.

4.4 Directeur de Sécurité

Le directeur de sécurité fonctionne principalement comme un serveur d'agents qui assure la bonne exécution des AGDH. On distingue deux types de directeur de sécurité, ordinaire et responsable de zone :

- Directeur de sécurité responsable de zone : une seule instance de ce serveur existe par zone. Ce directeur crée l'AGDH de la zone et distribue son proxy aux autres directeurs de sécurité

ordinaires. Par ailleurs, les directeurs de sécurité responsables de zones communiquent ensemble afin de partager les informations de l'activité intrusive. Ils détectent ainsi les attaques qui s'étalent sur tout le réseau.

- Directeur de sécurité ordinaire : durant son initialisation, ce directeur de sécurité demande au directeur de sécurité responsable de zone le proxy de l'agent AGDH. Ensuite, il utilise ce proxy pour appeler l'agent dés la réception de nouvelles entrées dans le tableau blanc. Néanmoins, si une ancienne information est déjà écrite dans le tableau blanc sans que l'AGDH ne soit passé, l'envoie du message est ignoré pour ne pas surcharger le réseau.

Les directeurs de sécurités responsables de zones coopèrent ensemble pour détecter les balayages sur tout le réseau. En effet, en détectant un balayage dans une zone, l'AGDH transmet à son directeur de sécurité responsable de zone les données concernant l'activité intrusive. Ce dernier se charge d'envoyer les données reçues vers un unique directeur de sécurité responsable d'une autre zone qui à son tour exécute la même tâche jusqu'à aboutir au premier directeur de sécurité. En recevant le message, un directeur de sécurité responsable de zone appelle son AGDH afin d'obtenir des données supplémentaires à fusionner avec les données reçues. Il transmet ensuite le message au directeur de sécurité suivant responsable d'une zone formant ainsi un anneau unidirectionnel d'envoie de messages.

Néanmoins, un problème se pose concernant la concurrence des requêtes transmises sur le réseau. Prenons par exemple le cas de deux directeurs de sécurité responsables de zones qui émettent simultanément des données concernant le même port ou la même adresse. Puisque les messages suivent un anneau unidirectionnel, chacun des deux directeurs de sécurité recevra les données émises par l'autre. La solution retenue consiste à marquer les données émises par l'initiateur du message. Ensuite en attribuant des priorités aux directeurs de sécurité responsables de zones, seul le message du plus prioritaire sera conservé.

4.5 Rapporteur

Il s'agit d'un agent statique présent sur une seule machine, généralement celle de l'administrateur, qui reçoit les alertes émises par les agents ASDV et AGDH et les directeurs de sécurité responsables de zones. Il résume ensuite les activités de balayage et établit des statistiques afin de mieux représenter les attaques courantes et leurs évolution au cours du temps. L'administrateur consulte ces rapports de sécurité pour connaître les activités intrusives sur son réseau. Ces attaques constituent généralement les premières phases pour exploiter un système informatique en visant les services vulnérables sur le réseau.

5 Implémentation

Notre système de détection des balayages de ports implèmente cinq composants : le senseur, l'agent statique ASDV, l'agent mobile AGDH, le directeur de sécurité et le rapporteur. Le diagramme de séquence ci dessous résume le fonctionnement global du système.

Nous sommes entrain d'implantés le détecteur du balayage de ports. Tous les composants à l'exception du rapporteur sont achevés ce que nous a permis d'effectuer quelques tests préliminaires. Le projet est divisé en deux parties. La première partie intègre le senseur et a été implanté en "C". On utilise la bibliothèque PCAP pour capter les paquets puis on analyse le contenu des trames à la recherche des signatures d'attaques et des exécutions anormales du protocole TCP. La deuxième partie du projet est développée en "Java" et se base sur les Aglet [93] afin de modéliser les agents mobiles. Ensuite nous assurons la communication entre les deux modules via les sockets.

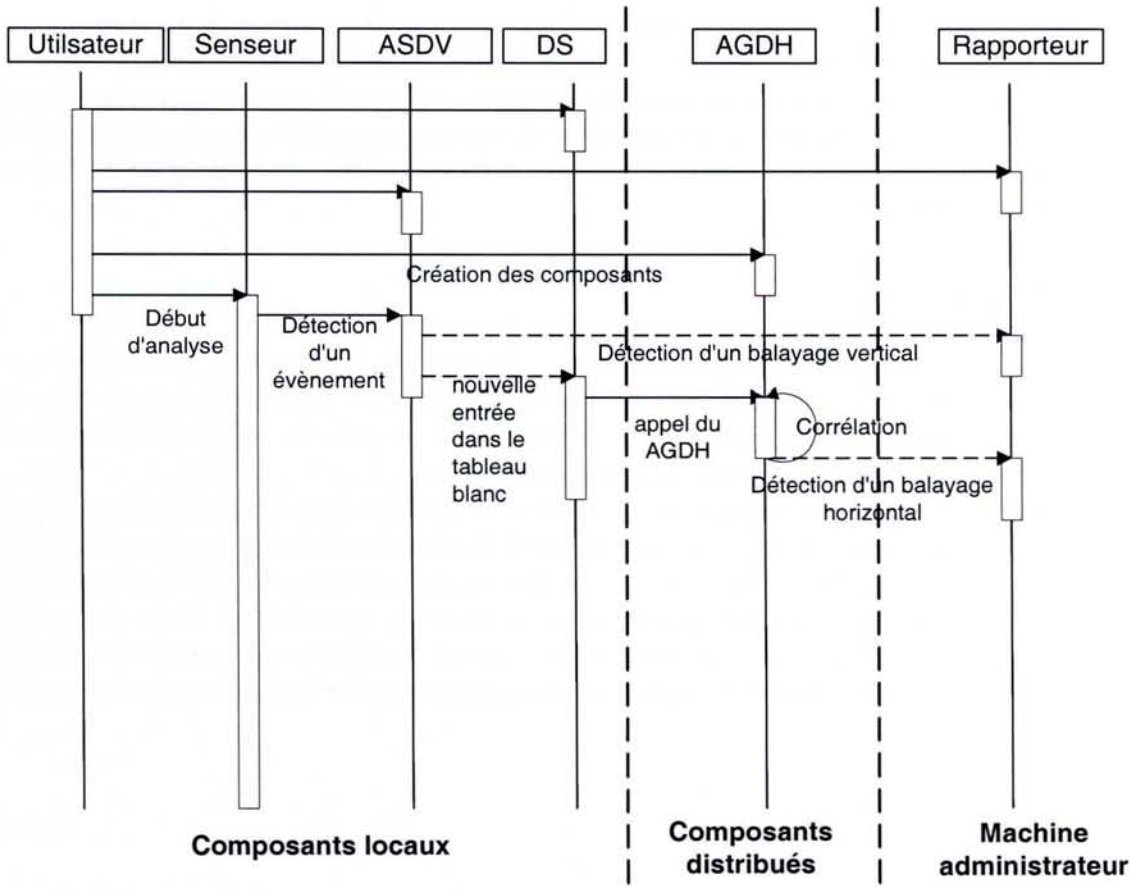


FIG. 6 – Diagramme de séquence du système

Nous avons réalisé notre première expérience sur une seule machine en déployant dessus plusieurs senseurs. Au cours de ces tests, nous avons forgé des paquets IP pour inclure 5 types d'attaques : SYN-FIN, SYN-ACK, XMAS, TCP NULL et UDP NULL. Par ailleurs, nous avons fragmenté les entêtes des protocoles TCP et UDP et créé des scénarios anormals d'ouverture de connexions TCP. Parmi ces scénarios, nous avons ouvert partiellement des connexions TCP et achevé parfois la connexion sans envoyer de données. Enfin, nous avons créé quelques segments TCP sans établir de connexions TCP. Le but principal des tests était de vérifier la bonne exécution du système en détectant tout ce qui a été insérées comme attaques. Nous avons réussi à détecter les différentes attaques ce qui prouve que les senseurs détectent les coups de sonde, génèrent les événements suspects et les transmets aux ASDV. Ces derniers détectent les balayages verticaux et nettoient régulièrement les tables Vscan. Par ailleurs, les agents mobiles visitent les différents directeurs de sécurités et détectent les balayages horizontaux et en bloc.

Nous avons crée au cours d'une deuxième expérience une zone contenant deux machines puis lancé l'outil Nmap contre ces hôtes. Une seule machine a été configurée pour contenir le directeur de sécurité responsable de zone. Ce dernier s'en charge alors pour créer l'AGDH. Ensuite, le seuil de détection a été fixé à 10 ce qui nous a permis de détecter les attaques après 3156 ms de la réception du premier événement suspect. Notons que le déplacement de l'agent d'une machine à une autre dépend des tâches effectuées sur chaque hôte. Nous avons obtenu une moyenne de 133 ms.

Nous envisageons compléter l'implémentation du système en développant le rapporteur puis déployer le système sur plusieurs machines localisées sur des brins distincts du réseau. Cette expérimentation nous permettra de mieux évaluer les performances de notre détecteur de balayage de ports. Par ailleurs, nous étudierons l'adaptation du système pour détecter des attaques plus complexes. En particulier, nous sommes intéressés par la détection des attaques sur les protocoles de routage filaire et Ad-hoc.

6 Conclusion

Les attaquants utilisent de plus en plus le balayage de ports afin de découvrir les vulnérabilités sur le réseau cible. De plus ils emploient diverses stratégies pour dissimuler leurs caractères intrusifs. Par conséquent, la détection de la reconnaissance active devient plus difficile mais indispensable afin de comprendre les intentions des attaquants.

Nous avons proposé dans cet Annexe une méthode décentralisée de détection des balayages de ports. L'approche est capable de détecter les différentes formes d'attaques à savoir les balayages verticaux, horizontaux et en bloc. Elle assure la supervision du trafic sur plusieurs zones. Chaque zone déploie un ensemble de senseurs pour capter le trafic et détecter les coups de sondes. Ensuite, les activités intrusives sont partagées via des agents mobiles ce qui permet de détecter les balayages sur chaque zone. Enfin les zones communiquent ensemble afin de déceler les attaques qui couvrent tout le réseau.

Bibliographie

[1] T. Abbes, A. Bouhoula, and M. Rusinowitch. On the fly pattern matching for intrusion detection with snort. *Annales de Telecommunications*, Vol. 59(N° 9-10) :941–967, Septembre-Octobre 2004.

[2] T. Abbes, A. Bouhoula, and M. Rusinowitch. Protocol analysis in intrusion detection using decision tree. In *International Conference on Information Technology : Coding and Computing (ITCC'04)*, volume 1, pages 404–408, Las Vegas, Avril 2004.

[3] T. Abbes and M. Rusinowitch. Fast multipattern matching for intrusion detection. In *U.E. Gattiker (Ed.), EICAR 2004 Conference CD-rom : Best Paper Proceedings*, Luxembourg, Mai 2004. (ISBN : 87-987271-6-8) 22 pages. Copenhagen : EICAR e.v.

[4] T. Abbes, M. Rusinowitch, and A. Haloi. Network traffic classification for intrusion detection. Technical Report RR-5230, INRIA-Lorraine, Juin 2004.

[5] A. Aho and M. Corasick. Efficient string matching : An aid to bibliographic search. In *Communications of the ACM, 18(6) :333–340*, 1975.

[6] K. G. Anagnostakis, E. P. Markatos, S. Antonatos, and M. Polychronakis. E2xB : A domainspecific string matching algorithm for intrusion detection. In *Proceedings of the 18th IFIP International Information Security Conference (SEC2003)*, pages 217–228, Mai 2003.

[7] H. Anderson. Introduction to Nessus. Technical report, Octobre 2003.

[8] J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P Anderson Co., Fort Washington, PA, Avril 1980.

[9] S. Antonatos, K. G. Anagnostakis, E. P. Markatos, and M. Polychronakis. Performance analysis of content matching intrusion detection systems. In *Proceedings of the International Symposium on Applications and the Internet (SAINT2004)*, Janvier 2004.

[10] S. Axelsson. Intrusion detection systems : A survey and taxonomy. Technical Report 99-15, Chalmers Univ., Mars 2000.

[11] F. Baboescu and G. Varghese. Aggregated bit vector search algorithms for packet filter lookups. Technical Report cs2001-0673, University of California, San Diego, Juin 2001.

[12] R. Bace and P. Mell. Intrusion detection systems. Special Publication 8000631, National Institute of Sta, 2001.

[13] D. Barbara, J. Couto, S. Jajodia, and N. Wu. Adam : a testbed for exploring the use of data mining in intrusion detection. *SIGMOD Rec.*, 30(4) :15–24, 2001.

[14] D. Barbara, J. Couto, S. Jajodia, and N. Wu. Adam : Detecting intrusions by data mining. In *Proceedings of the IEEE SMC Information Assurance Workshop*, West Point, NY, 2001.

[15] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M.T. Chen, and J. Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40(1) :31–55, 1985.

[16] R. S. Boyer and J. Strother Moore. A fast string searching algorithm. In *Communications of the ACM, 20(10) :762–772*, 1977.

[17] A. Briney. CISO Strategies - Survey, Mars 2003.

[18] M. M. Buddhikot, S. Suri, and M. Waldvogel. Space decomposition techniques for fast layer-4 switching. In *Proceedings of the IFIP TC6 WG6.1 & WG6.4 / IEEE ComSoc TC on Gigabit Networking Sixth International Workshop on Protocols for High Speed Networks VI*, pages 25–42. Kluwer, B.V., 2000.

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

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

[21] B. Caswell and M. Roesch. Snort rules database. http ://www.snort.org/snort-db/.

[22] C. Charras and T. Lecroq. *Handbook of Exact String Matching Algorithms.* King's College London, Octobre 2004.

[23] G. Cheung and S. McCanne. Optimal routing table design of ip address lookups under memory constraints. In *Proceeding of IEEE Infocom*, pages 1437–1444, 1999.

[24] S. Cho and S. Han. Two sophisticated techniques to improve hmm-based intrusion detection systems. In *Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection (RAID'2003)*, LNCS v. 2820, pages 207–219, Septembre 2003.

[25] C. Coit, S. Staniford, and J. McAlerney. Towards faster string matching for intrusion detection. In *Proc. of the DARPA Information Survivability Conference and Exhibition (DISCEX-02)*, pages 367–373, 2002.

[26] B. Commentz-Walter. A string matching algorithm fast on the average. In *Proceedings of the 6th Colloquium, on Automata, Languages and Programming*, pages 118–132. Springer-Verlag, 1979.

[27] M. Cooper, S. Northcutt, M. Fearnow, and K. Frederick. *Intrusion Signatures and Analysis.* New Riders, 1 edition, Janvier 2001.

[28] Microsoft Corporation. Unauthorized access to IIS servers through ODBC data access with RDS, Juillet 1999.

[29] Saint Corporation. Saint documentation contents. http ://www.saintcorporation.com/.

[30] M. Crochemore, A. Czumaj, L. Gąsieniec, T. Lecroq, T. Plandowski, and W. Rytter. Fast practical multi-pattern matching. *Information Processing Letters*, 71 :3–4, 1999.

[31] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

[32] F. Cuppens. Cooperative intrusion detection. In *International Sysmposium on Information superiority : tools for crisis and conflict-management*, Paris, Septembre 2001.

[33] F. Cuppens, S. Gombault, and T. Sans. Selecting appropriate counter-measures in an intrusion detection framework. In *Computer Security Foundation Workshop*, Juin 2004.

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

[35] H. Debar, M. Becker, and D. Siboni. A Neural Network Component for an Intrusion Detection System. In *Proceedings of the 1992 IEEE Computer Society Symposium on Reserach in Security and Privacy*, pages 240–250. IEEE, IEEE Service Center, Piscataway, NJ, 1992.

[36] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Comput. Networks*, 31(9) :805–822, 1999.

[37] H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion-detection systems. *Annales des Télécommunications*, 55(7–8) :361–378, 2000.

[38] H. Debar, L. Mé, and S. F. Wu, editors. *Recent Advances in Intrusion Detection, Third International Workshop, RAID 2000, Toulouse, France, Octobre 2-4, 2000, Proceedings*, volume 1907 of *Lecture Notes in Computer Science*. Springer, 2000.

[39] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small forwarding tables for fast routing lookups. In *SIGCOMM*, pages 3–14, 1997.

[40] D. E. Denning. An Intrusion-Detection Model. In *IEEE Trans. on Software Engg.*, volume 13, pages 222–232, Février 1987.

[41] N. Desai. Increasing performance in high-speed NIDS, a look at snort's internals. Technical report, SecurityFocus, Mai 2002.

[42] Dethy. Examining port scan methods - Analysing audible techniques. URL : www.in-f-or.it/informatica/docs/portscan.pdf.

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

[44] S.T. Eckmann. Translating Snort rules to STATL scenarios. In *Proceedings of the 4th International Workshop on the Recent Advances in Intrusion Detection (RAID'2001)*, LNCS v. 2212, pages 69–84, Octobre 2001.

[45] S. Edwards. Network Intrusion Detection Systems : Important IDS Network Security Vulnerabilities. Technical report, Top Layer Networks, Inc., Septembre 2002.

[46] R. Elz and R. Bush. RFC 2181 : Clarifications to the DNS specification, July 1997. Updates RFC1034, RFC1035, RFC1123. Status : PROPOSED STANDARD.

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

[48] H. Welte et al. The netfilter/iptables project. www.netfilter.org.

[49] B. Feinstein, G. Matthews, and J. White. The intrusion detection exchange protocol (idxp), Octobre 2002. IETF Intrusion Detection Working Group Internet Draft.

[50] A. Feldmann and S. Muthukrishnan. Tradeoffs for packet classification. In *INFOCOM (3)*, pages 1193–1202, Mai 2000.

[51] M. Fisk and G. Varghese. Fast content-based packet handling for intrusion detection. Rapport LA-UR-01-5459, University of California, San Diego, Departement of Computer science and Engineering, 2001.

[52] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 120. IEEE Computer Society, 1996.

[53] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-Nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, page 202. IEEE Computer Society, 1994.

[54] J. C. Foster. Realsecure 7.0. iss matures its ids into an enterprise-class, best of breed solution. Foundstone, Inc., Novembre 2002.

[55] N. Foukia and G. Di Marzo Serugendo. Self-organizing architecture for intrusion detection and response systems. In *Proceedings of HP-OVUA 2003 Workshop, Geneva, Switzerland*, Juillet 2003.

[56] K. L. Fox, R. R. Henning, J. H. Reed, and R. Simonian. A Neural Network Approach Towards Intrusion Detection. In *Proceedings of the 13th National Computer Security Conference*, pages 125–134, Washington, DC, Octobre 1990.

[57] Fyodor. The art of scanning. Phrack 51, URL : http ://www.phrack.com.

[58] T. D. Garvey and T. F. Lunt. Model based Intrusion Detection. In *Proceedings of the 14th National Computer Security Conference*, pages 372–385, Octobre 1991.

[59] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, pages 51–62. USENIX Association, 1999.

[60] R. Graham. Faq : Network intrusion detection systems. Version 0.8.3, March 21, 2000.

[61] R. Graham. Evolution of ids. URL : http ://www.robertgraham.com/slides/0304-evolution-of-ids.ppt, 2002.

[62] P. Gupta and N. McKeown. Packet classification on multiple fields. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 147–160. ACM Press, 1999.

[63] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Proceedings Hot Interconnects VII, Stanford*, pages 147–160, Août 1999.

[64] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, vol. 15, no. 2, pp. 24-32, 2001.

[65] N. Habra, B. Le Charlier, I. Mathien, and M. Abdelaziz. Asax : Software architecture and rule-based language for universal audit trail analysis. In *Proceedings of the Second European Symposium on Research in Computer Security (ESORICS 92)*, LNCS v. 648, pages 435–450, Novembre 1992.

[66] M. Handley, V. Paxson, and C. Kreibich. Network intrusion detection : Evasion, traffic normalization, and end-to-end protocol semantics. In *Proc. of the 10th USENIX Security Symposium*, 2001.

[67] A. Harrison. Cyberassaults hit buy.com, ebay, cnn and amazon. Computerworld company, Février 2000.

[68] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Computer Security*, 6 :151–180, 1998.

[69] R.N. Horspool. Practical fast searching in strings. Software - Practice & Experience, 1980. 10(6) :501-506.

[70] I.Charitakis, K.Anagnostakis, and E.Markatos. An active traffic splitter architecture for intrusion detection. In *Proceedings of 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2003), Orlando*, pages 238–241, Octobre 2003.

[71] K. Ilgun. USTAT : A Real-time Intrusion Detection System for UNIX. In *Proceedings of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA, Mai 1993.

[72] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State Transition Analysis : A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, 21(3), Mai 1995.

[73] Top Layer Networks Inc. IDS balancer with etrust intrusion detection. http ://www.toplayer.com/pdf/IDSBdsCAfipdf.pdf.

[74] R. Jagannathan, T. Lunt, D. Anderson, C. Dodd, F. Gilham, C. Jalali, H. Javitz, P. Neumann, A. Tamaru, and A. Valdes. System Design Document : Next-Generation Intrusion Detection Expert System (NIDES). Technical Report A007/A008/A009/A011/A012/A014, SRI International, Mars 1993.

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

[76] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(1) :323–350, 1977.

[77] C. Ko. Logic induction of valid behavior specifications for intrusion detection. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P 2000)*, page 142. IEEE Computer Society, 2000.

[78] C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference (ACSAC'94)*, pages 134–144, 1994.

[79] A. P. Kosoresow and S. A. Hofmeyr. Intrusion detection via system call traces. *IEEE Softw.*, 14(5) :35–42, 1997.

[80] J. Koziol. *SNORT 2.* août 2003.

[81] J. Koziol. *Snort 2.* Number ISBN : 2-7440-1624-1. CampusPress, 2003.

[82] C. Kruegel and T. Toth. Automatic rule clustering for improved, signature-based intrusion detection. Technical report, Distributed Systems Group , Technical University of Vienna, 2002.

[83] C. Kruegel and T. Toth. Using decision trees to improves signature-based intrusion detection. In *Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection (RAID'2003)*, LNCS v. 2820, pages 173–191, Novembre 2003.

[84] C. Kruegel, F. Valeur, G. Vigna, and R.A. Kemmerer. Stateful Intrusion Detection for High-Speed Networks. In *Proceedings of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA, Mai 2002. IEEE Press.

[85] G. Kucherov and M. Rusinowitch. Matching a set of strings with variable length don't cares. *Theoretical Computer Science*, 178(1–2) :129–154, 30 May 1997.

[86] S. Kumar. *A Pattern Matching Approach to Misuse Intrusion Detection.* PhD thesis, Purdue University, Department of Computer Sciences, août 1995.

[87] S. Kumar and E. H. Spafford. A Pattern Matching Model for Misuse Intrusion Detection. In *Proceedings of the 17th National Computer Security Conference*, pages 11–21, Octobre 1994.

[88] J. Kuri, G. Navarro, L. Mé, and L. Heye. A pattern matching based filter for audit reduction and fast detection of potential intrusions. In *Proceedings of the 3rd International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, LNCS v. 1907, pages 17–21, Octobre 2000.

[89] J. Kuri, G. Navarro, and L. Mé. Fast multipattern search algorithms for intrusion detection. *Fundamenta Informaticae*, 56(1–2) :23–49, 2003.

[90] T. V. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *Proceedings of ACM SIGCOMM '98*, pages 203–214, 1998.

[91] B. Lampson, V. Srinivasan, and G. Varghese. IP lookups using multiway and multicolumn search. *IEEE/ACM Transactions on Networking*, 7(3) :324–334, 1999.

[92] T. Lane and C. Brodley. An application of machine learning to anomaly detection. In *Proceedings of the 20th NIST-NCSC National Information Systems Security Conference*, volume 1, pages 366–380, 1997.

[93] D.B. Lange. *Java Aglet Application Programming Interface (J-AAPI)*. IBM Tokyo Research Laboratory, 1997.

[94] T. Lecroq. A variation on the boyer-moore algorithm. *Theoretical Computer Science*, 92(1) :119–144, 1992.

[95] W. Lee, L. Mé, and A. Wespi, editors. *Recent Advances in Intrusion Detection, 4th International Symposium, RAID 2001 Davis, CA, USA, Octobre 10-12, 2001, Proceedings*, volume 2212 of *Lecture Notes in Computer Science*. Springer, 2001.

[96] W. Lee, R. Nimbalkar, K. Yee, S. Patil, P. Desai, T. Tran, and S. Stolfo. A data mining and CIDF based approach for detecting novel and distributed intrusions. In *Proceedings of the 3rd International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, LNCS v. 1907, pages 49–65, Octobre 2000.

[97] W. Lee and S. J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4) :227–261, 2000.

[98] W. Lee, S. J. Stolfo, and K. W. Mok. Mining in a data-flow environment : experience in network intrusion detection. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 114–124. ACM Press, 1999.

[99] W. Lee, S. J. Stolfo, and K. W. Mok. Algorithms for mining system audit data. pages 166–189, 2002.

[100] W. Lee, S. J. Stolfo, and K.W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, Oakland, CA, 1999.

[101] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceedings of the IEEE Symposium on Security and Privacy*, page 130. IEEE Computer Society, 2001.

[102] P. Lichodzijewski, A. Zincir-Heywood, and M. I. Heywood. Host-based intrusion detection using self-organizing maps. In *The IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, IJCNN 02*, 2002.

[103] R. Lippmann, S. Webster, and D. Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *Proceedings of the 5th International Workshop on the Recent Advances in Intrusion Detection (RAID'2002)*, LNCS v. 2516, pages 307–326, Octobre 2002.

[104] R.P. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. Mcclung, D. Weber, S. Webster, D. Wyschograd, R. Cunninghan, and M. Zissman. Evaluating intrusion detection systems : The 1998 darpa off-line intrusion detection evaluation. In *Proceedings of the on DARPA Information Survivability Conference and Exposition (DISCEX '00)*, pages 12–26, Los Alamitos, CA, Janvier 2000. IEEE Computer Society Press.

[105] T. F Lunt. A Survey of Intrusion Detection Techniques. *Computers & Security*, 12(4) :405–418, Juin 1993.

[106] T. F. Lunt and R. Jagannathan. Prototype Real-Time Intrusion Detection Expert System. In *Proceedings - 1988 IEEE Symposium on Security and Privacy*, pages 59–66, Oakland, CA, Avril 1988. IEEE, New York, NY.

[107] T. F. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D. L. Edwards, P. G. Neumann, H. S. Javitz, and A. Valdes. Development and Application of IDES : A Real-Time Intrusion-Detection Expert System. Technical report, SRI International, 1988.

[108] T. F. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D. L. Edwards, P. G. Neumann, H. S. Javitz, and A. Valdes. IDES : The Enhanced Prototype. Technical Report SRI-CSL-88-12, SRI International, SRI International, Menlo Park, CA, Octobre 1988.

[109] T. F. Lunt, R. Jagannathan, R. Lee, A. Whitehurst, and S. Listgarten. Knowledge based Intrusion Detection. In *Proceedings of the Annual AI Systems in Government Conference*, Washington, DC, Mars 1989.

[110] L. Mé. Gassata, a genetic algorithm as an alternative tool for security audit analysis. In *Proceedings of the 1st International Workshop on the Recent Advances in Intrusion Detection (RAID'1998)*, Septembre 1998.

[111] L. Mé, Z. Marrakchi, C. Michel, H. Debar, and F. Cuppens. La détection d'intrusions : les outils doivent coopérer. *Revue de la REE*, Mai 2001.

[112] C. Marceau. Characterizing the behavior of a program using multiple-length N-grams. In *Proceedings of the 2000 workshop on New security paradigms*, pages 101–110. ACM Press, 2000.

[113] E. P. Markatos, S. Antonatos, M. Polychronakis, and K. D. Anagnostakis. Exclusion-based signature matching for intrusion detection. In *Proceedings of IASTED International Conference on Communications and Computer Networks (CCN 2002)*, pages 146–151, Octobre 2002.

[114] Nomad mobile research center. The hack faq. www.nmrc.org/pub/faq/hackfaq/index.html.

[115] J.H. Morris and V.R. Pratt. A linear pattern-matching algorithm. Technical report, University of California, Berkeley, 1970.

[116] D. R. Morrison. Patricia : Practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4) :514–534, 1968.

[117] MyCERT. Security tools : Scanner. http ://www.mycert.mimos.my/resource/scanner.htm.

[118] J. Nathan and B. Caswell. 100Mb IDS Tapping Diagram (with 1000bt span port). www.snort.org/docs.

[119] J. Nathan and B. Caswell. 100Mb IDS Tapping Diagram (with only 100bt span port). www.snort.org/docs.

[120] J. Nathan and B. Caswell. GIGE IDS Tapping Diagram (with load balancers).

[121] P. G. Neumann and P. A. Porras. Experience with emerald to date. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, pages 73–80. USENIX Association, 1999.

[122] S. Nilsson and G. Karlsson. IP-address lookup using LC-tries. *IEEE Journal on Selected Areas in Communications*, 17(6) :1083–1092, Juin 1999.

[123] S. Northcutt, J. Novak, and D. Mc Lachlan. *Détection des intrusions réseau.* CampusPress, Janvier 2001.

[124] M. Norton and D. Roelker. Hi-performance multi-rule inspection engine. Technical report, Sourcefire, Inc., Avril 2004.

[125] M. Norton and D. Roelker. Snort^{TM} 2.0 rule optimizer. Technical report, Sourcefire, Inc., Avril 2004.

[126] N. Nuansri, S. Singh, and T. S. Dillon. A process state-transition analysis and its application to intrusion detection. In *Proceedings of the 15th Annual Computer Security Applications Conference*, page 378. IEEE Computer Society, 1999.

[127] A. Zincir-Heywood P. Lichodzijewski and M. I. Heywood. Dynamic intrusion detection using self-organizing maps. In *The 14th Annual Canadian Information Technology Security Symposium (CITSS)*, 2002.

[128] V. Paxson. Bro : a system for detecting network intruders in real-time. *Comput. Networks*, 31(23-24) :2435–2463, 1999.

[129] M. Peters. Construction and use of a passive ethernet tap, Mai 2004.

[130] P. Porras and A. Valdes. Live traffic analysis of tcp/ip gateways. In *Internet Society Symposium on Network and Distributed System Security*, Mars 1998.

[131] P. A. Porras. STAT – A State Transition Analysis Tool for Intrusion Detection. Master's thesis, Computer Science Department, University of California, Santa Barbara, Juin 1992.

[132] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service : Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, January 1998.

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

[134] J. R. Quinlan. Induction of decision trees. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1 :81–106, 1986.

[135] M. Ramadas, S. Ostermann, and B. Tjaden. Detecting anomalous network traffic with self-organizing maps. In *Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection (RAID'2003)*, LNCS v. 2820, pages 36–54, Novembre 2003.

[136] M. Roesch. Snort : Lightweight intrusion detection for networks. In *Proceedings of the 13th Conference on Systems Administration*, pages 229–238. USENIX Association, 1999.

[137] M. Roesch and C. Green. Snort Users Manual Snort Release : 2.0.1. Snort Documentation, 2003.

[138] M. Roger and J. Goubault-Larrecq. Log auditing through model checking. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 220–236, 2001.

[139] D. Rovniagin and A. Wool. The geometric efficient matching algorithm for firewalls. Technical Report EES2003-6, Dept. Electrical Engineering Systems, Tel Aviv University, 2003.

[140] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous. Survey and taxonomy of IP address lookup algorithms. *IEEE Network Magazine*, pages pp. 8–23, Mars/Avril 2001.

[141] L. J. Ryan and R. Miikkulainen. Intrusion detection with neural networks. In *Advances in Neural Information Processing Systems*, volume 10, pages 943–949, Cambridge, 1998. MA : MIT Press.

[142] G. Savchuk and S. Egorov. *Fidelis NCA Platform's Packet Capture (DirectWire)*. Fidelis Security System Inc., 2003.

[143] F. B. Schneider. *Trust in Cyberspace.* National Academy Press, 1999.

[144] M. Sebring, E. Shellhouse, M. Hanna, and R. Whitehurst. Expert Systems in Intrusion Detection : A Case Study. In *Proceedings of the 11th National Computer Security Conference*, octobre 1988.

[145] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag. A high-performance network intrusion detection system. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 8–17. ACM Press, 1999.

[146] P. Selleres. The theory and computation of evolutionary distances : pattern recocognition. *Journal of Algorithms*, 1 :359–373, 1980.

[147] U. Shankar and V. Paxson. Active mapping : Resisting NIDS evasion without altering traffic. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 44. IEEE Computer Society, 2003.

[148] T. Shimeall. Cyberterrorism. http ://www.cert.org/archive/ppt/256,1,Cyberterrorism. Carnegie Mellon Unviversity, 2002.

[149] K. Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter Conference, Dallas, Texas*, Janvier 1991.

[150] K. Sklower. A tree-based routing table for berkeley unix. In *Proceedings of the 1991 Winter Usenix Conference*, pages 93–99, 1991.

[151] S. E. Smaha. Haystack : An Intrusion Detection System. In *Fourth Aerospace Computer Security Applications Conference*, pages 37–44, Tracor Applied Science Inc., Austin, TX, Décembre 1988.

[152] R. Sommer and V. Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proceedings of the 10th ACM conference on Computer and communication security*, pages 262–271. ACM Press, 2003.

[153] D. Song. Fragrouter. http ://packetstormsecurity.nl/groups/w00w00/sectools/fragrouter/, 2000.

[154] Sourcefire. Snort^{TM} 2.0 rule optimizer. Technical report.

[155] R. Spangler. Packet sniffer detection with antisniff. Technical report, University of Wisconsin - Whitewater, Department of Computer and Network Administration, Mai 2003.

[156] L. Spitzner. Know your enemy : III. Technical report, Honeynet Project, Mars 2000.

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

[158] R. Srinivasan. RFC 1831 : RPC : Remote procedure call protocol specification version 2, aout 1995. Status : PROPOSED STANDARD.

[159] R. Srinivasan. RFC 1832 : XDR : External data representation standard, aout 1995. Status : DRAFT STANDARD.

[160] R. Srinivasan. RFC 1833 : Binding protocols for ONC RPC version 2, aout 1995. Status : PROPOSED STANDARD.

[161] V. Srinivasan, S. Suri, and G. Varghese. Packet classification using tuple space search. In *SIGCOMM*, pages 135–146, 1999.

[162] V. Srinivasan and G. Varghese. Fast address lookups using controlled prefix expansion. *ACM Transactions on Computer Systems*, 17(1) :1–40, 1999.

[163] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *Proceedings of ACM SIGCOMM '98*, pages 191–202, Septembre 1998.

[164] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *J. Compututer Security*, 10(1-2) :105–136, 2002.

[165] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids - a graph-based intrusion detection system for large networks. In *The 19th National Information Systems Security Conference*, 1996.

[166] S. Staniford-Chen, B. Tung, and D. Schnackenberg. The common intrusion detection framework (cidf). In *Information Survivability Workshop*, Orlando, Octobre.

[167] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Cost-based modeling for fraud and instrusion detection : Results from the jam project. In *Proceedings of DARPA Information Survivability Conference and Exposition.* IEEE Computer Press, 2000.

[168] A. Sundaram. An introduction to intrusion detection. *Crossroads*, 2(4) :3–7, 1996.

[169] Internet Security Systems. Signatures reference guide version 6.0, Mai 2001.

[170] G. Taleck. Ambiguity resolution via passive os fingerprinting. In *Proceedings of the 6th International Workshop on the Recent Advances in Intrusion Detection (RAID'2003)*, LNCS v. 2820, pages 192–206, Novembre 2003.

[171] K. Tan. The application of neural networks to unix computer security. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, 1995.

[172] H. S. Teng, K. Chen, and S. Lu. Security audit trail analysis using inductively generated predictive rules. In *the Sixth Conference on Artificial Intelligence Applications*, pages 24–29, Piscataway, New Jersey, Mars 1990.

[173] T. Toth. *Improving Intrusion Detection Systems.* PhD thesis, Technischen Universität Wien, Mai 2003.

[174] Z. Trabelsi, H. Rahmani, K. Kaouech, and M. Frikha. Malicious sniffing systems detection platform. In *Symposium on Applications and the Internet (SAINT 2004)*, pages 201–207, Tokyo, Japan, Janvier 2004. IEEE Computer Society 2004.

[175] P. F. Tsuchiya. A search algorithm for table entries with non-contiguous wildcarding. unpublished paper, 1991.

[176] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memory-efficient string matching algorithms for intrusion detection. In *Proceedings of the IEEE Infocom Conference*, Hong Kong, China, Mars 2004.

[177] P. Uppuluri and R. Sekar. Experiences with specification based intrusion detection. In *Proceedings of the 4th International Workshop on the Recent Advances in Intrusion Detection (RAID'2001)*, LNCS v. 2212, pages 172–189, Octobre 2001.

[178] F. Veysset, O. Coutay, and O. Heen. New tool and technique for remote operating system fingerprinting. Technical report, Intranode Research Team, Avril 2002.

[179] G. Vigna, E. Jonsson, and C. Krügel, editors. *Recent Advances in Intrusion Detection, 6th International Symposium, RAID 2003, Pittsburgh, PA, USA, Septembre 8-10, 2003, Proceedings*, volume 2820 of *Lecture Notes in Computer Science.* Springer, 2003.

[180] G. Vigna and R. Kemmerer. NetSTAT : A Network-based Intrusion Detection Approach. In *Proceedings of the* 14^{th} *Annual Computer Security Application Conference*, Scottsdale, Arizona, Décembre 1998.

[181] G. Vigna and R.A. Kemmerer. NetSTAT : A Network-based Intrusion Detection System. *Journal of Computer Security*, 7(1) :37–71, 1999.

[182] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable high speed IP routing lookups. In *Proceedings of SIGCOMM '97*, pages 25–36, Septembre 1997.

[183] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls : Alternative data models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 133–145, Mai 1999.

[184] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *Proceedings of the 3rd International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, LNCS v. 1907, pages 110–129, Octobre 2000.

[185] A. Wespi, G. Vigna, and L. Deri, editors. *Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002, Zurich, Switzerland, Octobre 16-18, 2002, Proceedings*, volume 2516 of *Lecture Notes in Computer Science*. Springer, 2002.

[186] T. Woo. A modular approach to packet classification : Algorithms and results. In *Proceedings of IEEE Infocom2000*, pages 1213–1222, 2000.

[187] M. Wood and M. Erlinger. Intrusion detection message exchange requirements, Octobre 2002. IETF Intrusion Detection Working Group Internet Draft.

[188] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical report, University of Arizona at Tuscon, Mai 1994.

université HENRI POINCARÉ
NANCY 1

Monsieur **ABBES Tarek**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY 1

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER N°1038

Nancy, le 24 décembre 2004

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY 1 * LE PRESIDENT

J.P. FINANCE

Université Henri Poincaré, Nancy I
24-30 rue Lionnois - B.P. 3069 - 54013 NANCY Cédex
Tél. : 03 83 68 20 00 - Fax : 03 83 68 21 00

Résumé

La cybercriminalité augmente de plus en plus ces dernières années ce qui pousse les administrateurs à sécuriser convenablement leurs réseaux informatiques. Ils ont recours à la détection d'intrusions pour détecter les attaques à l'entrée du réseau ou déceler les intrusions passées inaperçues à travers les pare-feux et les routeurs filtrants.

Dans cette thèse, nous nous intéressons à des problèmes sensibles rencontrés par la détection d'intrusions à savoir le haut débit, les techniques d'évasion et les fausses alertes. Afin de soutenir le haut débit, nous procédons à une division du trafic réseau ce qui permet de répartir la charge d'analyse sur plusieurs systèmes de détection d'intrusions et de sélectionner pour chaque classe du trafic la meilleure méthode de détection. Par ailleurs nous réduisons le temps de traitement de chaque paquet en organisant convenablement les règles de détection d'attaques stockées sur les systèmes de détection d'intrusions. Au cours de cette analyse nous proposons un filtrage à la volée de plusieurs signatures d'attaques. Ainsi, nous évitons le réassemblage du trafic qui était auparavant nécessaire pour résister aux techniques d'évasion. Enfin nous assurons une analyse protocolaire avec des arbres de décisions pour accélérer la détection des attaques et éviter les inconvénients du filtrage brut de motifs tels que la génération abondante des faux positifs.

Mot clés : Détection d'intrusions, division du trafic réseau, organisation des règles, filtrage à la volée des motifs, analyse protocolaire.

Abstract

Today's, because of the fast cyber-criminality development, administrators show an increasing interest towards network security. Specially they are interested by the intrusion detection in order to complement the firewall supervision and thus discover more undetected attacks.

In this dissertation we are interested by some bottlenecks that the intrusion detection faces, namely the high load traffic, the evasion techniques and the false alerts generation. In order to ensure the supervision of overloaded networks, we classify the traffic using Intrusion Detection Systems characteristics and traffic properties. Therefore each IDS supervises less IP traffic and uses less detection rules (with respect to traffics it analyses). In addition we reduce the packet processing time by a wise attack detection rules application. During this analysis we rely on a fly pattern matching strategy of several attack signatures. Thus we avoid the traffic reassembly previously used to deceive evasion techniques. Besides, we employ a protocol analysis with decision tree in order to accelerate the intrusion detection and reduce the number of false positives noticed when using a raw pattern matching method.

Keywords : Intrusion detection, network traffic division, rules organisation, on the fly pattern matching, protocol analysis.